UNIVERSIDADE FEDERAL DO AMAZONAS - UFAM

FACULDADE DE TECNOLOGIA - FT

ENGENHARIA DA COMPUTAÇÃO

Estudo comparativo de desempenho de modelos de *Embedding* de Sentenças para agrupamento de artigos

Matheus Serrão Botto Barbosa

Manaus - AM

Setembro 2022

Matheus Serrão Botto Barbosa

Estudo comparativo de desempenho de modelos de *Embedding* de Sentenças para agrupamento de artigos

Monografia submetida à avaliação, como requisito parcial, para a obtenção do título de Bacharel em Engenharia da Computação pela Faculdade de Tecnologia, Universidade Federal do Amazonas

Orientador

Prof Dr. Moisés Gomes de Carvalho

Universidade Federal do Amazonas - UFAM

Faculdade de Tecnologia - FT

Manaus - AM

Setembro 2022

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Barbosa, Matheus Serrão Botto
B238e Estudo comparativo de desempenho de modelos de Embedding de Sentenças para agrupamento de artigos / Matheus Serrão Botto Barbosa . 2022
59 f.: il. color; 31 cm.

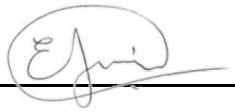
Orientador: Moisés Gomes de Carvalho
TCC de Graduação (Engenharia da Computação) - Universidade Federal do Amazonas.

1. Embedding de Sentenças. 2. Web Scraping. 3. K-Means. 4. Análise de desempenho. I. Carvalho, Moisés Gomes de. II. Universidade Federal do Amazonas III. Título

Monografia de Graduação sob o título Estudo comparativo de desempenho de modelos de *Embedding de Sentenças* para agrupamento de artigos apresentada por Matheus Serrão Botto Barbosa e aceita pelo Instituto de Computação da Universidade Federal do Amazonas, sendo aprovada por todos os membros da banca examinadora abaixo especificada:

Dr. Moisés Gomes de Carvalho
Orientador
Instituto de Computação
Universidade Federal do Amazonas

Dr. David Fernandes de Oliveira
Instituto de Computação
Universidade Federal do Amazonas



Dr. Eduardo James Pereira Souto
Instituto de Computação
Universidade Federal do Amazonas

Manaus - AM, 22 de setembro de 2022.

A minha família e amigos, em especial meu pai, que sempre se preocupou com a minha educação e a de meus irmãos.

Estudo comparativo de desempenho de modelos de *Embedding* de Sentenças para agrupamento de artigos

Autor: Matheus Serrão Botto Barbosa

Orientador: Prof Dr. Moisés Gomes de Carvalho

Resumo

Neste trabalho é analisado o desempenho de modelos de *Embedding* de Sentenças para o problema de agrupamento de artigos. Foi desenvolvido um sistema de *Web Scraping* que realizou a coleta de 58716 metadados de artigos. Foi desenvolvido um sistema de análise de desempenho de modelos de *Embedding* de Sentenças com base no agrupamento utilizando o algoritmo K-Means que é treinado com a base de artigos coletados assim calculando as métricas dos grupos resultantes e verificando o impacto dos *embedddings* gerados por cada modelo. Os modelos analisados são Doc2Vet, InferSent, Sentence-BERT e Universal Sentence Encoder. Nos cenários definidos, que variam desde o uso das 9 classes de artigos para agrupamento com todas as 58716 amostras da base de dados até 2 classes de artigos com a limitação de 500 amostras por classe, o Universal Sentence Encoder apresentou melhores valores de v-measure e índice Rand ajustado.

Palavras-chave: *Embedding* de Sentenças, *Web Scraping*, K-Means, Análise de desempenho.

Estudo comparativo de desempenho de modelos de *Embedding* de Sentenças para agrupamento de artigos

Autor: Matheus Serrão Botto Barbosa

Orientador: Prof Dr. Moisés Gomes de Carvalho

Abstract

In this work, the performance of *Sentece Embedding* models for the article clustering problem is analyzed. A *Web Scraping* system was developed and collected 58716 article metadata. A performance analysis system of *Sentece Embedding* models was developed based on clustering using the K-Means algorithm that is trained with the base of collected articles and calculating the metrics of the resulting clust ers and verifying the impact of the *embeddings* generated by each model. The analyzed models were Doc2Vet, InferSent, Sentence-BERT and Universal Sentence Encoder. In the defined settings, which vary from using 9 classes with all 58716 samples from the database to 2 classes with a limitation of 500 samples per class, the Universal Sentence Encoder showed better values of v-measure and adjusted Rand index.

Keywords: Sentence Embedding, Web Scraping, K-Means, Performance analysis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1

INTRODUÇÃO

Artigos são a base para a construção de novos conhecimentos científicos. A partir de um artigo alvo não é trivial mapear os artigos que sejam semelhantes ao mesmo, o processo de revisão de literatura requer um processo manual de investigação de vários possíveis artigos que sejam correlacionados ao artigo alvo.

Hoje existem mecanismos de buscas para artigos, porém a busca é baseada em palavras ignorando o contexto em que são utilizadas sendo assim uma análise estática para encontrar-se correlação nos artigos.

Nos últimos anos devido ao alto volume de dados textuais, resultante da evolução do processo de digitalização, que podem vir a conter informações e padrões implícitos que são úteis para os mais diversos domínios resultou-se na demanda de métodos e técnicas para processar dados textuais.

Os avanços na área de Processamento de Linguagem Natural permitem explorar novos métodos para análise de textos. Hoje existem métodos de análise de dados textuais que utilizam-se de lógica semântica, ou seja, a análise é feita em cima do conceito que o texto possui e não das palavras em si.

