PROCESSAMENTO DE LINGUAGEM NATURAL NA RESOLUÇÃO DE PROBLEMAS DE CLASSIFICAÇÃO

NATURAL LANGUAGE PROCESSING IN SOLVING CLASSIFICATION PROBLEMS

Giovani Müller VRECH
giovanivrech@gmail.com
Ciência da Computação, Centro Universitário Padre Anchieta, Jundiaí-SP

Prof. Rodrigo Kiyoshi SAITO
rodrigok@anchieta.br
Ciência da Computação, Centro Universitário Padre Anchieta, Jundiaí-SP

Prof. Dr. Carlos Eduardo CÂMARA
ccamara@anchieta.br
Ciência da Computação, Centro Universitário Padre Anchieta, Jundiaí-SP

RESUMO

Neste trabalho será desenvolvido um modelo de *Machine Learning* para resolução de um problema de classificação. O modelo a ser desenvolvido se encontra na área de *Processamento de Linguagem Natural* (PLN) que classificará um título de notícia em sua respectiva categoria; o modelo que obteve o melhor desempenho para esta solução chegou a uma acurácia de 82% neste projeto. Durante o desenvolvimento serão utilizados os dois modelos conhecidos de *Word2Vec,* sendo eles os modelos *Continuous Bag-of-Words* e *Continuous Skip-Gram.* Para implementação deste modelo de Inteligência Artificial, foi tomado como base um modelo já implementado pelo desenvolvedor Thiago Gonçalves Santos, que pode ser encontrado nesse link: https://github.com/alura-cursos/word2vec_treinamento.

Palavras-Chave: Inteligência Artificial; *Machine Learning*; *Processamento de Linguagem Natural; Word2Vec; Continuous Bag-of-Words; Continuous Skip-Gram.*

ABSTRACT

In this work, a Machine Learning model will be developed to solve a classification problem. The model to be developed is in the area of *Natural Language Processing* (PLN) that will classify a news headline in its respective category, the model that obtained the best performance for this solution, reached an accuracy of 82% in this project. During the development the two known Word2Vec models will be used, being the Continuous Bag-of-Words and Continuous Skip-Gram models. To implement this Artificial Intelligence model, it was taken as a base a model already implemented by the developer Thiago Gonçalves Santos, which can be found in this link: https://github.com/alura-cursos/word2vec_treinamento.

Keywords: Artificial Intelligence; *Machine Learning*; *Natural Language Processing; Word2Vec; Continuous Bag-of-Words; Continuous Skip-Gram*

1. INTRODUÇÃO

O conceito de Redes Neurais vem chamando muita atenção nos últimos anos devido ao grande poder que ela tem e às coisas incríveis que ela faz. Entretanto, essa tecnologia vem sendo estudada há décadas, mais especificamente desde o final da Segunda Guerra Mundial.

"*Em 1943, Warren McCulloch e Walter Pitts apresentam um artigo que fala pela primeira vez de redes neurais, estruturas de raciocínio artificiais em forma de modelo matemático que imitam o nosso sistema nervoso*". (KLEINA, 2018). Desde então, o ser humano vem buscando novos casos de usos de implementação de redes neurais e uma forma de facilitar a interação humano-máquina, abordando uma das subáreas importantes da Inteligência Artificial (IA) conhecida como Processamento de Linguagem Natural (PLN). (RODRIGUES, 2017).

Essa área estuda as capacidades e as limitações de uma máquina entender a linguagem humana. Esse conceito parece simples, mas é algo muito mais complexo, pois quando dizemos que a máquina deve "entender" a nossa linguagem, estamos afirmando que ela deve "*reconhecer o contexto, fazer análise léxica e morfológica, análise sintática, criar resumos, extrair informação, interpretar os sentidos, analisar sentimentos e até aprender conceitos com os textos processados."* (RODRIGUES, 2017).

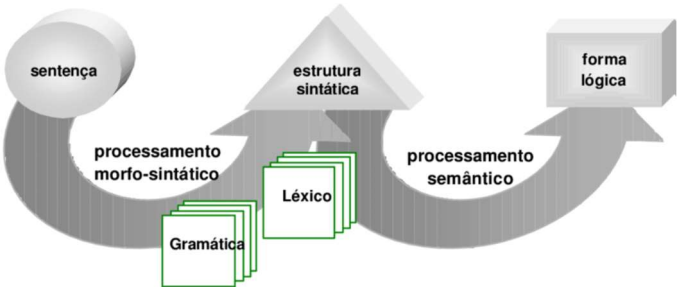
O PLN possui diversas aplicações em diversas áreas de pesquisas, tais como: tradução automática, *chat-bots*, assistentes virtuais, detecção de sentimentos etc. (RODRIGUES, 2017). Essa área também faz o uso de inteligência artificial na análise de dados textuais. Uma dessas aplicações seria a interpretação textual, como é citado por Gonçalves: "*fazer uso do machine learning para ler e compreender textos, tendo ainda a aptidão para responder perguntas baseadas em algumas passagens presentes no conteúdo."* (GONÇALVES, 2018). Dessa maneira, além do modelo conseguir ler e compreender o texto, essa aplicação também pode ser utilizada em tradutores, *chat-bots* etc.

2. PROCESSAMENTO DE LINGUAGEM NATURAL

O Processamento de Linguagem Natural (PLN) é uma subárea da ciência da computação, da linguística e da inteligência artificial, como é explicado por Vela e Gonzalez:

> *"trata computacionalmente os diversos aspectos da comunicação humana, como sons, palavras, sentenças e discursos, considerando formatos e referências, estruturas e significados, contextos e usos. Em sentido bem amplo, podemos dizer que o PLN visa fazer o computador se comunicar em linguagem humana, nem sempre necessariamente em todos os níveis de entendimento e/ou geração de sons, palavras, sentenças e discursos."* (VELA e GONZALEZ, 2003, p. 3)

Figura 1 – Transformação da sentença na estrutura sintática e na forma lógica



Fonte: VELA e GONZALEZ, 2003, p. 3.

2.1 NÍVEIS DE CONHECIMENTO

Existem alguns níveis de conhecimento do quais o computador necessita para entender um algoritmo PLN com maior assertividade, "*para que um sistema de PLN seja robusto necessita de, pelo menos, contemplar algumas tarefas base. Cada uma dessas tarefas concentra-se em resolver parte de um problema maior* **".** (PINTO, 2015, p. 6).

Entre esses níveis, está a análise sintática, que visa compreender as relações entre as palavras – "*A análise sintática, ou parsing, pretende estudar a relação entre as palavras na frase. Permite, por exemplo, reconhecer que um determinado adjetivo está a classificar um determinado nome numa frase*." (PINTO, 2015, p. 6). Este modelo de análise será utilizado no algoritmo para desenvolvimento da *Word Embeddings.*

Outro desses níveis é a análise semântica, que tem como objetivo entender o significado de uma palavra em um texto – "*pretende clarificar o significado das palavras num texto. Após esta análise obtém-se um texto em linguagem formal, passível de ser compreendido por um computador, ou seja, um texto sem ambiguidade, pois só assim é possível representá-lo em linguagem formal.*" (PINTO, 2015, p. 6). Esta análise será utilizada no algoritmo para desenvolvimento da *Word Embeddings*.

A análise pragmática tenta estabelecer uma relação entre a linguagem e o contexto – "*A pragmática defende que o significado do texto também depende do contexto e do conhecimento prévio entre as partes envolvidas*." (PINTO, 2015, p. 6)

Esta análise será utilizada neste trabalho para criar uma "memória" no modelo, fornecendo para o algoritmo não só a palavra, mas também o contexto.

2.2 WORD2VEC

"*Word2Vec é um método estatístico para aprender eficientemente um Word Embedding independente, a partir de um corpus de texto*." (TOMALOK, 2019).

Esta nova técnica foi desenvolvida por Tomas Mikolov e colaboradores no Google em 2013 (TOMALOK, 2019). Antes do modelo *Word2Vec* proposto por Mikolov, existia uma técnica chamada de *One-Hot Encoding*, como explica Tomalok:

> *"Por muito tempo a técnica utilizada para expressar palavras em um formato que o computador pudesse entender, foi utilizar uma matriz de coocorrência. Desta forma,*

cada palavra alvo presente no texto ou frase, era representada no formato binário; onde 0 representa que a palavra atual no vetor não é a palavra alvo, e 1 indica que a posição contém a palavra alvo". (TOMALOK, 2019).

Figura 2 – Exemplo de uma representação vetorial utilizando a técnica One-Hot Encondig

	O	cachorro	corre	atrás	do	gato	.	O	gato	corre	atrás	do	rato	.
O	1	0	0	0	0	0	0	1	0	0	0	0	0	0
cachorro	0	1	0	0	0	0	0	0	0	0	0	0	0	0
corre	0	0	1	0	0	0	0	0	0	1	0	0	0	0
atrás	0	0	0	1	0	0	0	0	0	0	1	0	0	0
do	0	0	0	0	1	0	0	0	0	0	0	1	0	0
gato	0	0	0	0	0	1	0	0	1	0	0	0	0	0
.	0	0	0	0	0	0	1	0	0	0	0	0	0	1
rato	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Fonte: TOMALOK, 2019.

A Figura 2 mostra como a técnica *One-Hot Encondig* cria a sua representação vetorial da frase "O cachorro corre atrás do gato. O gato corre atrás do rato.". Pode-se reparar que para cada palavra da frase foi criada uma coluna com a posição onde a palavra se encontra na frase.

Percebe-se que essa técnica é limitada, pois como se trata de um vetor horizontal, só é possível extrair informações de frequência de cada palavra na frase. (TOMALOK, 2019).

Em seu artigo, Mikolov e colaboradores salientam problemas das técnicas que existiam na época:

> *"However, the simple techniques are at their limits in many tasks. For example, the amount of relevant in-domain data for automatic speech recognition is limited - the performance is usually dominated by the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques"*. (MIKOLOV et al., 2013)

Com isso, surge o modelo *Word2Vec* e seu novo jeito de gerar a *Word Embeddings* por Redes Neurais, que será explicado com mais detalhes no próximo tópico.

2.3 WORD EMBEDDINGS

Word Embeddings são representações vetoriais de uma palavra, onde cada palavra recebe um valor e esse valor é usado como peso para agrupar palavras similares, como explicado por Liu e colaboradores:

> *"Word embedding, also known as word representation, plays an increasingly vital role in building continuous word vectors based on their contexts in a large corpus.*

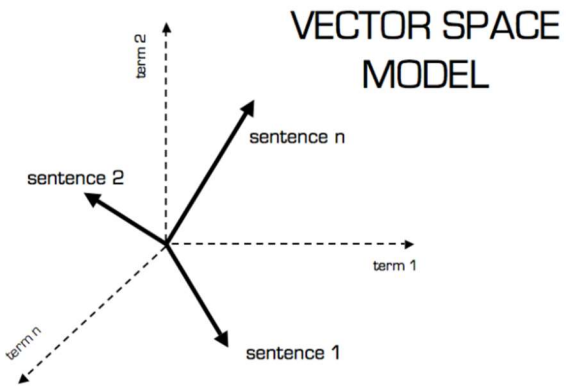
> *Word embedding captures both semantic and syntactic information of words, and can be used to measure word similarities, which are widely used in various IR and NLP tasks".* (LIU; LIU; CHUA; SUN, 2015, p. 1(2418))

Em seu artigo, Mikolov e colaboradores propõem dois novos modelos de *Word Embeddings*, o modelo *Continuous Bag-of-Words* e o modelo *Continuous Skip-gram*, que serão explicados com detalhes nos próximos tópicos.

Diferente da técnica vista anteriormente (*One-Hot Encondig*), a técnica proposta por Mikolov utiliza uma Rede Neural para gerar a representação vetorial, assim conseguindo mais características das palavras, como explicado por Tomalok:

> *"Agora, a grande sacada foi a seguinte: ao invés de passar apenas informações ao modelo da frequência de cada palavra no texto, foi fornecido muito mais informações que o modelo poderá explorar para aumentar seu "aprendizado" sobre o conjunto de texto".* (TOMALOK, 2019).

Figura 3 – Representação de texto através de coordenadas



Fonte: TOMALOK, 2019.

A Figura 3 mostra como a técnica de Mikolov funciona:

> *"um mapa de vetores multidimensional, que é uma forma algébrica de representar o texto através de coordenadas, onde cada vetor irá representar uma palavra/sentença; desta forma conseguimos utilizar a matemática a nosso favor, para efetuarmos cálculos, tais como a distância entre cada vetor, e diversas outras operações utilizando as propriedades dos vetores".* (TOMALOK, 2019).

Dessa forma, a representação vetorial de cada palavra recebe um número em ponto flutuante onde as palavras mais próximas podem ser agrupadas, e também é possível utilizar operações matemáticas para encontrar palavras similares, por exemplo: (mulher + professor) – homem = professora. Para visualizar com mais clareza o agrupamento de palavras similares, pode-se utilizar esse link: https://projector.tensorflow.org/.

2.3.1 CONTINUOUS BAG-OF-WORDS E CONTINUOUS SKIP-GRAM

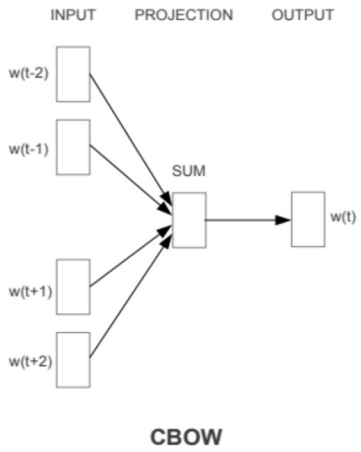
Uma das técnicas que existem dentro do modelo *Word2Vec* proposto por Mikolov foi o *Continuous Bag-of-Words*. Nesta técnica é fornecido para a camada de entrada da Rede Neural o contexto, e a camada de saída será a palavra alvo, por exemplo: imagine que é fornecido para a camada de entrada o contexto "EU – GOSTO – DO – LIVRO – *X*", onde *X* vai ser a palavra alvo, ou seja, a arquitetura *Bag-of-Words* irá retornar qual palavra pode ser utilizada de acordo com as palavras de contextos fornecidas, como explica Mikolov em seu artigo:

> *"The first proposed architecture is similar to the feedforward NNLM, where the non-linear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words in the history does not influence the projection. Furthermore, we also use words from the future; we have obtained the best performance on the task introduced in the next section by building a log-linear classifier with four future and four history words at the input, where the training criterion is to correctly classify the current (middle) word"*. (MIKOLOV et al., 2013)

Já o *Continuous Skip-Gram* é a outra técnica existente dentro do modelo *Word2Vec* proposta por Mikolov. Nesta técnica é fornecida para a camada de entrada da Rede Neural a palavra alvo e a camada de saída é (será) o contexto em que essa palavra se encontra, como explica Mikolov:

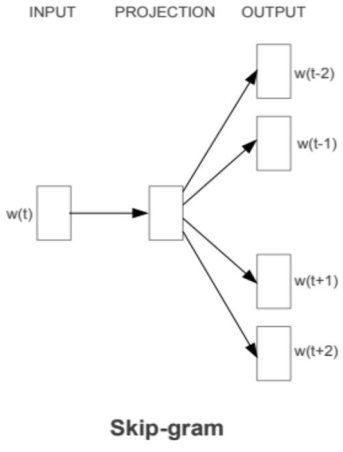
> *"The second architecture is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. We found that increasing the range improves quality of the resulting word vectors, but it also increases the computational complexity. Since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those words in our training examples"*. (MIKOLOV et al., 2013)

Figura 4 – Modelo *Continuous Bag-of-Words*



Fonte: MIKOLOV et al., 2013.