A possibilidade de usar análise semântica para encontrar-se relação entre textos abre uma nova possibilidade para mecanismos de busca.

Embedding de Sentenças são métodos para o processamento de linguagem natural capazes de representar sentenças no formato de vetores de números reais preservando a sua semântica. É possível comparar os *embeddings* de sentenças e obter o resultado do

quanto as sentenças são relacionadas ou similares do ponto de vista semântico.

Visando explorar essa possibilidade no contexto de comparação e correlação de artigos é promissor o uso de métodos de análise semântica em cima do conteúdo de artigos visando otimizar o processo de revisão de literatura.

1.1 Contextualização ou definição do problema

Existem diferentes métodos capazes de realizar uma análise semântica de um texto e a partir desta análise realizar comparações com outros textos. Sendo assim é necessário se ter uma ideia de desempenho para que seja possível comparar o desempenho de tais métodos.

A exploração da capacidade de tais métodos nos diversos cenários em cima de dados textuais de artigos podem ser usadas para a implementação de sistemas de mecanismos de busca de alta qualidade que facilitem o processo de revisão da literatura.

1.2 Objetivos

Esse trabalho apresenta os seguintes objetivos:

1.2.1 Objetivos Gerais

- Analisar o desempenho de diferentes métodos de *Embedding* de Sentenças em diferentes cenários de agrupamento

1.2.2 Objetivos Específicos

- Criar uma base de dados de artigos de temas pré-definidos a partir de técnicas de *Web Scraping*.
- Utilizar ferramentas de processamento de texto para melhorar desempenho dos

modelos.

- Utilizar modelos de *Embedding* de Sentenças: Doc2Vec, SentenceBERT, InferSent e Universal Sentence Encoder com a base dados de artigos.
- Treinar e avaliar modelos de agrupamento K-Means a partir dos *embeddings* obtido dos modelos de *Embedding* de Sentenças.

1.3 Organização do Trabalho

O trabalho a seguir divide-se da seguinte forma: No capítulo 2 é apresentado a representação teórica que consta de forma resumida e simplificado as técnicas e tecnologias a serem utilizadas para a modelagem e desenvolvimento dos sistemas de *Web Scraping* e *Sentence Embedding*, consta a introdução teórica dos modelos de *Sentence Embedding* a serem analisados.

No capítulo 3 relata-se a metodologia utilizada, onde é explanada de forma mais técnica e prática de como foi a modelagem do sistema *Web Scraping* e como foi desenvolvido o sistema *Sentence Embedding* detalhando sobre a definição de hiperparâmetros, execução dos treinamentos necessários e aplicação do algoritmo K-Means.

No capítulo 4 são apresentados os resultados obtidos dos sistemas de *Web Scraping* e *Sentence Embedding* e a interpretação dos resultados obtidos.

2

FUNDAMENTOS

Neste capítulo são apresentados, de maneira simplificada, os principais conceitos e técnicas utilizadas para a elaboração dos experimentos contidos neste trabalho. A maioria dos conceitos são pertinentes a Ciência de Dados e Aprendizado de Máquina.

2.1 Ciência de Dados

Nos últimos anos há uma larga quantidade de dados que derivam das mais diversas atividades. Dados são recebidos de diversas fontes e disponíveis em vários formatos. O processo de digitalização devido a evolução da tecnologia nas últimas décadas foi um dos fatores dessa abundância de dados.

Dado é um recurso básico dos campos de Ciência, Tecnologia e Gestão. A análise de dados pode resultar na descoberta de padrões implícitos que podem solucionar problemas ou melhorar processos existentes. Ciência de Dados é a área que analisa e manipula dados a fim de extrair informações para os mais diversos fins, dados esses que podem estar em formatos como áudio, imagem, vídeo ou texto. (ŽIŽKA; DAŘENA; SVOBODA, 2019)

2.1.1 Mineração de Texto

Dados textuais são altamente relacionados a muitas tarefas humanas, a análise desses dados pode resultar em informações úteis para praticamente todos os domínios.

Textos são escritos em linguagem natural cujo é um dos formatos mais comuns de comunicação, porém a linguagem natural não é o melhor formato para ser usado em algoritmos, todavia há uma grande quantidade de dados textuais disponíveis, resultando na demanda de novos métodos computacionais para a exploração desta vasta coleção de dados textuais. Devido a tal demanda a área de Mineração de Texto se tornou bastante popular e atrativa. Mineração de Texto é o campo que usa ferramentas analíticas para identificar e explorar padrões em coleções textuais (ŽIŽKA; DAŘENA; SVOBODA, 2019).

2.1.2 Web Scraping

Web Scraping é o processo de extrair dados da World Wide Web de modo automatizado a fim de exploração e manipulação dos dados obtidos (CHAPAGAIN, 2019). A popularidade da internet e a expansão da World Wide Web em todo Globo resulta no grande crescimento na quantidade de dado bruto, tal dado que se coletado, processado e organizado pode ser útil para extração de informações e padrões implícitos.

Hoje a Web é uma fonte de informações em tempo real e dinâmica, e páginas webs não são simples textos, são documentos que são construídos a partir de várias tecnologias como HTML, JavaScript e CSS. A necessidade da obtenção dos dados da web somados a complexidade de páginas web resultaram em ferramentas e técnicas para o processo de Web Scraping.

2.1.3 Engenharia da Dados

Dados coletados de diversas fontes frequentemente não estarão no formato ideal para fins de análise, portanto é necessário um processo intermediário entre a coleta de dados e a análise de dados.

Figura 1 – Fluxo simplificado de um processo de Web Scraping.

Engenharia de Dados é o desenvolvimento de processos que a partir de dados brutos produzem dados de alta qualidade que podem ser úteis para fins de análise e aprendizado de máquina (REIS; HOUSLEY, 2022). O engenheiro de dados é responsável pelo ciclo de vida do dado, as principais etapas do ciclo de vida de um dado são: geração, armazenamento, consumo, transformação e distribuição.

A qualidade dos mecanismos usados para realizar essas etapas influenciam nos resultados dos processos que utilizaram esses dados.

2.2 Aprendizado de Máquina

Aprendizado de Máquina é o subcampo de Inteligência Artificial mais estudado e praticado atualmente (GÉRON, 2019). Aprendizado de Máquina é a ciência cujo o objetivo é desenvolver algoritmos com a capacidade de aprender a partir de dados.

Os algoritmos desenvolvidos tem a capacidade de simular as habilidades intelectuais dos organismos vivos. Aprendizado de Máquina é uma ferramenta presente no dia a dia. Aplicações como reconhecimento facial, reconhecimento de voz, geração de imagens completamente originais, sistemas de recomendação são exemplos do uso de aprendizado de máquina em aplicações do cotidiano.

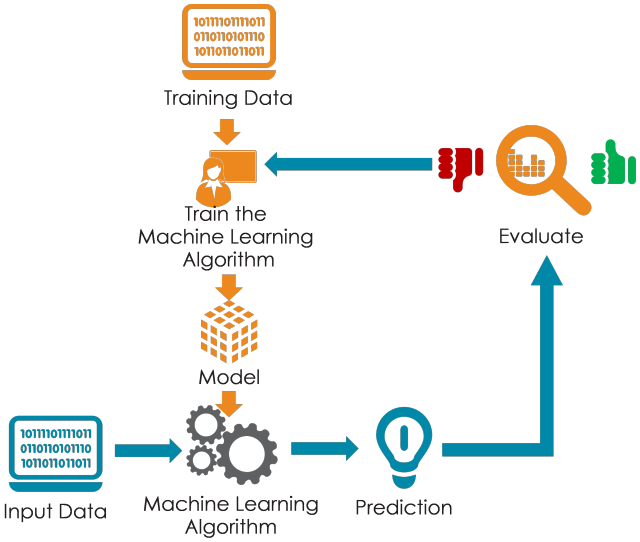


Figura 2 – Fluxo de desenvolvimento de um algoritmo de aprendizado de máquina.

2.2.1 Processamento de Linguagem Natural

Linguagem natural é linguagem que os humanos usam para compartilhar informações uns com os outros.

Processamento de Linguagem Natural é o subcampo de Inteligência Artificial cujo objetivo é processar linguagem natural, geralmente esse processamento envolve transformar linguagem natural, frequentemente preservando algum aspecto semântico, em um dado cujo pode ser usado por um algoritmo computacional (HAPKE; HOWARD; LANE, 2019). Tais dados podem ser usados para a geração de linguagem natural com lógica semântica.

2.2.2 Embedding

Embedding é um vetor de relativamente baixa dimensão cujo representa informações em altas dimensões (EMBEDDINGS, 2022). A capacidade de representar dados de altas dimensões em vetores de baixa dimensão facilita o uso de tais dados para fins como Aprendizado de Máquina. Sentence Embedding é a capacidade de representar frases, documentos, textos em vetores de baixa dimensão, muitas vezes de tamanho

fixo, preservando o sentido semântico das frases, documentos, textos.

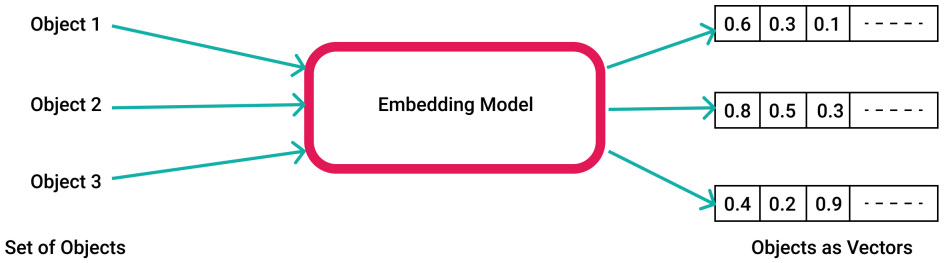


Figura 3 – Comportamento simplificado de um modelo de embedding

2.2.2.1 Doc2Vec

Sendo uma extensão do Word2Vec (MIKOLOV et al., 2013), modelo criado capaz de gerar uma representação numérica de palavras em um espaço n-dimensional. O Doc2Vec (LE; MIKOLOV, 2014) é um modelo capaz de gerar uma representação numérica para documentos em um espaço n-dimensional. Word2Vec pode ser implementado usando duas possíveis opções de redes neurais, Continuous Bag-of-Words (CBOW) ou Skip-Gram. Abaixo uma imagem pra explicar o funcionamento do Continuous Bag-of-Words.

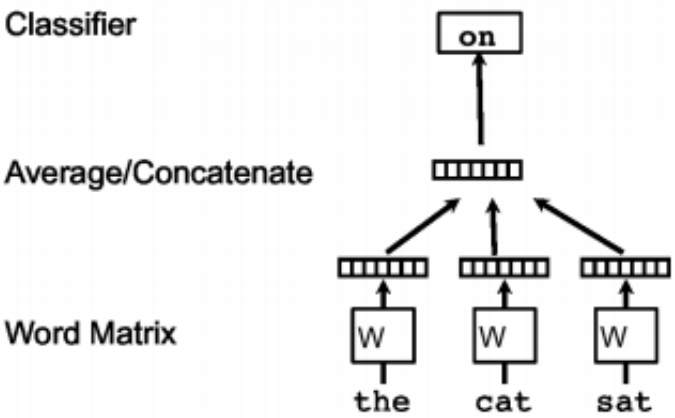


Figura 4 – Funcionamento do Continuous Bag-of-Words.