Figura 5 – Modelo *Continuous Skip-Gram*



Fonte: MIKOLOV et al., 2013.

3. CONCEITOS IMPORTANTES

Para entendimento completo e claro de todo algoritmo adaptado neste trabalho, será necessário entender alguns conceitos importantes dentro do mundo de *Machine Learning*.

3.1 DATASETS

Datasets são os conjuntos de dados que são utilizados na análise de dados; geralmente esses conjuntos possuem milhares de registros (dados) sobre um determinado assunto e são organizados em linhas e colunas. Os formatos mais utilizados nesses conjuntos de dados são: .csv, .txt, .xls e json.

Na maior parte do tempo, os conjuntos de dados não vêm em um formato de *dataset*, às vezes os dados vêm bagunçados, sem limpeza, sem correlações etc. Como citado no site *Aquarela*, segue um exemplo de conjunto de dados **não** organizado em formato de *dataset* e um exemplo organizado:

Figura 6 – Conjunto não organizado em *dataset*

UNIDADES VENDIDAS	VENDAS	CUSTOS TOTAIS	LUCRO/ PREJUÍZO
0	R$0	R$8.140	-R$8.140
15	R$1.155	R$8.335	-R$7.180
30	R$2.310	R$8.530	-R$6.220
45	R$3.465	R$8.725	-R$5.260
60	R$4.620	R$8.920	-R$4.300
75	R$5.775	R$9.115	-R$3.340
90	R$6.930	R$9.310	-R$2.380
105	R$8.085	R$9.505	-R$1.420
120	R$9.240	R$9.700	-R$460
135	R$10.395	R$9.895	R$500
150	R$11.550	R$10.090	R$1.460
165	R$12.705	R$10.285	R$2.420
180	R$13.860	R$10.480	R$3.380
195	R$15.015	R$10.675	R$4.340
210	R$16.170	R$10.870	R$5.300
225	R$17.325	R$11.065	R$6.260
240	R$18.480	R$11.260	R$7.220
255	R$19.635	R$11.455	R$8.180
270	R$20.790	R$11.650	R$9.140
285	R$21.945	R$11.845	R$10.100
300	R$23.100	R$12.040	R$11.060

Fonte: Site Aquarela[1].

Figura 7 – Conjunto organizado em *dataset*

unidades vendidas	vendas	Custos Totais	lucro/prejuízo
0	0	8.140	-8.140
15	1155	8.335	-7.180
30	2310	8.530	-6.220
45	3465	8.725	-5.260
60	4620	8.920	-4.300
75	5775	9.115	-3.340
90	6930	9.310	-2.380
105	8085	9.505	-1.420
120	9240	9.700	-460
135	10395	9.895	500
150	11550	10.090	1.460
165	12705	10.285	2.420
180	13860	10.480	3.380
195	15015	10.675	4.340
210	16170	10.870	5.300
225	17325	11.065	6.260
240	18480	11.260	7.220
255	19635	11.455	8.180
270	20790	11.650	9.140
285	21945	11.845	10.100
300	23100	12.040	11.060

Fonte: Site Aquarela.

3.2 MODELOS DE MACHINE LEARNING

Como é explicado na documentação de *Machine Learning* da *Microsoft*:

> *"Um modelo de machine learning é um arquivo que foi treinado para reconhecer determinados tipos de padrões. Você treina um modelo em um conjunto de dados, fornecendo a ele um algoritmo que pode ser usado para ponderar e aprender com esses dados"*. (Windows Machine Learning, 2019).

[1] Disponível em: <https://www.aquare.la/datasets-o-que-sao-e-como-utiliza-los/>.

Para realizar o treinamento existem diversas formas, metodologias e algoritmos, dependendo do problema que está sendo tratado; os problemas podem variar de classificação até de regressão, e para resolver esse problema, cabe ao desenvolvedor do modelo decidir qual algoritmo e método será implementado para a resolução.

No processo de treinamento do modelo, é possível ser criada uma solução do zero, ou seja, sem a utilização de alguma ferramenta já pronta, mas também é possível utilizar alguma arquitetura de algoritmo que já está em funcionamento (apenas adaptando a arquitetura para o seu problema); para isso, existem diversas bibliotecas disponíveis; entre as mais famosas, pode-se citar a biblioteca de código aberta chamada *Scikit-learn,* que é uma biblioteca Python que possui diversas ferramentas que auxiliam no treinamento de um modelo. Essa biblioteca ajuda a resolver diversos tipos de problemas, tais como: *Classification*, *Regression*, *Clustering*, *Dimensionality reduction* etc.

Após realizar o treinamento do modelo, é possível utilizá-lo com dados que ele nunca viu antes, ou seja, com dados reais, dados que serão fornecidos como entrada no dia a dia para que ele faça previsões, como é citado na documentação de *Machine Learning* do *Windows*:

> *"Depois de treinar o modelo, você pode usá-lo para ponderar dados que ele não viu antes e fazer previsões sobre esses dados. Digamos, por exemplo, que você deseja criar um aplicativo capaz de reconhecer as emoções de um usuário com base nas expressões faciais. É possível treinar um modelo fornecendo-lhe imagens de rostos marcadas com determinada emoção e, em seguida, usar esse modelo em um aplicativo que consiga reconhecer a emoção de qualquer usuário".* (Windows Machine Learning, 2019).

3.3 EXPLORAÇÃO DE PARÂMETROS NO MODELO

Quando são utilizados modelos de *Machine Learning*, existem diversos parâmetros que podem ser fornecidos para o modelo, na tentativa de conseguir obter uma melhoria em seu desempenho. Testar, modificar e validar cada parâmetro de cada modelo não é uma tarefa simples, já que as possibilidades de combinação são inúmeras, por exemplo:

Figura 8 – Exemplo de parâmetros em um modelo

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0)  [source]
```

Fonte: Documentação no site *Scikit-Learn*[2].

Na Figura 8 pode ser vista a quantidade de parâmetros e possibilidades que existem no modelo *Decision Tree Classifier;* além disso, em cada parâmetro pode existir uma ou mais variações, por exemplo:

[2] Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>.

- ***criterion*: "**gini" ou "entropy".
- ***max_depth*:** Qualquer número inteiro ou nada.
- ***min_samples_split*:** Qualquer número inteiro ou ponto flutuante.

Explorar todo esse espaço de possibilidades testando uma combinação de cada vez tomaria muito tempo. Para resolver esse problema, será utilizada a função *RandomizedSearchCV* da biblioteca *Scikit-learn*; essa função utiliza a validação cruzada para testar as configurações fornecidas nos parâmetros. Ao contrário da função *GridSearchCV*, ela não irá testar todos os valores dos parâmetros, mas um número fixo de valores, definido antes, como é explicado na própria documentação da função na biblioteca do *Scikit-learn* – "*In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n_iter*."
(https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

3.4 MÉTRICAS PARA VALIDAÇÃO DO MODELO

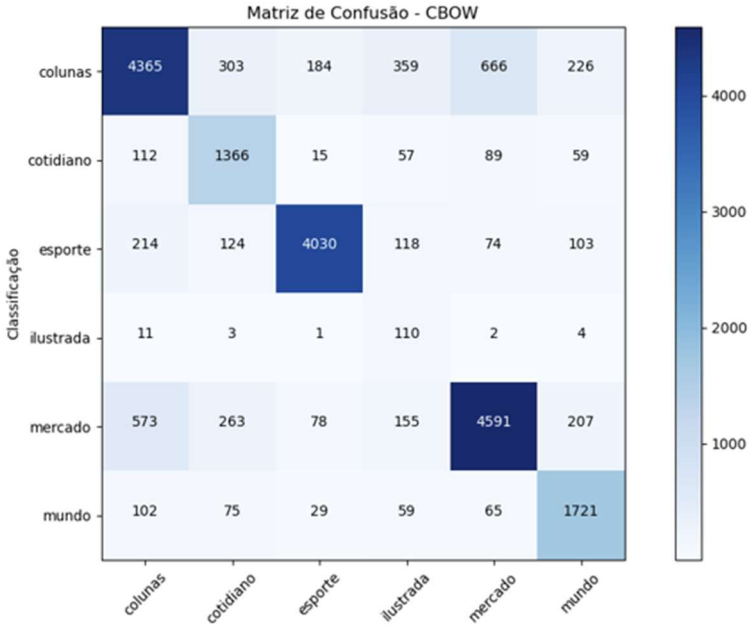
Após realizar a implementação do algoritmo para resolver o problema em questão e utilizar esse modelo com dados que ele nunca viu antes, é necessário utilizar algumas métricas para verificar se o modelo é viável ou não para ser utilizado. Neste trabalho só serão citadas as métricas que foram realizadas nas validações dos modelos.

3.4.1 MATRIZ DE CONFUSÃO

A matriz de confusão é uma tabela que mostra qual foi a taxa de classificação do modelo para cada classe do problema em questão; dessa forma, é possível validar os acertos e erros do modelo. Esta tabela tem como objetivo retornar os seguintes valores:

- ***True Positive* – TP:** Ocorre quando no valor a ser previsto a classe buscada foi prevista corretamente. Exemplo: A resposta deveria ser "sim", e o modelo classificou como "sim".
- ***False Positive* – FP:** Ocorre quando no valor a ser previsto a classe buscada foi prevista incorretamente. Exemplo: A resposta deveria ser "sim", e o modelo classificou como "não".
- ***True Negative* – TN:** Ocorre quando no valor a ser previsto a classe que não está sendo buscada foi prevista corretamente. Exemplo: A resposta deveria ser "não", e o modelo classificou como "não".
- ***False Negative* – FN:** Ocorre quando no valor a ser previsto a classe que não está sendo buscada foi prevista incorretamente. Exemplo: A resposta deveria ser "não", e o modelo classificou como "sim".

Figura 9 – Exemplo de uma matriz de confusão



Fonte: VRECH, 2020.

Na Figura 9 é mostrado um exemplo de matriz de confusão retirada de um dos modelos analisados neste trabalho, que será explicada com mais detalhes posteriormente. Para este gráfico em específico, existe uma régua de temperatura ao lado; essa régua serve para orientar em relação à frequência de certas classificações previstas pelo modelo (quanto mais escura for a cor, mais acertos o modelo obteve). Repare que neste modelo foram classificados 4.365 títulos de notícias na categoria de "colunas", e ele realmente era uma categoria de "colunas", mas também foram classificados 303 títulos de "colunas" como "cotidiano", e assim por diante.

3.4.2 ACURÁCIA

A acurácia mostra a porcentagem de identificações positivas que estavam realmente positivas, como citado no *Machine Learning Crash Course*[3] – "*Informally,* ***accuracy*** *is the fraction of predictions our model got right.*" A seguir, será mostrada a representação matemática para a forma da acurácia e a forma para classificações binárias:

Representação forma da acurácia

$$Accuracy = \frac{\text{Number of correct predictions}}{Total\ number\ of\ predictions}$$

Fonte: Machine Learning Crash Course.

Classificação binária:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+F}$$

Fonte: Machine Learning Crash Course.

[3]Curso intensivo de Machine Learning fornecido pelo Google: https://developers.google.com/machine-learning/crash-course/classification/accuracy.

Figura 10 – Exemplo do cálculo de acurácia

True Positive (TP): • Reality: Malignant • ML model predicted: Malignant • Number of TP results: 1	False Positive (FP): • Reality: Benign • ML model predicted: Malignant • Number of FP results: 1
False Negative (FN): • Reality: Malignant • ML model predicted: Benign • Number of FN results: 8	True Negative (TN): • Reality: Benign • ML model predicted: Benign • Number of TN results: 90

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1 + 90}{1 + 90 + 1 + 8} = 0.91$$

Fonte: Site do Machine Learning Crash Course.

A Figura 10 mostra uma matriz de confusão e o cálculo de acurácia para um modelo que classificou 100 tumores como malignos (classe positiva) e benignos (classe negativa). No exemplo, a acurácia chega em 0.91 ou 91%. Mas isso significa que o modelo apresentado é excelente? Não necessariamente, pois não se pode levar em consideração apenas a acurácia final, é preciso analisar outras métricas, como é explicado no *Machine Learning Crash Course*:

> *"Of the 91 benign tumors, the model correctly identifies 90 as benign. That's good. However, of the 9 malignant tumors, the model only correctly identifies 1 as malignant—a terrible outcome, as 8 out of 9 malignancies go undiagnosed*!"

3.4.3 RECALL

O *Recall* mostra a porcentagem de *True Positive* que foram classificados corretamente; a representação matemática do Recall é definida por:

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

Fonte: Site do Machine Learning Crash Course.

O *Machine Learning Crash Course* também fornece um exemplo de cálculo de *Recall* no problema visto anteriormente (classificação de tumores). Segue:

Figura 11 – Exemplo de matriz de confusão e cálculo do Recall

True Positives (TPs): 1	False Positives (FPs): 1
False Negatives (FNs): 8	True Negatives (TNs): 90

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{1}{1 + 8} = 0.11$$

Fonte: Site do *Machine Learning Crash Course.*

Neste exemplo, o modelo teve um *Recall* de 0.11 ou 11%, ou seja, o modelo apresentado identifica corretamente 11% de todos os tumores malignos.

3.4.4 F-SCORE

O *F-Score* ou *F1-Score* mostra uma média entre a acurácia e o *Recall* – "*The F-score is a way of combining the precision and recall of the model, and it is defined as the harmonic mean of the model's precision and recall*." (WOOD)

Representação matemática do *F-Score*:

$$2 * \frac{precision * recall}{precision + recall}$$

Fonte: Site do *Machine Learning Crash Course.*

3.4.5 CURVA AUC-ROC

A curva ROC (*Receiver Operanting Characteristic)* é um método gráfico para avaliação da classificação de classes previstas por um modelo. Esta curva possui dois parâmetros: taxa de *True Positive* e taxa de *False Positive*.

TPR, representação matemática:

$$TPR = \frac{TP}{TP+FN}$$

Fonte: *Machine Learning Crash Course.*

FPR, representação matemática:

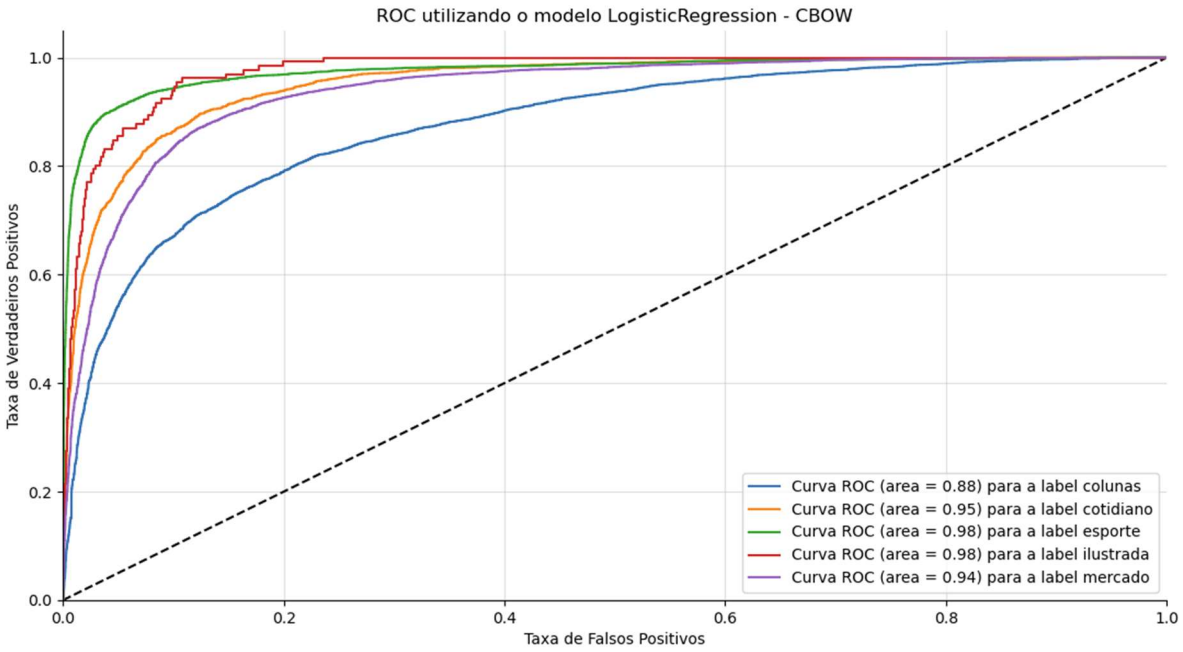
$$FPR = \frac{FP}{FP+T}$$

Fonte: *Machine Learning Crash Course.*

O AUC representa uma medida de separação. Com essa curva pode-se dizer o quanto o modelo é capaz de diferenciar cada classe proposta; quanto mais alto o AUC, melhor será o modelo na hora da predição de classes, como é citado no *Machine Learning Crash Course:*

> *"AUC stands for 'Area under the ROC Curve.' That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1)".*

Figura 12 – Exemplo de curva AOC-ROC



Fonte: VRECH, 2020.

A Figura 12 é um exemplo de curva AOC-ROC retirada de um dos modelos gerados neste trabalho; quanto mais próxima de 1.0 a curva da classe, melhor para o modelo proposto.

3.4.6 MODELO DUMMY CLASSIFIER

O *Dummy Classifier* é apenas um modelo de *Machine Learning* usado para se obter uma linha de base para validação do modelo real, ou seja, é um modelo gerado com regras simples e sem muita complexidade, logo, o modelo real precisa ter um desempenho melhor que um modelo *Dummy*; estes tipos de modelos não devem ser utilizados em problemas reais, como explica a própria documentação do Dummy Classifier no *Scikit-learn* – "*DummyClassifier is a classifier that makes predictions using simple rules. This classifier is useful as a simple baseline to compare with other (real) classifiers. Do not use it for real problems*.".

4. ADAPTAÇÃO DO MODELO DE PLN

O projeto inicial foi retirado do repositório do desenvolvedor Thiago Gonçalves Santos em seu *GitHub*[4], que pode ser encontrado neste link: https://github.com/alura-cursos/word2vec_treinamento.

O objetivo da implementação desse algoritmo de *Machine Learning* é realizar a classificação de títulos de notícias em suas respectivas categorias utilizando técnicas de Processamento de Linguangem Natural.

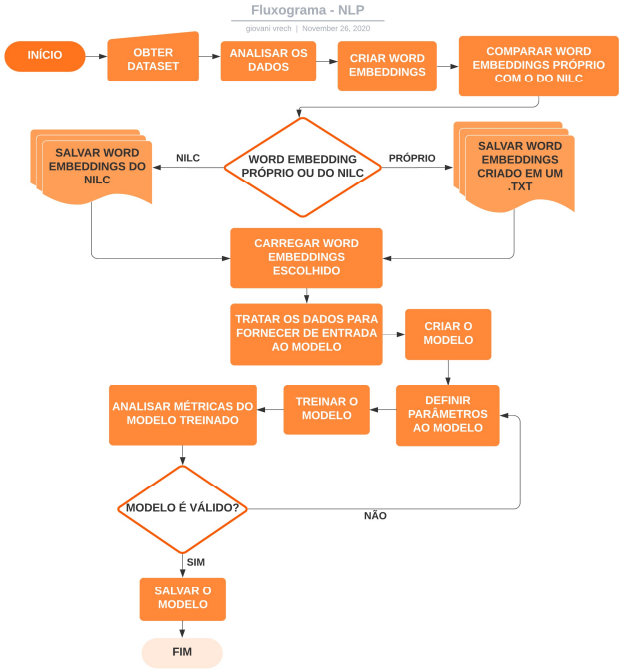
[4] Plataforma de hospedagem de código-fonte para controle de versionamento; é permitida a contribuição em projetos privados ou Open Source.

Para este trabalho serão mostrados códigos desenvolvidos na linguagem Python, utilizando o *Google Colaboratory*[5] para uma análise e treinamento inicial mais rápida e simples; também será utilizado o *Visual Studio Code*[6] para o desenvolvimento final da aplicação.

4.1 FLUXOGRAMA

Na Figura 13 é apresentado o fluxo de desenvolvimento desse modelo para um entendimento mais claro e simples do que está sendo feito.

Figura 13 – Fluxograma da implementação do modelo de NLP



Fonte: VRECH, 2020.

4.2 DATASET UTILIZADO PARA MODELO DE NLP

O conjunto de dados utilizado no desenvolvimento desse modelo está dividido em dois arquivos CSV, sendo um deles para os dados de treino e o outro para os dados de testes (não sendo necessário utilizar a função *train_test_split* [7] do *Scikit-learn*).

[5] Serviço em nuvem desenvolvido pelo Google e disponibilizado de forma gratuita para desenvolvimento de aplicações em Python, bastante utilizado na área de análise de dados.
[6] Editor de código desenvolvido pela Microsoft.
[7] Função que se encontra na biblioteca do *Scikit-learn*, que tem como objetivo dividir arrays ou matrizes de forma randômica para gerar subconjuntos para os dados de teste.

4.2.1 ANÁLISE DOS DADOS

O primeiro passo é carregar os dados (Figura 14), e para analisar os mesmos, é de extrema importância entender como os dados estão compostos e tentar gerar insights que ajudarão no desenvolvimento do modelo. Será observado na Figura 15 o formato dos dados, onde o conjunto de treino possui 90.000 linhas por 6 colunas.

Figura 14 – Carregamento dos dados