Em um frase, cada palavra da frase é representada por um vetor, cada vetor é calculado a partir da média dos vetores das palavras adjacentes, tal processo é capaz de

preservar o sentido semântico da palavra. O Doc2Vec usa um algoritmo que é uma extensão do Continuous Bag-of-Words (CBOW) chamado de Distributed Memory version of Paragraph Vector (PV-DM). Abaixo uma imagem pra explicar o funcionamento do Distributed Memory version of Paragraph Vector.

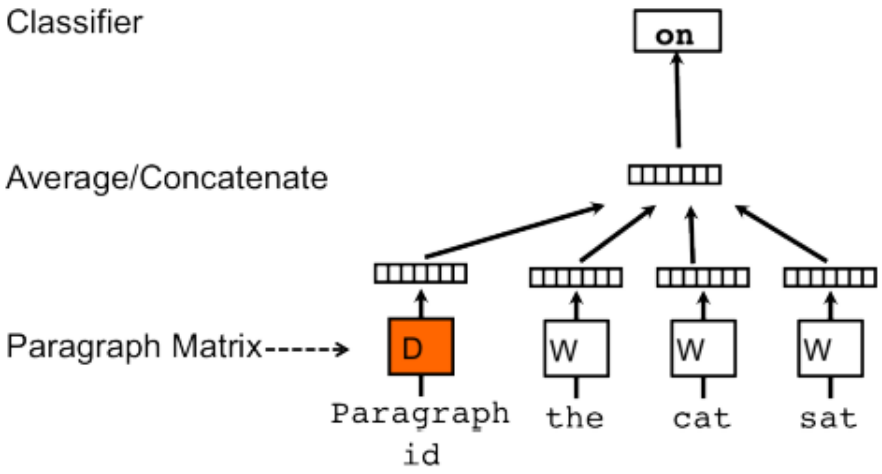


Figura 5 – Funcionamento do Distributed Memory version of Paragraph Vector.

Em um frase, cada palavra da frase é representada por um vetor, cada vetor é calculado a partir da média dos vetores das palavras adjacentes **com a inclusão no calculo do vetor D cujo é o vetor que representa a frase em si**.

Tanto no Continuous Bag-of-Words (CBOW) como Distributed Memory version of Paragraph Vector ocorre o processo de treinamento cujo atualiza os pesos da rede neural e resulta com que cada palavra analisada no conjunto de treino seja representada por um vetor W, todavia no Distributed Memory version of Paragraph como também é usado o vetor D que representa a frase, **cada frase analisada no conjunto de treinamento é representada por um vetor D que preserva o conceito semântico da frase**.

Usando a rede neural Distributed Memory version of Paragraph pós-treinamento é possível fazer a inferência de novas frase para a gerar a sua representação numérica.

2.2.2.2 Sentence-BERT

BERT é um modelo de aprendizado profundo do Google para PLN que se encontra no estado da arte para a resolução de problemas como similaridade textual semântica de uma par de frases. O grande diferencial da rede BERT se comparado com outros modelos de PLN é a capacidade de modelo em identificar e segregar o significado da palavra no contexto utilizado, pois uma mesma palavra pode ter significados completamente diferentes dependendo do contexto.

Um problema da rede BERT é que para se calcular a similaridade entre um par de frases é preciso inserir ambas sentenças no modelo, o que resulta em uma massiva sobrecarga computacional quando se aumenta o número de sentenças a serem analisadas cresce.

Sentence-BERT (REIMERS; GUREVYCH, 2019) é uma modificação da rede BERT com a capacidade de gerar representações numéricas de frases que podem ser comparadas usando distância cosseno.

2.2.2.3 InferSent

InferSent (CONNEAU et al., 2017) é um modelo de representação numérica de frases supervisionado tendo sido introduzido pelo Facebook AI Research. A principal característica desse modelo é o fato dele ter sido treinado com o Stanford Natural Language Inference (SNLI), cujo é uma base de dados que consiste em 570 mil pares de sentenças em inglês geradas por humano que foram rotuladas em uma de três possíveis categorias: vinculação, contradição e neutra.

Foram sugeridas várias arquiteturas para a geração da representação numérica de frases, a melhor relatada no artigo foi a BiLSTM com pooling maxímo/médio, uma rede bidirecional LSTM que calcula n-vetores para as n-palavras da frase, cada vetor é uma concatenação da saída de uma LSTM no qual é inserida a frase no sentido convencional e a saída de uma LSTM no qual é inserida a frase no sentido oposto. Tendo n-vetores resultantes é aplicado pooling máximo ou pooling médio em cada vetor resultante assim gerando um vetor final que é a representação numérica da frase.

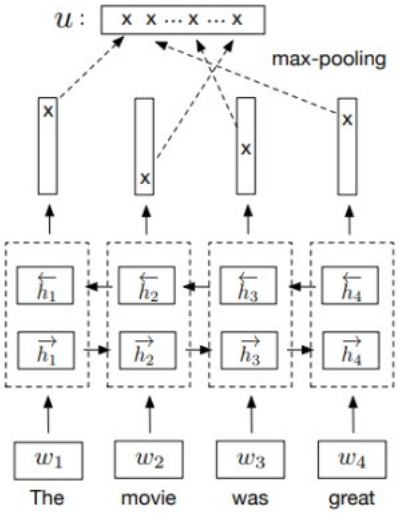


Figura 6 – Fluxo de InferSent para a geração de embeddings.

2.2.2.4 Universal Sentence Encoder

Introduzido pela Google, o Universal Sentence Encoder (CER et al., 2018) é um modelo de representação numérica de frases que se destaca pela sua capacidade de transferência de aprendizado que é carregada nas representação numéricas resultantes do modelo que apresentam bons desempenhos em múltiplas tarefas de PLN. Como, por exemplo, análise de sentimentos, classificação de textos e cálculo de similaridade de frases.

O Universal Sentence Encoder possui duas opções de encoders: Transformer e Deep Averaging Network(DAN), ambos encoders conseguem resultar em representações numéricas de palavras ou de frases. Ambos geram um vetor de 512 dimensões, sendo o Transformer modelado para atingir grande acurácia ao custo de alta complexidade do modelo e alto consumo de recursos, e o Deep Averaging Network opera de maneira mais eficiente ao custo de leve redução de acurácia se comparado com o Transformer.

2.2.3 K-Means

K-Means é um simples algoritmo de agrupamento cujo particiona uma base de dados em K grupos onde cada amostra pertence ao grupo cujo há a menor distância entre a

amostra e o centro do grupo(GÉRON, 2019).

Sendo K o número de grupos, cada grupo possui um centróide, que pode ser definido ou não, cada instância é atribuída a um grupo de forma que a instância pertença ao grupo do centróide mais próximo. Se os rótulos das instância são conhecidos pode-se localizar o centróide com facilidade, mas senão escolhe-se K centróides a partir de algum método de inicialização, em seguida as instâncias são rotuladas, os centróides são atualizados, novamente as instâncias são rotuladas e assim sucessivamente até que o centróide varie minimamente, ou seja, o treinamento do modelo convergiu.

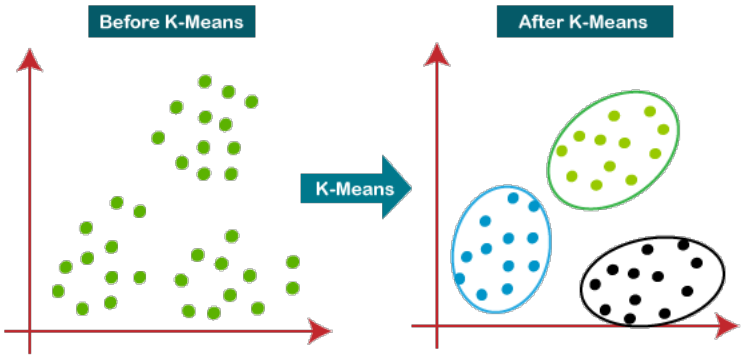


Figura 7 – Exemplo de uso do algoritmo K-Means.

2.2.4 Métricas de agrupamento

Metrificar o desempenho de algoritmos de agrupamentos não é uma tarefa trivial. As métricas devem analisar a qualidade do agrupamento baseado na comparação da separação de dados realizados pelo algoritmo versus a separação de dados baseadas em alguma característica dos dados agrupados. Algumas da métricas mais adotadas pela literatura para a avaliação de agrupamentos são (CLUSTERING..., 2022).

- Homogeneidade
- Completude
- V-Measure
- Índice de Rand Ajustado

- Coeficiente de Silhouette

Homogeneidade mede o quanto cada grupo contém apenas membros de uma única classe, os valores se limitam entre 0 e 1, sendo 1 o melhor cenário.

Completude mede o quanto todos os membros de uma classe estão contidos no mesmo grupo, os valores se limitam entre 0 e 1, sendo 1 o melhor cenário.

V-measure é um harmônico entre homogeneidade e completude, dada pela seguinte fórmula:

$$v = \frac{(1+\beta) \times \text{homogeneidade} \times \text{completude}}{(\beta \times \text{homogeneidade} + \text{completude})} \tag{2.1}$$

Beta é convencionalmente definido como 1, valores menores que 1 atribuem maior peso a homogeneidade e valores maiores que 1 atribuem maior peso a completude.

Índice Rand mede a similaridade entre duas rotulações. O índice Rand ajustado é uma variação que faz com que rotulações aleatórias tenham valores próximos a zero, os valores se limitam entre -1 e 1, sendo 1 o melhor cenário.

Coeficiente de Silhouette avalia a densidade dos grupos que foram criados, não sendo necessário conhecer os rótulos das amostras. Os valores se limitam entre -1 e 1, sendo 1 o melhor cenário.

3

METODOLOGIA

Neste capítulo é apresentado a metodologia adotada para a elaboração deste trabalho. São detalhados as configurações selecionadas para a execução dos sistemas e experimentos, sendo explicado de maneira encadeada tópicos como o desenvolvimento do sistema de *Web Scrapping* e as tomadas de decisões para a execução do mesmo até a explicação de como utilizar o algoritmo de agrupamento K-Means para analisar o desempenho dos modelos de *Embedding* de Sentenças.

3.1 Sistema de Web Scraping

A primeira parte do trabalho consistiu na escolha de temas de artigos e por consequência na procura de bases de dados que tivessem artigos de tais temas. Para esse estudo foram selecionados os seguintes temas:

- Visão Computacional
- Cibersegurança
- *Sentence Embedding*
- Veículos Autônomos
- Internet das Coisas
- 5G

- *Big Data*
- Ansiedade
- Política

Os temas de Engenharia foram selecionados com intuito de observar-se a variabilidade intraclasse. Os artigos do tema de Engenharia foram limitados entre os anos de 2019 e 2022. Ansiedade e Política são temas pertencentes, respectivamente, as áreas de Psicologia e Ciências Sociais. No estudo será possível observar a correlação entre as áreas(Engenharia, Psicologia e Ciências Sociais) e a correlação dos temas entre si. A fonte de dados para os artigos de Engenharia foi o IEEE (IEEE. . . , 2022), para os artigos de Psicologia foi o Frontiers (FRONTIERS, 2022)e para os artigos de Ciências Sociais foi o University of Chicago Press (UNIVERSITY. . . , 2022).