```
[5]  1 data_train = pd.read_csv("/content/drive/My Drive/IA/TCC/Data/train.csv")
     2 data_test = pd.read_csv("/content/drive/My Drive/IA/TCC/Data/test.csv")
```

Fonte: VRECH, 2020.

Figura 15 – Formato do conjunto

```
[8]  1 data_train.shape

     (90000, 6)
```

Fonte: VRECH, 2020.

Figura 16 – Colunas X linhas do conjunto de dados

```
[9]  1 data_train.tail()
```

	title	text	date	category	subcategory	link
89995	Mural: Há 30 anos, aeroporto não foi bem receb...	Década de 1970. Congonhas já estava superlotad...	2015-01-22	cotidiano	NaN	http://www1.folha.uol.com.br/cotidiano/2015/01...
89996	As notícias sobre Schumacher não são boas, diz...	O ex-presidente da Ferrari Luca di Montezemolo...	2016-04-02	esporte	NaN	http://www1.folha.uol.com.br/esporte/2016/02/1...
89997	De olho em R$ 50 bilhões, governo pode concede...	Para fazer caixa, o governo estuda conceder pa...	2017-08-29	mercado	NaN	http://www1.folha.uol.com.br/mercado/2017/08/1...
89998	Moro deu a Lula o papel de coitadinho	Realizou-se parcialmente o primeiro objetivo d...	2016-06-03	colunas	eliogaspari	http://www1.folha.uol.com.br/colunas/eliogaspa...
89999	Velocidade da aprovação das reformas tem 'exce...	O Banco Central afirmou que a velocidade da ap...	2016-10-25	mercado	NaN	http://www1.folha.uol.com.br/mercado/2016/10/1...

Fonte: VRECH, 2020.

Na Figura 16 são mostradas as colunas existentes no *dataset* de treino e alguns registros aleatórios que existem nesses dados. Aqui pode-se ter uma ideia de como os dados estão registrados, quais colunas não serão necessárias para esse modelo, qual será a coluna a ser prevista e como deverão ser tratados os dados textuais. Para o problema proposto neste modelo, serão utilizadas apenas as colunas *title* e *category*, já que o objetivo é dizer qual é a categoria de acordo com o título da notícia.

Figura 17 – Registros únicos na coluna de category

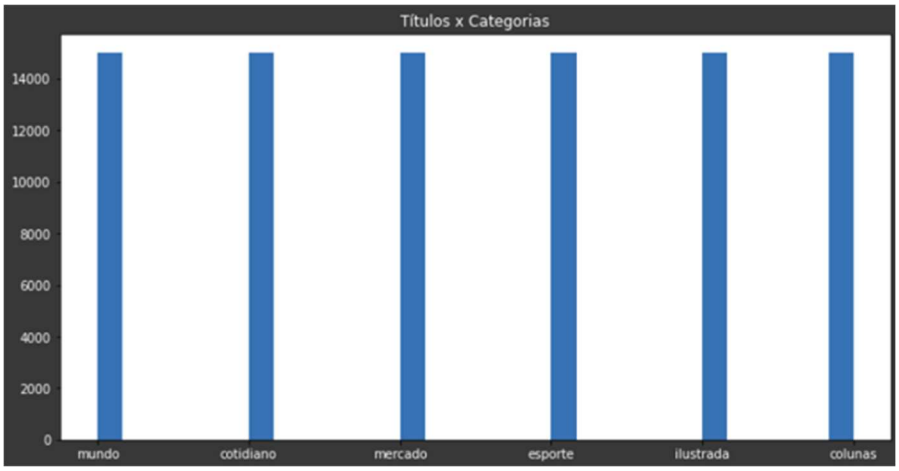
```
1 data_train['category'].unique()

array(['mundo', 'cotidiano', 'mercado', 'esporte', 'ilustrada', 'colunas'],
      dtype=object)
```

Fonte: VRECH, 2020.

A coluna *category* presente nesse conjunto de dados irá ser a coluna que o modelo tentará predizer, ou seja, será a saída do modelo. Logo, na Figura 17 são mostrados quais sãos os valores únicos presentes nesta coluna, ou seja, os valores que não se repetem. Dessa forma, é possível analisar quais serão as prováveis saídas do modelo.

Figura 18 – Relação de títulos e categorias no conjunto de dados para treino



Fonte: VRECH, 2020.

A Figura 18 mostra a quantidade de títulos de notícias existentes em cada categoria. Neste exemplo, pode-se ver que existe a mesma quantidade de títulos distribuídos igualmente entre as categorias.

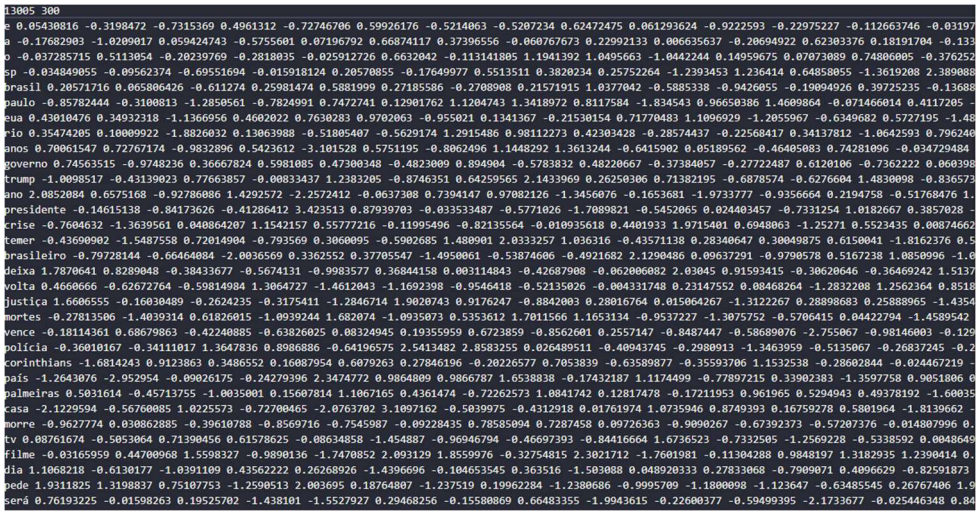
4.3 CRIAÇÃO DO WORD EMBEDDINGS

Para esse projeto, foi testada a adaptação de um *Word Embeddings* próprio, na tentativa de uma melhor acurácia no modelo. Para isso, primeiramente é necessário tratar os dados, convertendo todos os valores textuais para minúsculo, removendo as *Stop Words*[8] e removendo valores não numéricos, por exemplo: fornecendo a frase "Modelo de TraTamEntO 12312 $$ %%]!!dos teXtos!!" e aplicando todos os conceitos falados anteriormente, será obtida como saída a frase "modelo tratamento textos"; logo, pode-se observar que o tratamento removeu as *Stop Words* "de" e "dos", também deixou o texto em minúsculas e removeu tudo que não era valor textual. Após esse processo, é necessário realizar a *tokenização* do texto, ou seja, separar cada palavra do texto, deixando-as isoladas, assim criando um *array* de palavras.

Logo após aplicar todos esses tratamentos em todo o conjunto de dados, é necessário fornecer esses novos dados para o treinamento e construção do vocabulário de *Word Embeddings*. Após realizar a construção do vocabulário no formato *Word2Vec* e utilizando os modelos *Continuous Bag-of-Words* e *Continuous-Skip Gram*, obtém-se o seguinte exemplo de arquivo .txt, apresentado na Figura 19.

[8] São palavras que não agregam valor no texto e podem ser removidas, tais como: com, a, e, os, de etc.

Figura 19 – Exemplo do arquivo que contém o Word Embeddings



Fonte: VRECH, 2020.

4.3.1 COMPARAÇÃO DO WORD EMBEDDINGS PRÓPRIO COM O DO NILC

O NILC, ou Núcleo Interinstitucional de Linguística Computacional, fornece para os desenvolvedores e pesquisadores da área um repositório contendo um armazenamento gigante de *Word Embeddings*. Para o repositório foram utilizados 17 corpora diferentes, que totalizam no final 1.395.926.282 *tokens*. Dentro desses corpora há textos retirados de *Wikipedia*, *GoogleNews*, SARESP, FAPESP etc.

Estes *Word Embeddings* fornecidos pelo NILC possuem diversos formatos, incluindo o *Word2Vec* com os modelos *Continuous Bag-of-Words* e *Continuous-Skip Gram*, que são os mesmos utilizados neste projeto.

Na comparação dos *Word Embeddings* fornecidos pelo NILC com os *Word Embeddings* gerados neste trabalho, pôde-se observar que o vocabulário obtido do NILC teve uma melhora de 1% em alguns modelos, provavelmente pela grande quantidade e variação de *tokens* dentro da representação vetorial. Por esse motivo, neste modelo serão utilizados o *Word Embeddings* CBOW e SKIP-GRAM, ambos com 300 dimensões, fornecidos pelo NILC.

4.4 PROPOSTA DE MODELOS

Neste projeto foram analisados diversos modelos de *Machine Learning* e testados com diferentes variações de parâmetros, na tentativa de obter um resultado cada vez melhor. A seguir, serão apresentados os modelos utilizados neste projeto, uma breve explicação de cada modelo e os resultados obtidos. Todos os modelos presentes neste trabalho foram adaptados utilizando a biblioteca de código aberto do Python, a *Scikit-learn*.

4.4.1 LOGISTIC REGRESSION

Segundo a documentação da *Scikit-learn:*

> *"Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function".*

Essa função possui os seguintes parâmetros como padrão:

Figura 20 – Exemplo de parâmetros do modelo LogisticRegression

```
class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True,
intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0,
warm_start=False, n_jobs=None, l1_ratio=None)                                                                    [source]
```

Fonte: Documentação do *LogisticRegression* no site do *Scikit-Learn*.

Figura 21 – Parâmetros iniciais para o LogisticRegression