Já definidos os temas e as bases de dados de artigos disponibilizadas por bibliotecas de artigo científico, pode-se desenvolver o sistema de Web Scraping para coleta automatizada. O sistema de Web Scraping foi criado usando Python sendo as duas principais bibliotecas usadas o Selenium e o Beautiful Soup. O sistema é baseado em pipeline composto nas seguintes partes:

1. Seleção de tema
2. Construção de URL para consulta
3. Acesso a página via Selenium
4. Extracão e formatação via Beautiful Soup
5. Registro dos metadados em planilha

Em um processo iterativo, um tema é selecionado, e.g, Visão Computacional, o tema possui alguns dados associados para a realização do WebScrap, o site com a biblioteca de artigos associada, no caso de Visão Computacional como já descrito foi definido o IEEE, também é associado um número de artigos a serem coletados, dependendo do tema a biblioteca de artigos pode ter ou não a quantidade desejada,

no caso de Visão Computacional foi definido 10000 artigos e como é um tema de Engenharia acontece a restrição de ano publicado entre 2019 e 2022.

A partir dessas informações é realizada a construção de uma URL que servirá como base para a coleta de artigos via Selenium. Para o desenvolvimento de um mecanismo de construção de URL é preciso entender como o mecanismo de busca do site alvo opera.

A partir da URL construída é possível acessar o site com o banco de dados de artigos do tema selecionado, ou seja, usando Selenium é possível acessar o site de forma automatizada. Acessando o site possui-se o acesso ao código fonte do site que contém os metadados dos artigos do tema selecionado.

Alguns do metadados que o IEEE provê por artigo são:

- Título
- Resumo
- Autores
- Palavras chave
- Métricas
- Notas de rodapé

Usando Selenium e Beautiful Soup tem-se acesso a cada uma das informações acima por artigo de maneira consistente e automatizada via código Python. O Selenium é responsável por obter o código fonte da página Web a partir da URL construída e também encontrar as informações de interesse a partir de alguma miníma instrução, o código fonte dá pagina Web é escrito em HTML. O Beautiful Soup é responsável extrair os dados de interesses retirando os elementos HTML garantido que o dado extraído só contenha as informações de interesse.

Para esse estudo as informações coletadas por artigo foram título e resumo, esse processo se repete por artigo até a quantidade definida pelo tema seja atingida ou o banco de dados não tenha mais artigos nas restrições definidas.

Como para cada área são sites diferentes, o processo de construção de URL e acesso a página via Selenium varia, mas independente do site alvo os procedimentos são os mesmos variando apenas o código Python.

Os dados por tema são registrados em uma planilha e posteriormente armazenados em um banco de dados Mongo.

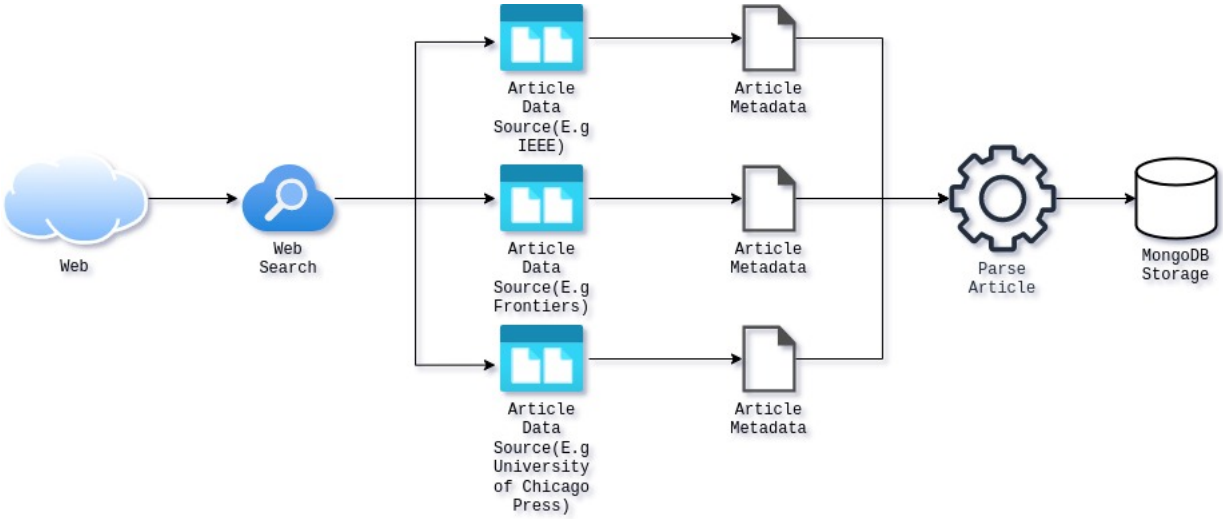


Figura 8 – Componentes do sistema WebScrap.

3.2 Modelos de *Embedding* de Sentenças

Já possuindo as coleções de artigos dos temas definidos armazenadas em um banco de dados Mongo para fácil manipulação, começa-se o processo de *embedding* de sentenças.

Para o processo de *embedding* de sentenças foi usado o resumo do artigo para ser processado e posteriormente usado para o treino de modelo.

Cada resumo antes de ser processado primeiro recebe o seguinte tratamento:

1. Conversão para caixa baixa
2. Remoção de palavras com menos de 2 caracteres
3. Remoção de palavras com mais de 15 caracteres
4. Remoção de palavras que possuem caracteres numéricos

5. Vetorização do texto

A conversão para caixa baixa deve-se a necessidade de não diferenciação da mesma palavra escrita em maiúsculo ou minúsculo. A remoção de palavras com menos de 2 caracteres e mais de 15 caracteres é, respectivamente, para que palavras com pouco significado a agregar, ou seja, comuns em muitos textos e para que palavras que derivem de algum erro de processamento, ruído, não tenham impacto na análise. A remoção de palavras com caracteres numéricos se deve ao alto uso de métricas quantitativas em artigos que pode vir a causar uma falsa correlação entre os artigos por causa de tais números.

Aplicando esse processo para todos os resumos da base de dados é possível começar o processo de geração de *embedding* de sentenças. Para fins de eficiência foi criado uma constante ***tokenized_sentences*** usando a biblioteca Pickle, tal constante armazena todos os dados necessários para os experimentos a serem realizados.

Título	Tema	Resumo vetorizado
'Automated Distributed Electric Vehicle Controller for Residential Demand Side Management'	autonomous_vehicle	['electric', 'vehicles', 'evs', 'are', 'recently', 'gaining', ...
'Machine Learning Techniques for Classifying Network Anomalies and Intrusions'	cybersecurity	['using', 'machine', 'learning', 'techniques', 'to', 'detect', ...
'Unburials, Generals, and Phantom Militarism: Engaging with the Spanish Civil War Legacy'	political	['karma', 'and', 'grace', 'are', 'grammars', 'for', 'material', ...
'IoT Based Smart Farming: Are the LPWAN Technologies Suitable for Remote Communication?'	iot	['the', 'internet', 'of', 'things', 'iot', 'has', 'changed',...

Tabela 1 – Amostras de informações contidas na constante ***tokenized_sentences***.

Para o uso do modelo Doc2Vet foi usado a biblioteca Gensim para usar importar o modelo Doc2Vet. Cada resumo é transformado em TaggedDocument sendo a concatenação dos mesmos a base de dados que será usada para treinamento do Doc2Vet.

Em seguida o modelo é treinado. Os embeddings que modelo inferirá são de 30 dimensões, palavras com frequência total inferior a 3 são descartas pelo modelo e o treinamento terá 100 iterações. Neste estado já é possível gerar embeddings de documentos usando o modelo resultante.

Para o uso do modelo Sentence-BERT foi usado um modelo Sentence-BERT pré-treinado, cujo não há a necessidade de ajustes para o seu uso. Neste estado já é possível gerar embeddings de documentos usando o modelo resultante.

Para usar o modelo InferSent é necessário uma preparação mínima de ambiente. Primeiramente deve-se baixar o arquivo cujo contém o modelo e baixar o modelo pré-treinado que gera embedding de palavras, o GloVe.

Após esse processo, é necessário realizar alguns parâmetros do modelo a ser treinado. O *batch size* definido em 64, a dimensionalidade do embedding de palavras definido em 300, a dimensionalidade do embedding de documentos definido em 2048, o tipo de pooling definido em pooling máximo, a frequência do dropout definido em 0 e a configuração de encoder a ser usada definido no tipo 2. Após esse processo inicia-se o treinamento do modelo. Neste estado já é possível gerar embeddings de documentos usando o modelo resultante.

Para usar o modelo Universal Sentence Encoder algumas bibliotecas auxiliares ao TensorFlow, sendo possível importar o modelo pré-treinado.

Neste estado já é possível gerar embeddings de documentos usando o modelo resultante. Com os 4 modelos preparados pra utilização é possível analisar o desempenho dos mesmos em diferentes cenários.

3.3 Agrupamento com o K-Means

Para cada modelo de *embedding* de sentenças foi gerado uma planilha que contém o embedding de cada resumo de artigo e seu respectivo tema. A partir disso é possível treinar modelos de K-Means e analisar o desempenho dos modelos de *embedding* de sentenças.

Uma das planilhas que contém os embeddings realizados por um modelo é selecionada. Resultando em um dataframe pronto para ser processado.

A partir do dataframe são isolados os valores $\mathbf{X}$ e $\mathbf{y}$, respectivamente, os embeddings e as classes dos artigos. Sendo os valores de X normalizados e y, nesse cenário, convertidos em valores numéricos.

Para a avaliação dos modelos de K-Means foram utilizadas as 5 métricas apresentadas no capítulo anterior: Homogeneidade, Completude, V-Measure, Índice de Rand Ajustado e Coeficiente de Silhouette.

Para cada planilha é gerado 3 modelos de K-Means, o primeiro e o segundo diferem no tipo de inicialização dos centroides dos grupos e o terceiro é feito depois que um processo de redução de dimensionalidade é feito nos embeddings.

Com as métricas a disposição é possível analisar o desempenho dos modelos de *embedding* de sentenças. Todo o processo de treinamento dos modelos de K-Means é estático, sendo o único fator variante os embeddings de qual modelo que está sendo consumido.