```
params = {
    "penalty": ["l1", "l2", "elasticnet", "none"],
    "dual": [True, False],
    "C": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],
    "fit_intercept": [True, False],
    "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
    "max_iter": [800, 900, 1000],
    "multi_class": ["auto", "ovr", "multinomial"],
    "verbose": [1, 2, 3],
    "warm_start": [True, False]
}
```

Fonte: VRECH, 2020.

A Figura 21 mostra quais foram os parâmetros iniciais fornecidos ao modelo para que ele possa percorrer e testar diferentes combinações com o objetivo de encontrar o melhor estimador.

Neste trabalho, o modelo apresentado obteve uma acurácia média de 81% no modelo de *Bag-of-Words* e 82% no modelo de *Skip-Gram*. Estes resultados foram obtidos a partir da seguinte arquitetura:

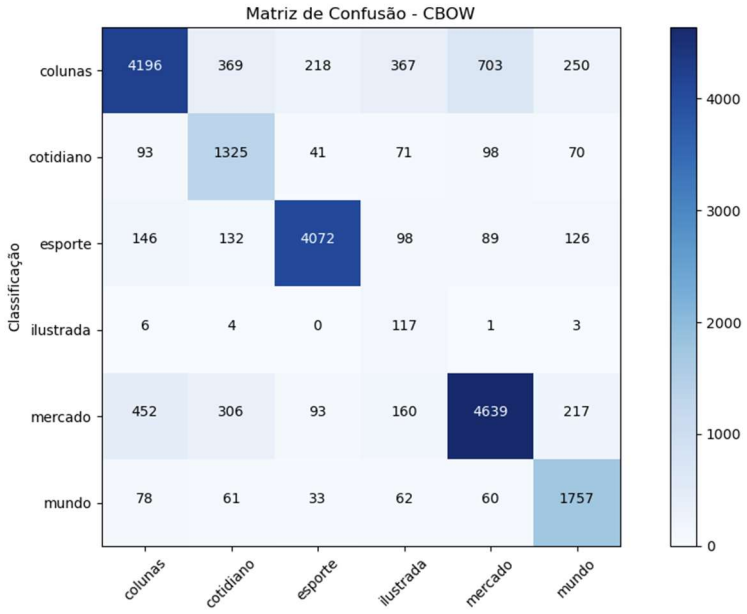
Figura 22 – Arquitetura do modelo LogisticRegression utilizada

```
LogisticRegression(C=0.4, max_iter=1000, multi_class='ovr', solver='saga',
                   verbose=1, warm_start=True)
```

Fonte: VRECH, 2020.

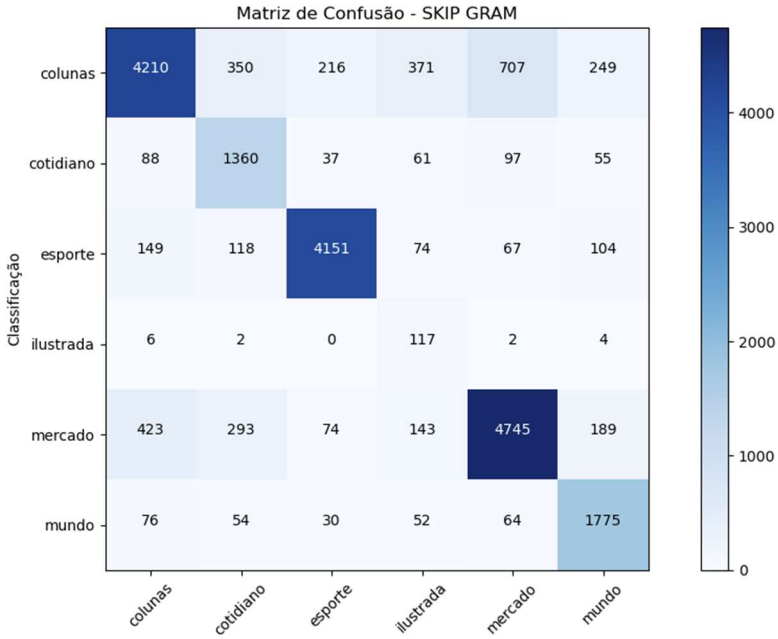
4.4.1.1 RESULTADOS

Figura 23 – Matriz de confusão no *LogisticRegression* utilizando o modelo *Bag-of-Words*



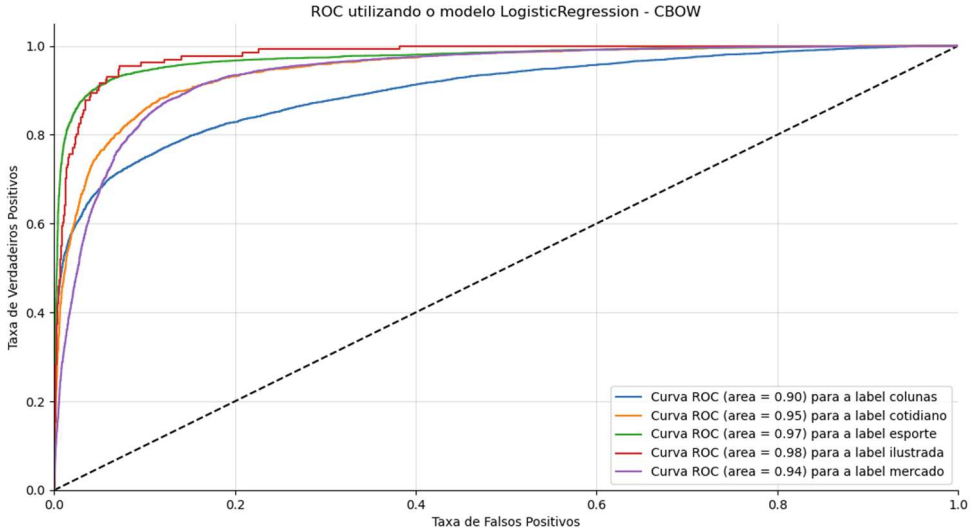
Fonte: VRECH, 2020.

Figura 24 – Matriz de confusão no *LogisticRegression* utilizando o modelo *Skip-Gram*



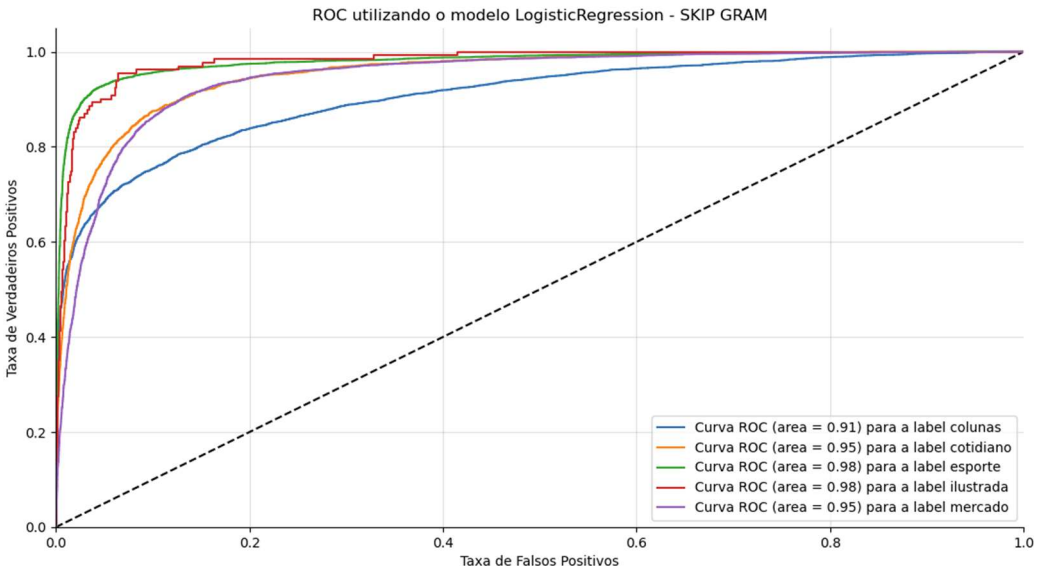
Fonte: VRECH, 2020.

Figura 25 – Curva AOC-ROC no *LogisticRegression* utilizando o modelo *Bag-of-Words*



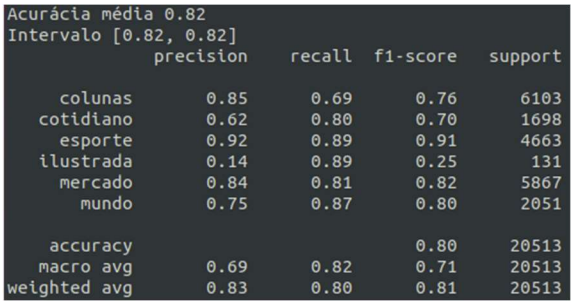
Fonte: VRECH, 2020.

Figura 26 – Curva AOC-ROC no *LogisticRegression* utilizando o modelo *Skip-Gram*



Fonte: VRECH, 2020.

Figura 27 – Métricas gerais do *Bag-of-Words*



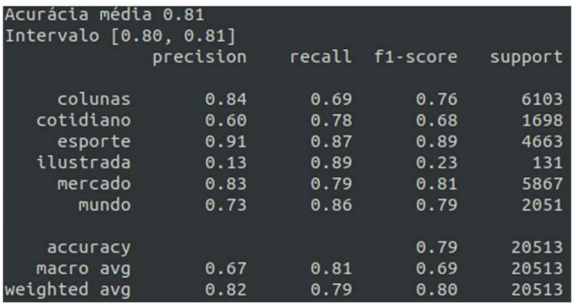
```
Acurácia média 0.82
Intervalo [0.82, 0.82]
              precision    recall  f1-score   support

     colunas       0.85      0.69      0.76      6103
   cotidiano       0.62      0.80      0.70      1698
     esporte       0.92      0.89      0.91      4663
   ilustrada       0.14      0.89      0.25       131
     mercado       0.84      0.81      0.82      5867
       mundo       0.75      0.87      0.80      2051

    accuracy                           0.80     20513
   macro avg       0.69      0.82      0.71     20513
weighted avg       0.83      0.80      0.81     20513
```

Fonte: VRECH, 2020.

Figura 28 – Métricas gerais do *Skip-Gram*



```
Acurácia média 0.81
Intervalo [0.80, 0.81]
              precision    recall  f1-score   support

     colunas       0.84      0.69      0.76      6103
   cotidiano       0.60      0.78      0.68      1698
     esporte       0.91      0.87      0.89      4663
   ilustrada       0.13      0.89      0.23       131
     mercado       0.83      0.79      0.81      5867
       mundo       0.73      0.86      0.79      2051

    accuracy                           0.79     20513
   macro avg       0.67      0.81      0.69     20513
weighted avg       0.82      0.79      0.80     20513
```

Fonte: VRECH, 2020.

Pode-se concluir que esses resultados foram interessantes, pois foram obtidas uma acurácia relativamente alta e uma boa matriz de confusão, juntamente com uma boa curva ROC.

4.4.2 LINEAR DISCRIMINANT ANALYSIS

Segundo a documentação da *Scikit-learn:*

> *"A classifier with a linear decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule. The model fits a Gaussian density to each class, assuming that all classes share the same covariance matrix".*

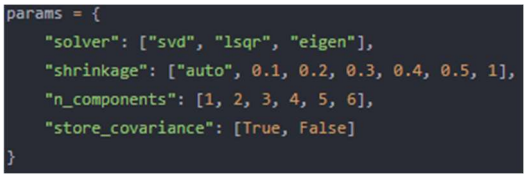
Essa função possui os seguintes parâmetros como padrão:

Figura 29 – Exemplo de parâmetros do modelo *LinearDiscriminantAnalysis*

class sklearn.discriminant_analysis.**LinearDiscriminantAnalysis**(**, solver='svd', shrinkage=None, priors=None, n_components=None, store_covariance=False, tol=0.0001*) [source]

Fonte: Documentação do *LinearDiscriminantAnalysis* do site *Scikit-Learn.*

Figura 30 – Parâmetros iniciais para o *LinearDiscriminantAnalysis*



```
params = {
    "solver": ["svd", "lsqr", "eigen"],
    "shrinkage": ["auto", 0.1, 0.2, 0.3, 0.4, 0.5, 1],
    "n_components": [1, 2, 3, 4, 5, 6],
    "store_covariance": [True, False]
}
```

Fonte: VRECH, 2020.

Neste trabalho, o modelo apresentado obteve uma acurácia média de 78% no modelo de *Bag-of-Words* e 80% no modelo de *Skip-Gram*; esses resultados foram obtidos a partir da seguinte arquitetura:

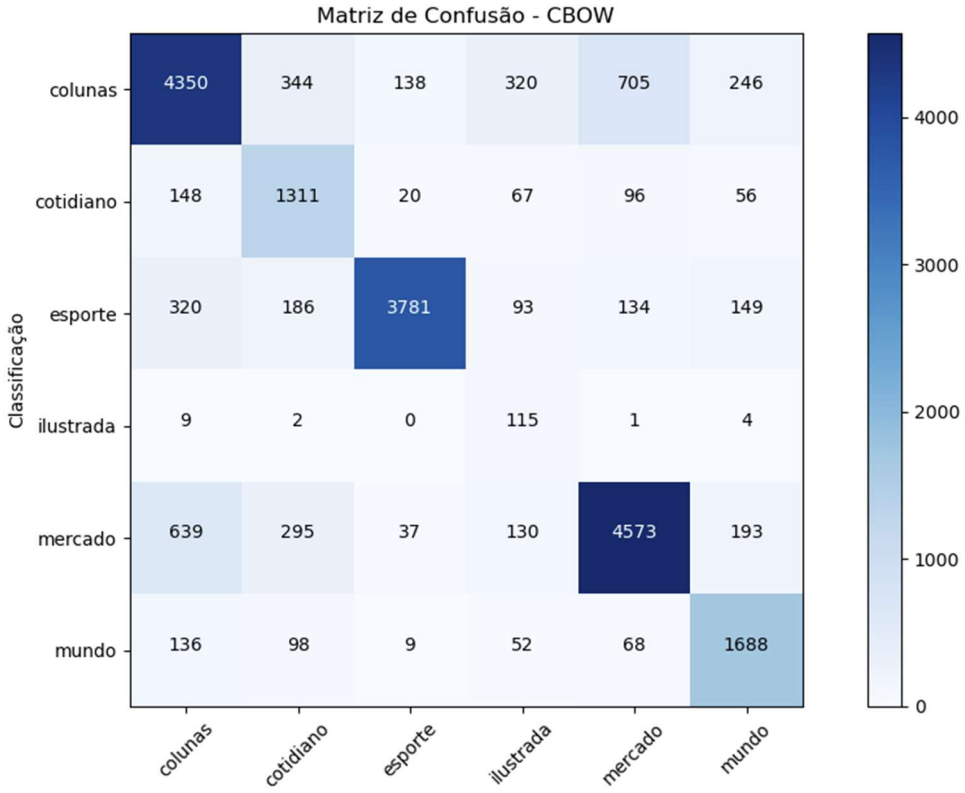
Figura 31 – Arquitetura do modelo *LinearDiscriminantAnalysis* utilizada

```
LinearDiscriminantAnalysis(n_components=3, shrinkage='auto', solver='lsqr')
```

Fonte: VRECH, 2020.

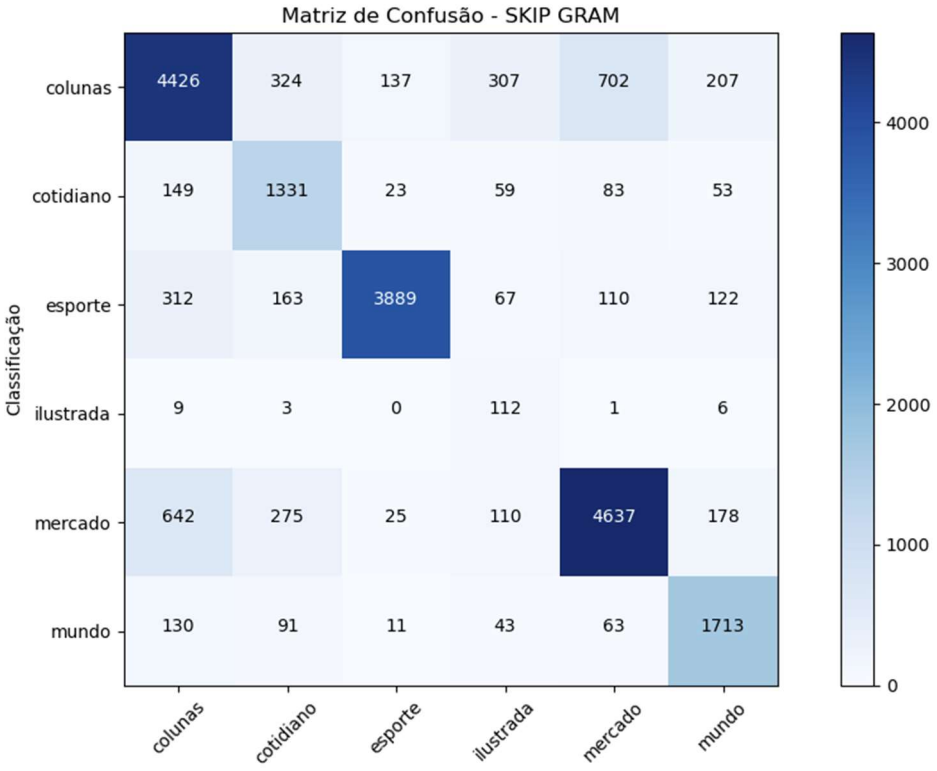
4.4.2.1 RESULTADOS

Figura 32 – Matriz de confusão no *LinearDiscriminantAnalysis* utilizando o modelo *Bag-of-Words*



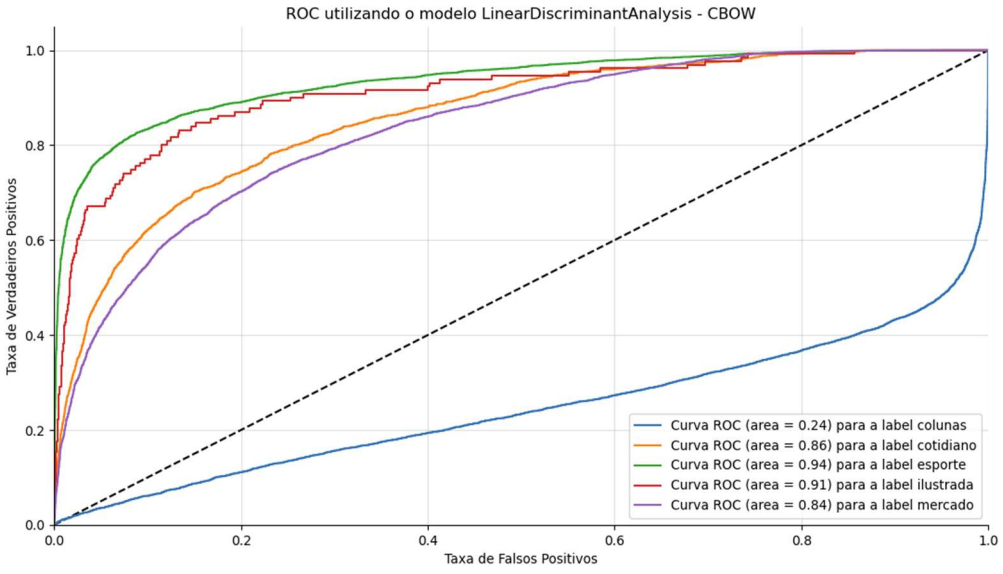
Fonte: VRECH, 2020.

Figura 33 – Matriz de confusão no *LinearDiscriminantAnalysis* utilizando o modelo *Skip-Gram*



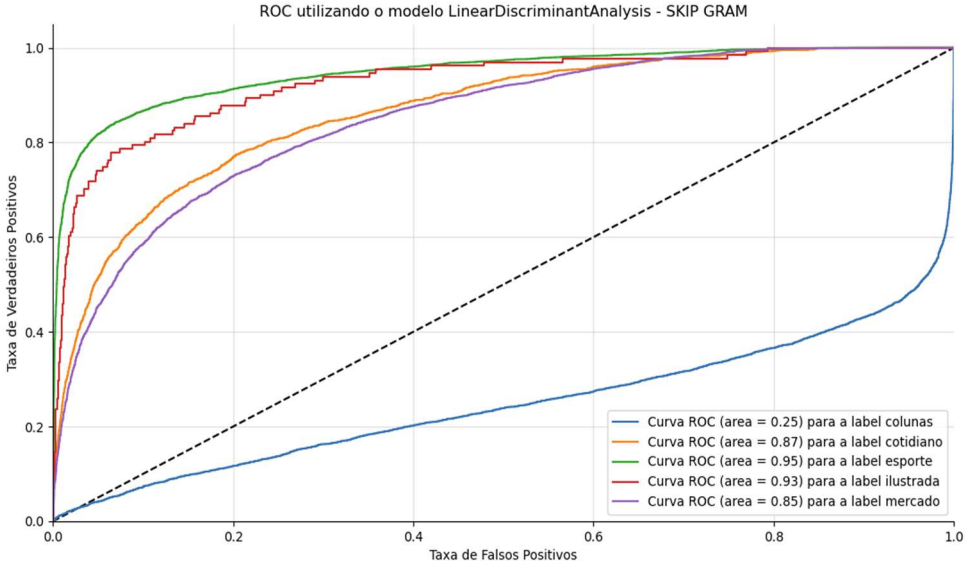
Fonte: VRECH, 2020.

Figura 34 – Curva AOC-ROC no *LinearDiscriminantAnalysis* utilizando o modelo *Bag-of-Words*



Fonte: VRECH, 2020.

Figura 35 – Curva AOC-ROC no *LinearDiscriminantAnalysis* utilizando o modelo *Skip-Gram*



Fonte: VRECH, 2020.

Figura 36 – Métricas gerais do *Skip-Gram*

Acurácia média 0.78
Intervalo [0.78, 0.79]

	precision	recall	f1-score	support
colunas	0.78	0.71	0.74	6103
cotidiano	0.59	0.77	0.67	1698
esporte	0.95	0.81	0.87	4663
ilustrada	0.15	0.88	0.25	131
mercado	0.82	0.78	0.80	5867
mundo	0.72	0.82	0.77	2051
accuracy			0.77	20513
macro avg	0.67	0.80	0.68	20513
weighted avg	0.80	0.77	0.78	20513

Fonte: VRECH, 2020.

Figura 37 – Métricas gerais do *Bag-of-Words*

Acurácia média 0.80
Intervalo [0.79, 0.80]

	precision	recall	f1-score	support
colunas	0.78	0.73	0.75	6103
cotidiano	0.61	0.78	0.69	1698
esporte	0.95	0.83	0.89	4663
ilustrada	0.16	0.85	0.27	131
mercado	0.83	0.79	0.81	5867
mundo	0.75	0.84	0.79	2051
accuracy			0.79	20513
macro avg	0.68	0.80	0.70	20513
weighted avg	0.81	0.79	0.79	20513

Fonte: VRECH, 2020.

Apesar de esse modelo ter obtido uma acurácia interessante, a curva ROC não foi tão boa assim, ou seja, o modelo tem dificuldades em classificar algumas classes do problema, que neste caso seria a classe de "colunas".

4.4.3 NEAREST CENTROID

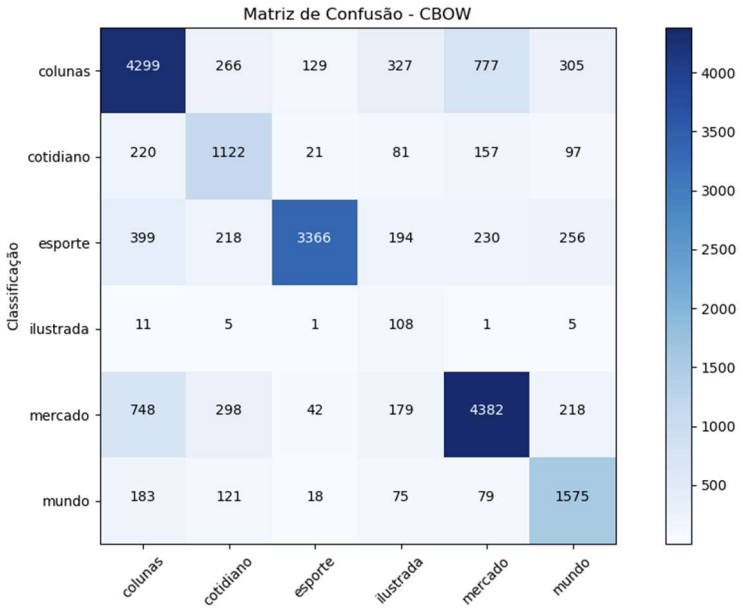
Segundo a documentação da *Scikit-learn:*

> *"The NearestCentroid classifier is a simple algorithm that represents each class by the centroid of its members. In effect, this makes it similar to the label updating phase of the sklearn.cluster.KMeans algorithm. It also has no parameters to choose, making it a good baseline classifier. It does, however, suffer on non-convex classes, as well as when classes have drastically different variances, as equal variance in all dimensions is assumed".*

Como especificado na documentação, esse modelo não possui parâmetros. Neste trabalho, o modelo apresentado obteve uma acurácia média de 72% no modelo de *Bag-of-Words* e 73% no modelo de *Skip-Gram*; esses resultados foram obtidos apenas utilizando o método, já que não existem parâmetros aqui.

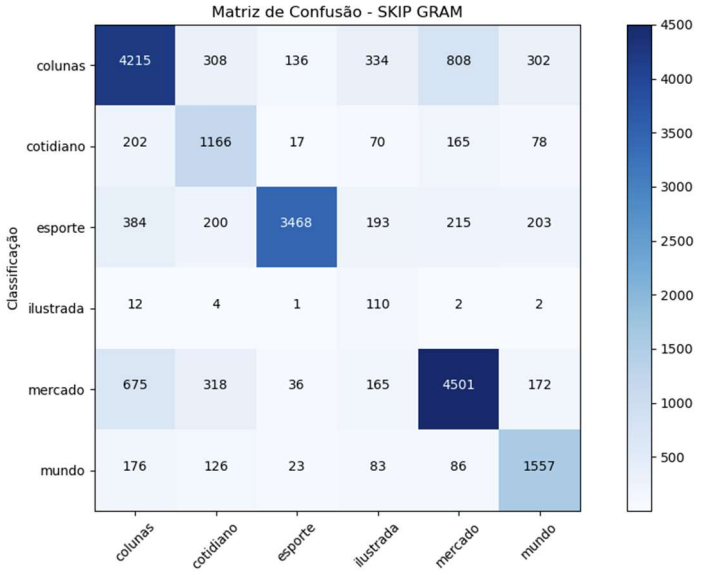
4.4.3.1 RESULTADOS

Figura 38 – Matriz de confusão no *NearestCentroid* utilizando o modelo *Bag-of-Words*



Fonte: VRECH, 2020.

Figura 39 – Matriz de confusão no *NearestCentroid* utilizando o modelo *Skip-Gram*



Fonte: VRECH, 2020.

Figura 40 – Métricas gerais do *Bag-of-Words*

	precision	recall	f1-score	support
colunas	0.73	0.70	0.72	6103
cotidiano	0.55	0.66	0.60	1698
esporte	0.94	0.72	0.82	4663
ilustrada	0.11	0.82	0.20	131
mercado	0.78	0.75	0.76	5867
mundo	0.64	0.77	0.70	2051
accuracy			0.72	20513
macro avg	0.63	0.74	0.63	20513
weighted avg	0.77	0.72	0.74	20513

Fonte: VRECH, 2020.

Figura 41 – Métricas gerais do *Skip-Gram*

	precision	recall	f1-score	support
colunas	0.74	0.69	0.72	6103
cotidiano	0.55	0.69	0.61	1698
esporte	0.94	0.74	0.83	4663
ilustrada	0.12	0.84	0.20	131
mercado	0.78	0.77	0.77	5867
mundo	0.67	0.76	0.71	2051
accuracy			0.73	20513
macro avg	0.63	0.75	0.64	20513
weighted avg	0.77	0.73	0.75	20513

Fonte: VRECH, 2020.

Neste modelo a acurácia já se mostra inferior em comparação com os resultados anteriores.

4.4.4 DECISION TREE CLASSIFIER

Segundo a documentação da *Scikit-learn:* "*Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.*"

Essa função possui os seguintes parâmetros como padrão:

Figura 42 – Exemplo de parâmetros do modelo *DecisionTreeClassifier*

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0)   [source]
```

Fonte: Documentação do *DecisionTreeClassifier* no site do *Scikit-Learn.*

Figura 43 – Parâmetros iniciais para o *DecisionTreeClassifier*

```
params = {
    "max_depth": [3, 5],
    "min_samples_split": [32, 64, 128],
    "min_samples_leaf": [32, 64, 128],
    "criterion": ["gini", "entropy"]
}
```

Fonte: VRECH, 2020.

A Figura 43 mostra quais foram os parâmetros iniciais fornecidos ao modelo para que ele possa percorrer e testar diferentes combinações com o objetivo de achar o melhor estimador; entretanto, neste modelo específico, após alguns testes foi constatado que se não fosse fornecido nenhum parâmetro, o desempenho era nitidamente melhor.

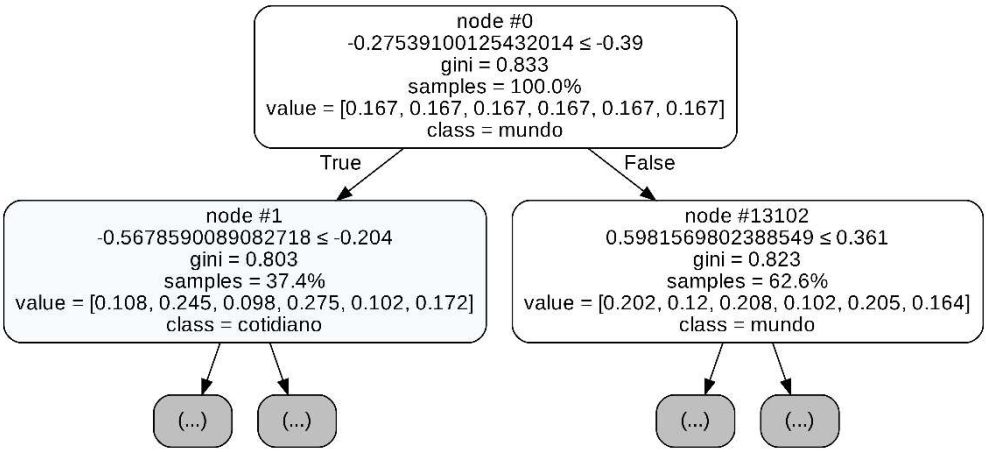
Neste trabalho, o modelo apresentado obteve uma acurácia média de 48% no modelo de *Bag-of-Words* e 57% no modelo de *Skip-Gram*; esses resultados foram obtidos a partir da seguinte arquitetura:

Figura 44 – Arquitetura do modelo *DecisionTreeClassifier*

```
model = DecisionTreeClassifier()
```

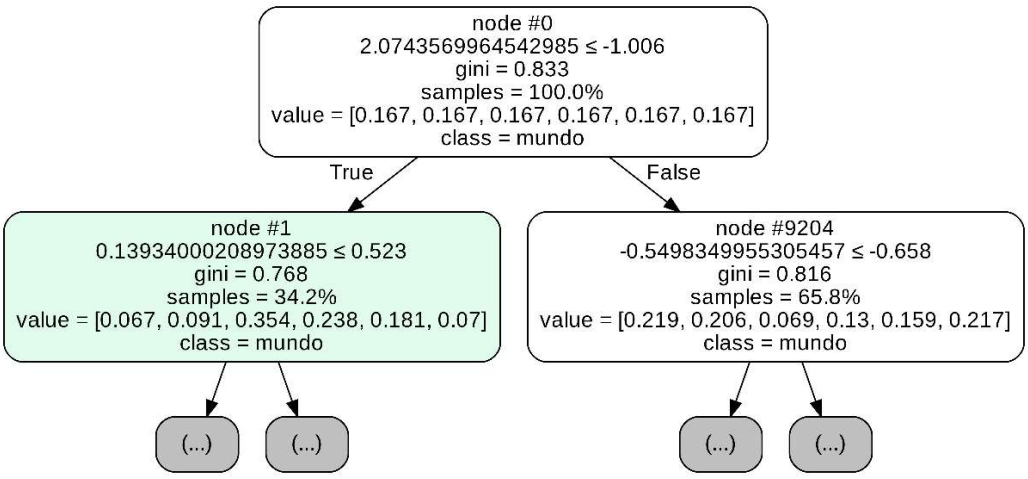
Fonte: VRECH, 2020.

Figura 45 – Árvore de decisão tomada pelo modelo *Bag-of-Words* no *DecisionTreeClassifier*



Fonte: VRECH, 2020.

Figura 46 – Árvore de decisão tomada pelo modelo *Skip-Gram* no *DecisionTreeClassifier*

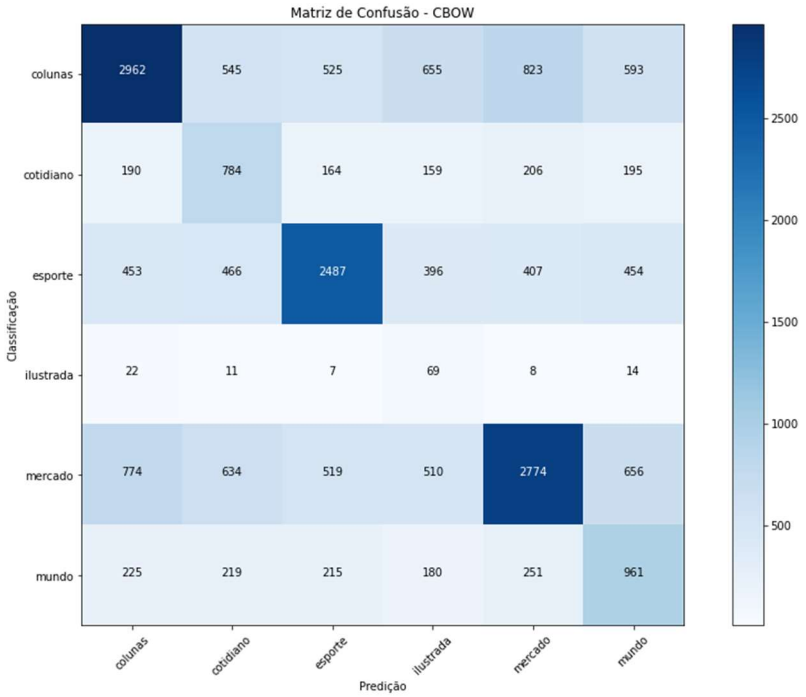


Fonte: VRECH, 2020.

Nas Figuras 45 e 46, é possível visualizar (com uma profundidade de nós até 1) quais foram as estruturas de decisões tomadas pelo algoritmo. Na Figura 45 é mostrada a decisão de escolhas tomadas pelo modelo Bag-of-Words, enquanto na Figura 46 utiliza-se o modelo Skip-Gram.

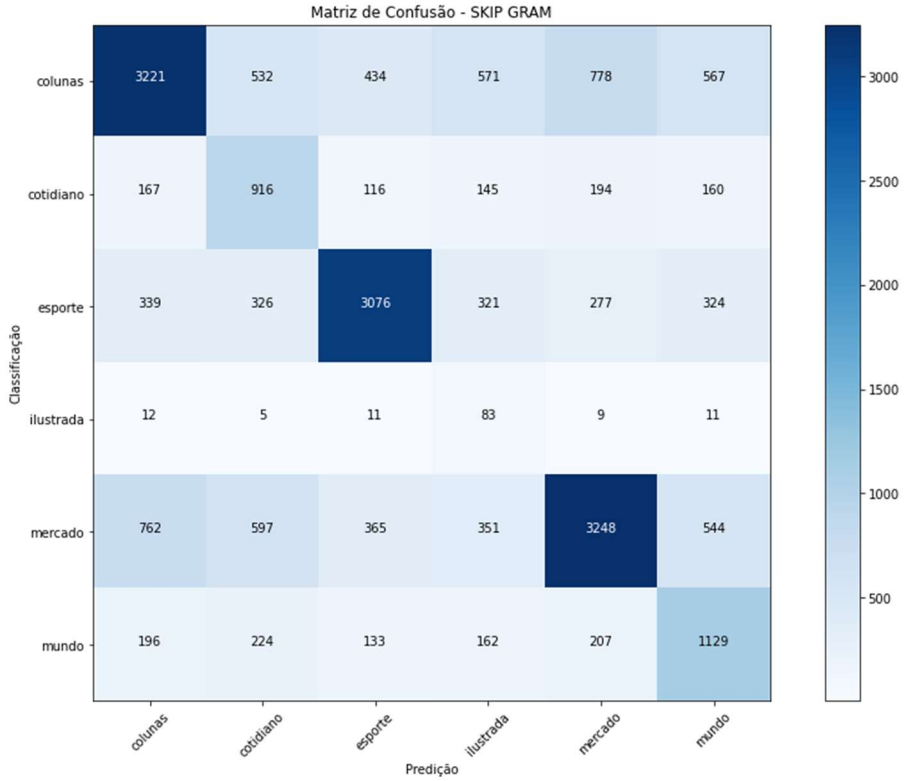
4.4.4.1 RESULTADOS

Figura 47 – Matriz de confusão no *DecisionTreeClassifier* utilizando o modelo *Bag-of-Words*



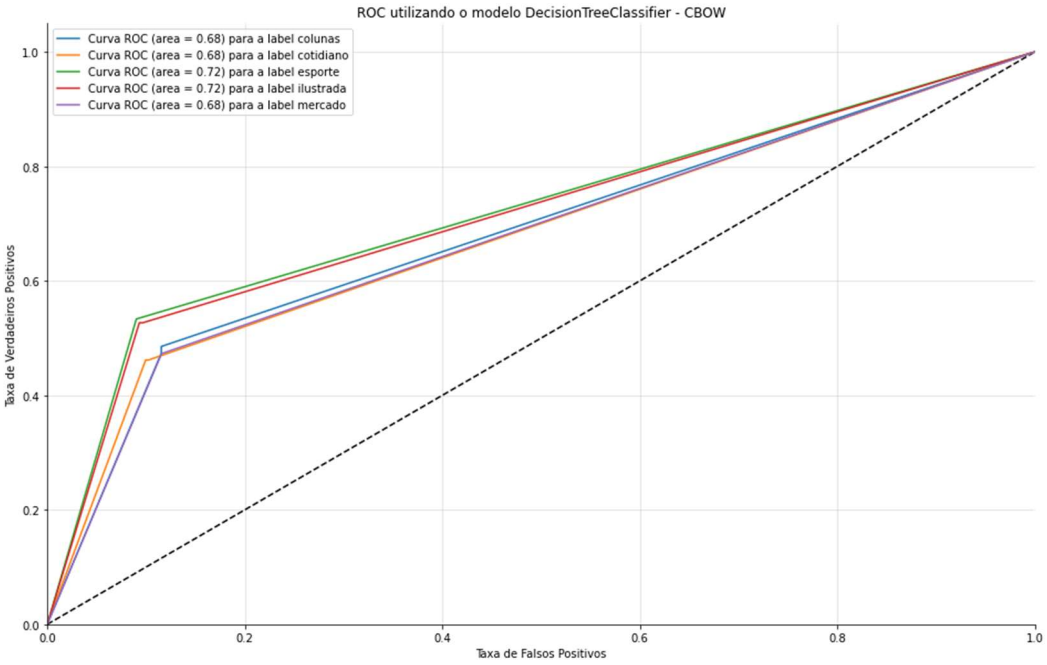
Fonte: VRECH, 2020.

Figura 48 – Matriz de confusão no *DecisionTreeClassifier* utilizando o modelo *Skip-Gram*



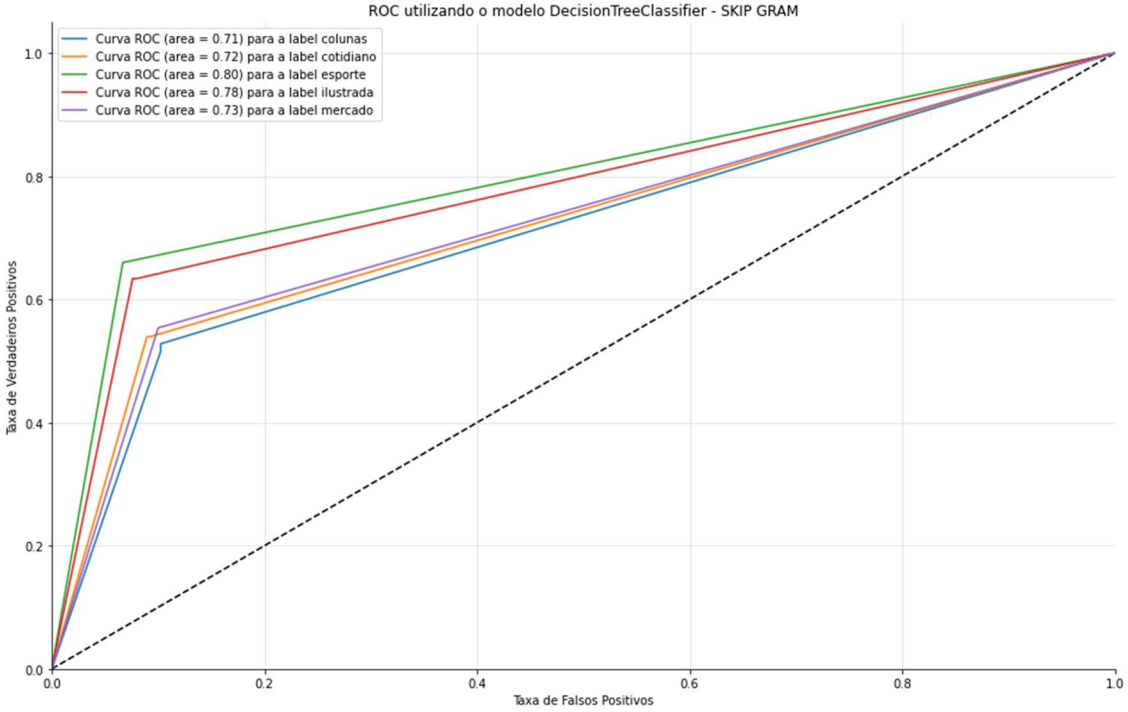
Fonte: VRECH, 2020.

Figura 49 – Curva AOC-ROC no *DecisionTreeClassifier* utilizando o modelo *Bag-of-Words*



Fonte: VRECH, 2020.

Figura 50 – Curva AOC-ROC no *DecisionTreeClassifier* utilizando o modelo *Skip-Gram*



Fonte: VRECH, 2020.

Figura 51 – Métricas gerais do *Bag-of-Words*

```
CBOW
Acurácia média 0.48
Intervalo [0.48, 0.48]
              precision    recall  f1-score   support

     colunas       0.64      0.49      0.55      6103
   cotidiano       0.29      0.46      0.36      1698
     esporte       0.63      0.53      0.58      4663
   ilustrada       0.04      0.53      0.07       131
     mercado       0.62      0.47      0.54      5867
       mundo       0.33      0.47      0.39      2051

    accuracy                           0.49     20513
   macro avg       0.43      0.49      0.41     20513
weighted avg       0.57      0.49      0.52     20513
```

Fonte: VRECH, 2020.

Figura 52 – Métricas gerais do *Skip-Gram*

```
SKIP GRAM
Acurácia média 0.57
Intervalo [0.56, 0.58]
              precision    recall  f1-score   support

     colunas       0.69      0.53      0.60      6103
   cotidiano       0.35      0.54      0.43      1698
     esporte       0.74      0.66      0.70      4663
   ilustrada       0.05      0.63      0.09       131
     mercado       0.69      0.55      0.61      5867
       mundo       0.41      0.55      0.47      2051

    accuracy                           0.57     20513
   macro avg       0.49      0.58      0.48     20513
weighted avg       0.64      0.57      0.60     20513
```

Fonte: VRECH, 2020.

Os resultados obtidos nesse modelo de longe foram os piores até o momento. Acurácia baixa, apesar da curva ROC não estar abaixo da linha de base, ela se encontra um pouco distante do ponto ideal (1.0), a matriz de confusão tem diversos erros, ou seja, o modelo tem dificuldades em classificar uma classe correta como realmente correta e uma classe incorreta como realmente incorreta.

4.4.5 DUMMY CLASSIFIER

Como explicado anteriormente neste trabalho, o modelo *Dummy* serve apenas como base para outros modelos, ou seja, para verificar se os outros modelos estão obtendo um bom desempenho. Este modelo não será utilizado como arquitetura final.

Essa função possui os seguintes parâmetros como padrão:

Figura 53 – Exemplo de parâmetros do modelo *DummyClassifier*

```
class sklearn.dummy.DummyClassifier(*, strategy='warn', random_state=None, constant=None)   [source]
```

Fonte: Documentação do *DummyClassifier* do site *Scikit-Learn.*

Neste trabalho, o modelo apresentado obteve uma acurácia média de 30% em ambos os modelos de *Bag-of-Words*; no modelo de *Skip-Gram*, esses resultados foram obtidos a partir da seguinte arquitetura:

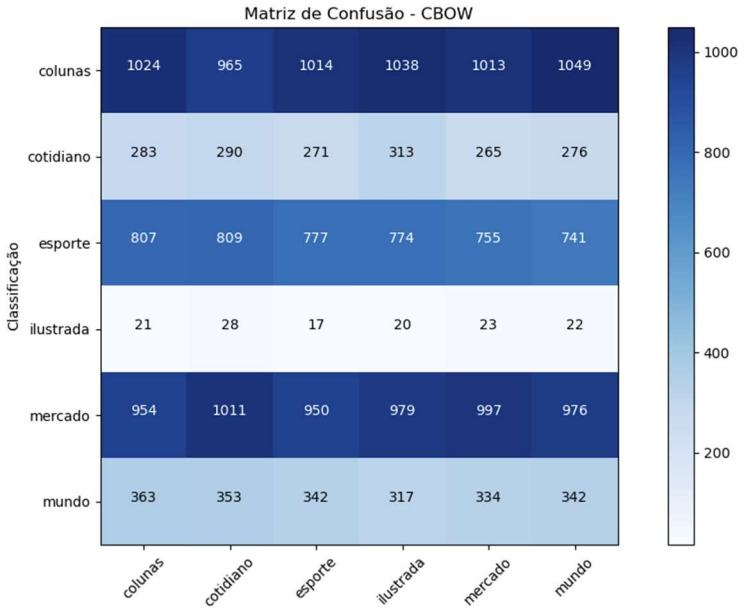
Figura 54 – *Arquitetura do modelo DummyClassifier*

```
DummyClassifier(strategy="prior")
```

Fonte: VRECH, 2020.

4.4.5.1 RESULTADOS

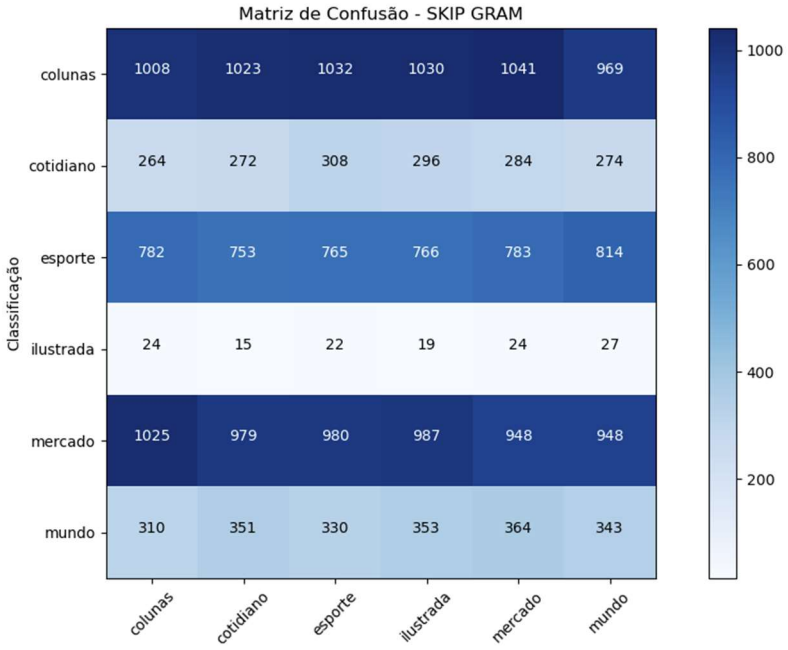
Figura 55 – Matriz de confusão no *DummyClassifier* utilizando o modelo *Bag-of-Words*



Fonte: VRECH, 2020.

Percebe-se que na Figura 55 a matriz de confusão obtida não é boa, já que o gráfico de calor está mais escuro em quase todas as regiões, ou seja, o modelo classificou errado a maioria das classes. Como se trata de um modelo *Dummy*, isso não tem problema. Na verdade, é interessante que essa matriz esteja pior que as matrizes apresentadas anteriormente, pois isso significa que os modelos propostos são melhores que um modelo *Dummy*.

Figura 56 – Matriz de confusão no *DummyClassifier* utilizando o modelo *Skip-Gram*



Fonte: VRECH, 2020.

Figura 57 – Métricas gerais do *Bag-of-Words*

CBOW
prior

	precision	recall	f1-score	support
colunas	0.30	1.00	0.46	6103
cotidiano	0.00	0.00	0.00	1698
esporte	0.00	0.00	0.00	4663
ilustrada	0.00	0.00	0.00	131
mercado	0.00	0.00	0.00	5867
mundo	0.00	0.00	0.00	2051
accuracy			0.30	20513
macro avg	0.05	0.17	0.08	20513
weighted avg	0.09	0.30	0.14	20513

Fonte: VRECH, 2020.

Figura 58 – Métricas gerais do *Skip-Gram*

SKIP GRAM
prior

	precision	recall	f1-score	support
colunas	0.30	1.00	0.46	6103
cotidiano	0.00	0.00	0.00	1698
esporte	0.00	0.00	0.00	4663
ilustrada	0.00	0.00	0.00	131
mercado	0.00	0.00	0.00	5867
mundo	0.00	0.00	0.00	2051
accuracy			0.30	20513
macro avg	0.05	0.17	0.08	20513
weighted avg	0.09	0.30	0.14	20513

Fonte: VRECH, 2020.

Como explicado anteriormente, o modelo *Dummy Classifier* é bastante utilizado como métrica de validação, o que se mostra bastante interessante para esse projeto, já que nenhum modelo ficou abaixo deste.

4.4.6 RANDOM FOREST CLASSIFIER

Segundo a documentação do *Scikit-learn* – "*A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting*".

Essa função possui os seguintes parâmetros como padrão:

Figura 59 – Exemplo de parâmetros do modelo *RandomForestClassifier*

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) [source]

Fonte: Documentação do *RandomForestClassifier* do site *Scikit-Learn.*

Figura 60 – Parâmetros iniciais para o *RandomForestClassifier*

```
params = {
    "n_estimators": [100, 120, 140, 160, 180, 200, 210, 220],
    "criterion": ["gini", "entropy"],
    "max_depth": [3, 5],
    "min_samples_split": [32, 64, 128],
    "min_samples_leaf": [32, 64, 128],
    "min_weight_fraction_leaf": [0.0, 0.5],
    "max_features": ["auto", "sqrt", "log2"],
    "max_leaf_nodes": [1, 2, 3, 4, 5, 6],
    "min_impurity_decrease": [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0],
    "bootstrap": [True],
    "oob_score": [True, False],
    "verbose": [1, 2, 3, 4, 5, 6],
    "warm_start": [True, False],
    "max_samples": [3, 5],
}
```

Fonte: VRECH, 2020.

A Figura 60 mostra quais foram os parâmetros iniciais fornecidos ao modelo para que ele possa percorrer e testar diferentes combinações, com o objetivo de encontrar o melhor estimador. Entretanto, neste modelo específico, após fazer alguns testes foi constatado que se fosse fornecido apenas o parâmetro *n_estimators* (parâmetro que define o número de árvores utilizada no *RandomForestClassifier*), o desempenho era nitidamente melhor.

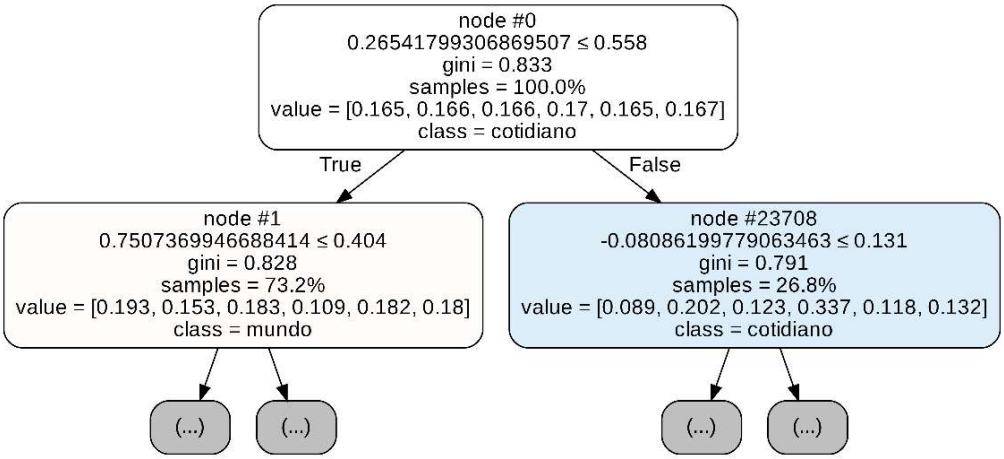
Neste trabalho, o modelo apresentado obteve uma acurácia média de 63% no modelo de *Bag-of-Words* e de 70% no modelo de *Skip-Gram*; esses resultados foram obtidos a partir da seguinte arquitetura:

Figura 61 – Arquitetura utilizada no modelo *RandomForestClassifier*

```
RandomForestClassifier(n_estimators=10)
```

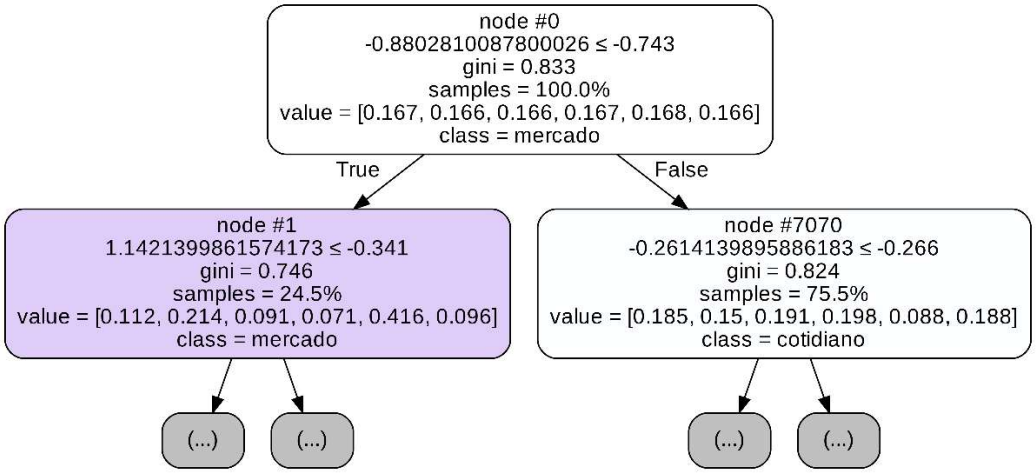
Fonte: VRECH, 2020.

Figura 62 – Árvore de decisão tomada pelo modelo *Bag-of-Words* no *RandomForestClassifier*



Fonte: VRECH, 2020.

Figura 63 – Árvore de decisão tomada pelo modelo *Skip-Gram* no *RandomForestClassifier*

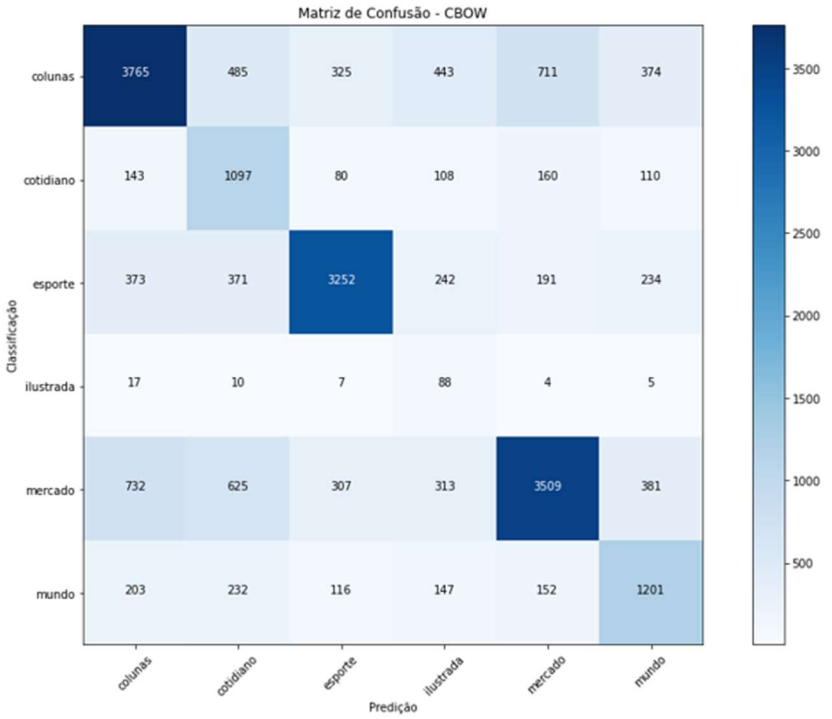


Fonte: VRECH, 2020.

Essas figuras mostram como a árvore de decisão tomou cada escolha, ou seja, o que ela levou em consideração ao classificar uma classe no problema.

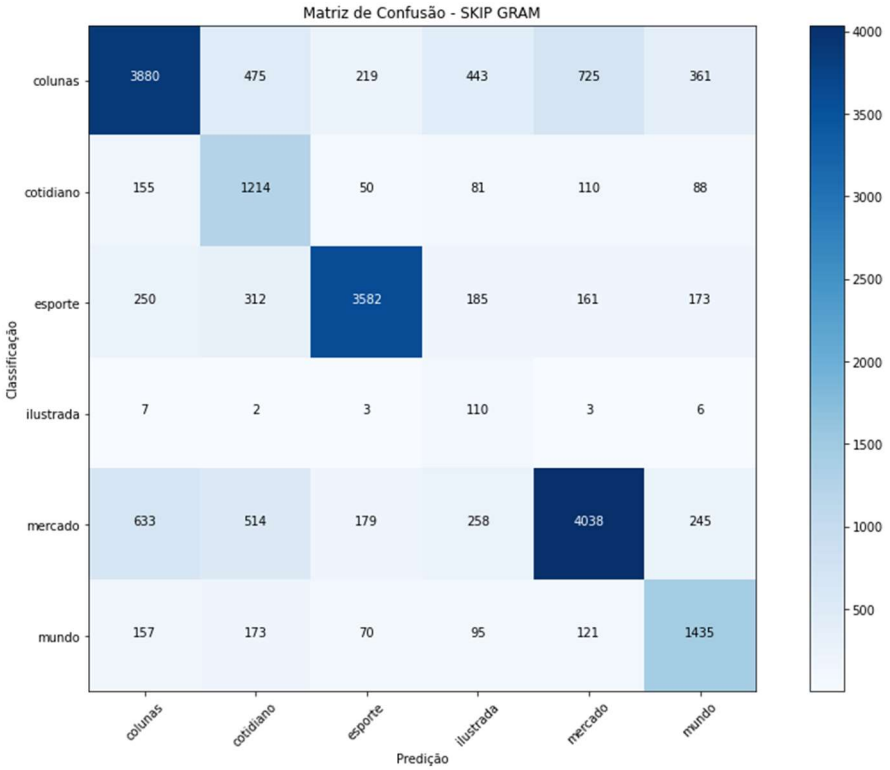
4.4.6.1 RESULTADOS

Figura 64 – Matriz de confusão no *RandomForestClassifier* utilizando o modelo *Bag-of-Words*



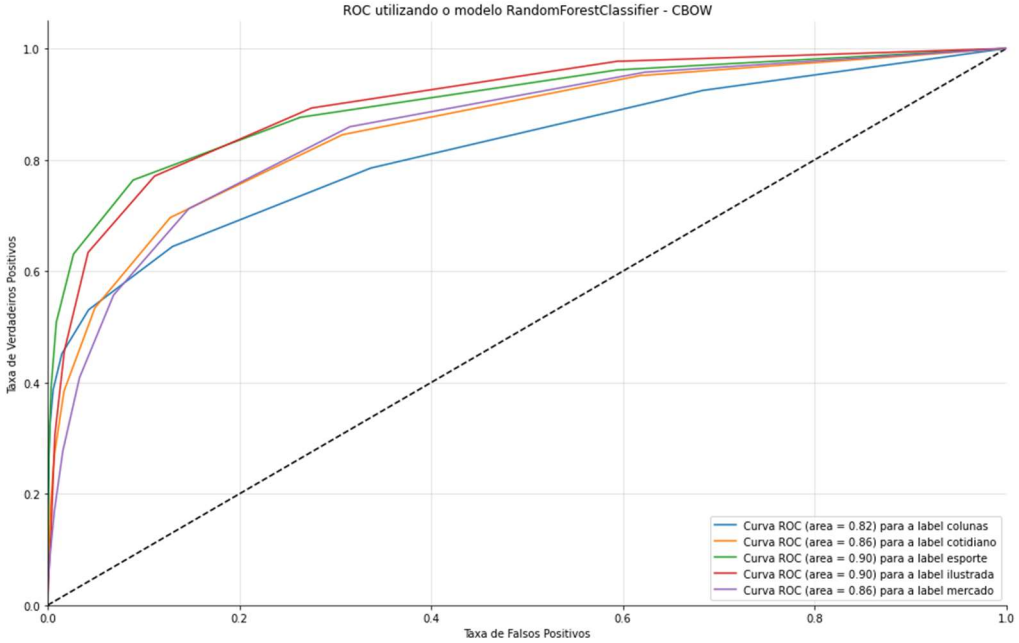
Fonte: VRECH, 2020.

Figura 65 – Matriz de confusão no *RandomForestClassifier* utilizando o modelo *Skip-Gram*



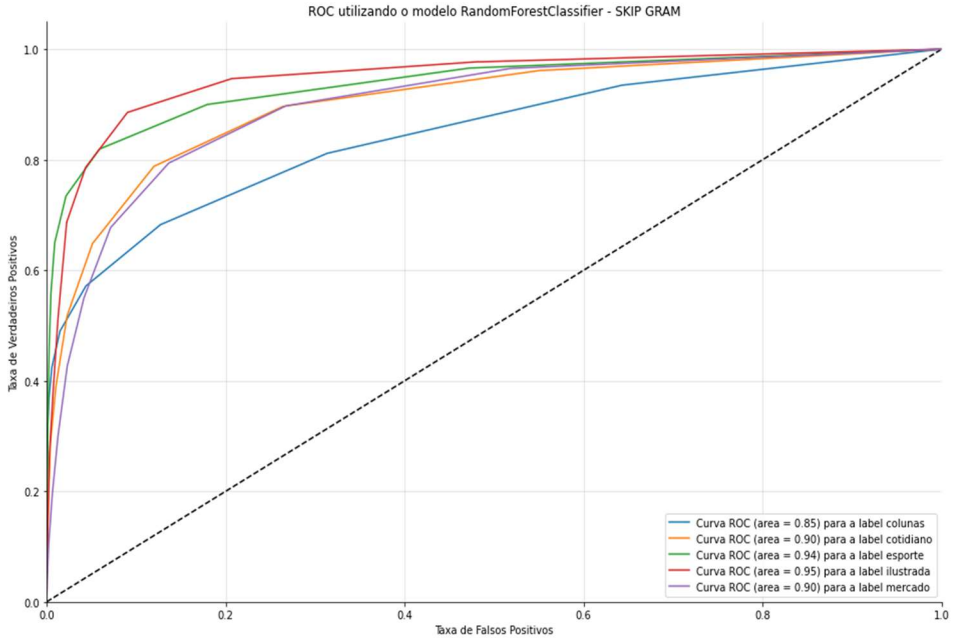
Fonte: VRECH, 2020.

Figura 66 – Curva AOC-ROC no *RandomForestClassifier* utilizando o modelo *Bag-of-Words*



Fonte: VRECH, 2020.

Figura 67 – Curva AOC-ROC no *RandomForestClassifier* utilizando o modelo *Skip-Gram*



Fonte: VRECH, 2020.

Figura 68 – Métricas gerais do *Bag-of-Words*

```
CBOW
Acurácia média 0.63
Intervalo [0.63, 0.64]
              precision    recall  f1-score   support

     colunas       0.72      0.61      0.66      6103
   cotidiano       0.38      0.63      0.47      1698
     esporte       0.78      0.68      0.73      4663
   ilustrada       0.07      0.76      0.13       131
     mercado       0.73      0.59      0.65      5867
       mundo       0.54      0.59      0.56      2051

    accuracy                           0.62     20513
   macro avg       0.54      0.64      0.53     20513
weighted avg       0.69      0.62      0.64     20513
```

Fonte: VRECH, 2020.

Figura 69 – Métricas gerais do *Skip-Gram*

```
SKIP GRAM
Acurácia média 0.71
Intervalo [0.70, 0.71]
              precision    recall  f1-score   support

     colunas       0.76      0.64      0.69      6103
   cotidiano       0.45      0.71      0.55      1698
     esporte       0.87      0.77      0.82      4663
   ilustrada       0.09      0.84      0.17       131
     mercado       0.78      0.69      0.73      5867
       mundo       0.62      0.70      0.66      2051

    accuracy                           0.70     20513
   macro avg       0.60      0.72      0.60     20513
weighted avg       0.75      0.70      0.71     20513
```

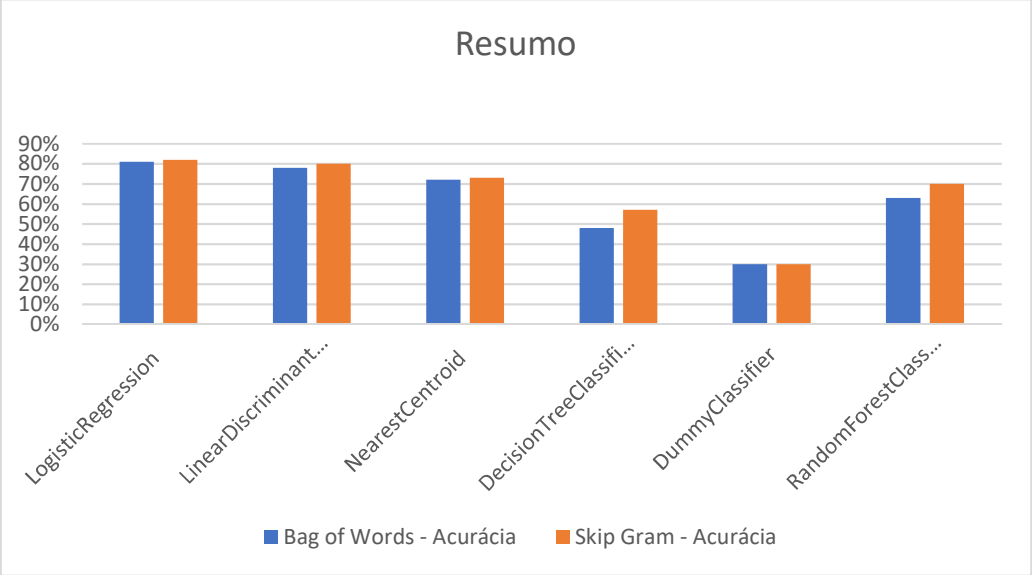
Fonte: VRECH, 2020.

Este modelo também não obteve resultados interessantes quando comparados com alguns modelos anteriores, como, por exemplo, o modelo de *LogisticRegression*. Isso pode ter ocorrido por diversos fatores, tais como: não ser o melhor modelo para esse tipo de problema; não ter encontrado os melhores parâmetros para o modelo; pré-processamento de dados não foi o suficiente etc.

4.4.7 RESUMO DOS MODELOS APRESENTADOS

A seguir, um gráfico mostra a acurácia de todos os modelos apresentados anteriormente, permitindo verificar qual modelo obteve o melhor desempenho em relação a esse critério.

Figura 70 – Resumo das acurácias dos modelos apresentados neste trabalho



Fonte: VRECH, 2020.

Na Figura 70 são mostradas as acurácias obtidas neste trabalho; pode-se perceber que o modelo utilizando o *LogisticRegression* teve a melhor acurácia, e isso deve-se a diversos fatores, tais como: modelo se encaixa bem para resolução desse tipo de problema; combinação de parâmetros bem utilizados; boa adaptação na utilização do *Word Embeddings* etc.

5. CONCLUSÕES

Este trabalho teve como início o estudo na área de *Natural Language Processing* (NLP). Com este estudo, pôde-se entender como funciona a comunicação homem-máquina, como é feito o *Words Embeedings*, como é composto um conjunto de dados e quais tratamentos são possíveis na parte de pré-processamento dos dados. Foi apresentado como é realizado o treinamento de um modelo de *Machine Learning* na área de NLP.

Após esse estudo, foi encontrado um problema para cuja resolução o *Machine Learning* seria útil. O problema consiste em classificar as várias classes disponíveis; sendo fornecido o título de uma notícia, o modelo estimaria em qual categoria este título se encaixaria (Mundo, Cotidiano, Mercado, Ilustrada, Esporte ou Colunas). A partir deste problema, foi possível realizar a implementação de seis modelos distintos de *Machine Learning* e avaliar cada um deles individualmente, com o objetivo de mostrar o desempenho dos mesmos, utilizando técnicas de validação cruzada, na tentativa de encontrar os melhores parâmetros para cada modelo. Em cada modelo foram utilizados os dois tipos de abordagens do *Word2Vec*, o *Bag-of-Words* e o *Skip-Gram*. O modelo LogisticRegression, utilizando a técnica *Skip-Gram*, obteve a maior acurácia, de 82%, dentre todos os modelos.

Para trabalhos futuros, com o foco maior no pré-processamento dos dados, aliado a uma exploração de parâmetros mais variados, certamente é possível conseguir desempenhos ainda melhores para os modelos desenvolvidos nos dois problemas. A utilização de *GPU (Graphics Processing Unit)* irá ajudar imensamente no processamento de treinamento dos modelos, diminuindo o tempo de processamento dos mesmos.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AQUARELA. **Datasets, o que são e como utilizá-los**, 2018. Disponível em: <https://www.aquare.la/datasets-o-que-sao-e-como-utiliza-los/>. Acesso em: 2 de nov. de 2020.

GENSIM. **Gensim,** 2009. Disponível em: <https://radimrehurek.com/gensim/>. Acesso em: 18 de out. de 2020.

GITHUB. **GitHub**, 2008. Disponível em: <https://github.com/>. Acesso em: 18 de out. de 2020.

GONÇALVES, Luis. Machine Reading Comprehension – inteligência artificial que consegue ler e interpretar textos. **Medium,** 2018. Disponível em: <https://medium.com/luisfredgs/uma-inteligência-artificial-que-consegue-ler-e-interpretar-textos-da108a2f1041>. Acesso em: 13 de dez. de 2020.

GONZALES e VERA. Recuperação de Informação e Processamento da Linguagem Natural. **PUCRS**, 2003. Disponível em: <https://www.marilia.unesp.br/Home/Instituicao/Docentes/EdbertoFerneda/mri-06---gonzales-e-lima-2003.pdf >. Acesso em: 17 de out. de 2020.

GOOGLE. **Machine Learning Crash Course**, 2018. Disponível em: <https://developers.google.com/machine-learning/crash-course>. Acesso em: 2 de nov. de 2020.

KLEINA, Nilton. A história da inteligência artificial. **Tecmundo,** 2018. Disponível em: <https://www.tecmundo.com.br/mercado/135413-historia-inteligencia-artificial-video.htm>. Acesso em: 22 de ago. de 2020.

LIU, Yang; LIU, Zhiyuan; CHUA, Tat-Seng; SUN, Maosong. Topical Word Embeddings. **Tsinghua University**, 2015. Disponível em: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.699.6286&rep=rep1&type=pdf>. Acesso em: 18 de out. de 2020.

MIKOLOV, Tomas *et al*. Efficient Estimation of Word Representations in Vector Space. **ARXIV,** 2013. Disponível em: <https://arxiv.org/pdf/1301.3781.pdf>. Acesso em: 18 de out. de 2020.

MICROSOFT, **O que é um modelo de machine learning,** 2019. Disponível em: <https://docs.microsoft.com/pt-br/windows/ai/windows-ml/what-is-a-machine-learning-model#:~:text=Um%20modelo%20de%20machine%20learning%20%C3%A9%20um%20arquivo%20que%20foi,recognize%20certain%20types%20of%20patterns.>. Acesso em: 2 de nov. de 2020.

NILC. **USP**, 2017. Repositório de Word Embeddings do NILC. Disponível em: <http://nilc.icmc.usp.br/nilc/index.php/repositorio-de-word-embeddings-do-nilc>. Acesso em: 18 de out. de 2020.

PINTO, Sara. Processamento de Linguagem Natural e Extração de Conhecimento. **Universidade de Coimbra**, 2015. Disponível em: <https://estudogeral.uc.pt/bitstream/10316/35676/1/Processamento%20de%20Linguagem%20Natural%20e%20Extracao%20de%20Conhecimento.pdf>. Acesso em: 18 de out. de 2020.

RODRIGUES, Jéssica. O que é o Processamento de Linguagem Natural? **Medium,** 2017. Disponível em: <https://medium.com/botsbrasil/o-que-%C3%A9-o-processamento-de-linguagem-natural-49ece9371cff>. Acesso em: 22 de ago. de 2020.

SANTOS, Gisélia. Análise fonético-acústica das vogais orais e nasais do português: Brasil e Portugal. **Universidade Federal de Goiás**, 2013. Disponível em: <https://repositorio.bc.ufg.br/tede/bitstream/tede/3719/5/Tese-%20Gis%c3%a9lia%20Brito%20dos%20Santos%20-%202013.pdf>. Acesso em: 18 de out. de 2020.

SCIKIT-LEARN. **Scikit-learn**, 2007. Disponível em: <https://scikit-learn.org/stable/index.html#>. Acesso em: 2 de nov. de 2020.

TOMALOK, Everton. Word2Vec e sua importância na etapa de pré-processamento. **Medium**, 2019. Disponível em: <https://medium.com/@everton.tomalok/word2vec-e-sua-import%C3%A2ncia-na-etapa-de-pr%C3%A9-processamento-d0813acfc8ab>. Acesso em: 12 de set. de 2020.

WOOD, Thomas. What is the F-score. **DeepAI**. Disponível em: <https://deepai.org/machine-learning-glossary-and-terms/f-score>. Acesso em: 2 de nov. de 2020.

7. AGRADECIMENTOS

Em primeiro lugar, a Deus, que me forneceu esperanças e forças para continuar lutando. Ao Prof. Dr. Carlos Eduardo Câmara e ao Prof. Esp. Rodrigo Saito, não somente por serem profissionais incríveis e preparados, mas também por serem amigos que se preocupam com o aluno e fazem de tudo para nos ajudar; agradeço-lhes por todo conhecimento passado, estímulo, orientação e paciência. Agradeço também por toda essa jornada que passamos e que muito me ensinou. A todos professores do curso de Ciência da Computação do Centro Universitário Padre Anchieta, que durante esses quatro anos me ajudaram de alguma forma. A minha mãe, D. Leonice Aparecida Müller Vrech, pelo exemplo que sempre me deu, por me mostrar o caminho certo e me apoiar nas minhas decisões. Ao meu pai, Sr. Gil Aparecido Vrech, por sempre me ensinar a correr atrás dos meus ideais, me apoiar e ser um exemplo de perseverança. Ao meu irmão, Paulo Müller Vrech, por me incentivar a ingressar e seguir na área da computação, me fornecendo suporte, mentoria e estímulos para continuar. A todos os amigos, em especial Renato Cruz, Júnior Vitor, Arthur Sinelli, Christian Sanches, Luiz Mário, Leonardo Piusi, Luis Henrique, Lucas Timóteo, Felipe Andreas, Lyncoln Nascimento e Lucas Santos, por todo o apoio durante o desenvolvimento do trabalho. Aos meus amigos e companheiros de trabalho, em especial Jean Codogno, Renato Júnior, Cleberson Santos e Eduardo Mizani, por toda a experiência e ensinamentos passados para mim. Ao Centro Universitário Padre Anchieta, por oferecer toda a estrutura necessária para o desenvolvimento deste trabalho.