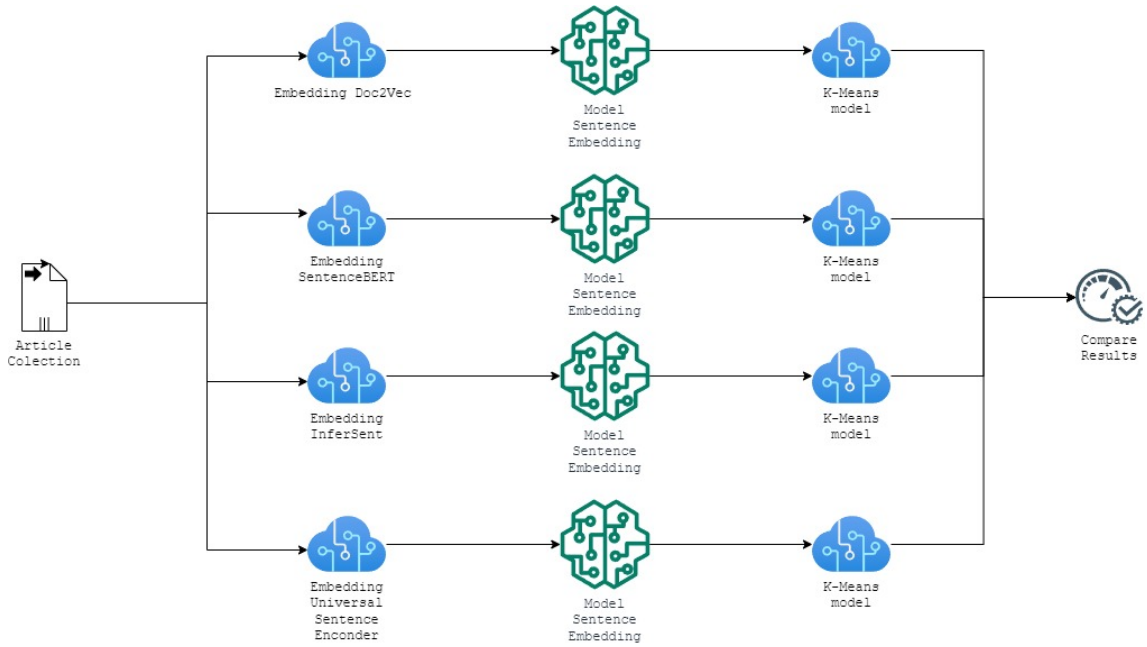


Figura 9 – Fluxo do sistema de análise de desempenho de modelos de embeddings de sentenças.

4

RESULTADOS

Neste capítulo é apresentado os resultados dos sistemas e experimentos contidos neste trabalho. São descritos valores quantitativos de desempenho da execução do sistema de *Web Scrapping* e são descritos os experimentos com *Sentence Embedding*, sendo cada experimento descrito e explicando os objetivos a serem alcançados com o tal experimento.

4.1 Hardware e Software utilizados

Todo o trabalho foi realizado na seguinte configuração de hardware:

- Processador: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs), 2.6GHz
- Memória: 32768MB RAM
- Sistema Operacional: Windows 10 Enterprise 64-bit
- Placa de vídeo: NVIDIA GeForce RTX 3060 Laptop GPU

Bibliotecas e softwares destacáveis para a execução do trabalho:

- Scikit-learn
- Selenium
- BeautifulSoup
- MongoDB

4.2 Sistema de Web Scraping

A construção de um sistema de Web Scraping com diferentes fontes de dados traz pro sistema uma carga indesejável de complexidade. O fato do sistema possuir diferentes fontes faz com que seja necessário inspecionar cada uma das fontes para a criação de um mecanismo de coleta que muitas vezes é único para aquela fonte, ou seja, não pode ser reaproveitado.

Mas independente da fonte de dados é possível analisar alguns aspectos desse sistema. Na tabela abaixo segue alguns dados sobre tempo de processamento e volume de dados.

Total de artigos coletados	58716
Tempo média de coleta por artigo	1,96 seg
Tempo total de coleta	31 hrs 58 min

Tabela 2 – Quantitativos do processo de Web Scraping.

O tempo de coleta não leva em conta as interrupções por indisponibilidade da fonte de dados, perda de conexão de internet ou problemas técnicos. A coleta de dados não foi realizada em apenas uma sessão

A distribuição da coleção de artigos é demonstrada na tabela abaixo.

Classe	Quantidade
Visão Computacional	10000
Cibersegurança	4000
Sentence Embedding	502
Veículos Autônomos	10000
5G	10000
Big Data	10000
Ansiedade	1378
Política	2578

Tabela 3 – Quantidades de artigos por classe.

A base apresenta desbalanceamento devido a quantidade de artigos publicados nas fontes de dados alvos. As classes de artigos pertencentes a área de Engenharia atingiram o máximo estabelecido do 10000 artigos por classe, com exceção da classe Cibersegurança e Sentence Embedding e como descrito anteriormente, os artigos de Engenharia possuem data de publicação entre 2019 e 2022.

As classes de artigo Ansiedade e Política apresentam baixa quantidade de artigos coletados mesmo sem possuir limitação de data de publicação.

4.3 Experimentos com Sentence Embedding

4.3.1 Características dos Embeddings

Cada modelo de sentence embedding gera embeddings (vetores) de tamanho N, sendo que N varia de modelo para modelo. Segue abaixo a descrição do tamanho do embeddings para cada modelo.

Modelo	Tamanho do embedding
Doc2Vec	30
Sentence-BERT	768
InferSent	4096
Universal Sentence Encoder	512

Tabela 4 – Dimensionalidade dos embedding gerados por modelo.

Todos os tamanhos foram fixos para todas as bases processadas, todos os embedding são do tipo ponto flutuante de precisão simples(números de 32 bits). O impacto do tamanho dos embeddings se nota principalmente no tempo de processamento dos embeddings no tempo de treino dos modelos K-Means.

4.3.2 Experimentos 1 - Treinamento com toda a base de dados

Neste experimento toda a base de dados é usada para treinamento dos algoritmos de K-Means. Sendo o objetivo identificar qual modelo desem melhor nestas condições.

Modelo	Inicialização	Tempo	Homo	Compl	V-Measure	IRA	Silhouette
Doc2Vet	k-means++	0.425s	0.418	0.377	0.397	0.294	0.050
	aleatória	0.330s	0.382	0.346	0.364	0.257	0.044
	baseada em PCA	0.119s	0.389	0.353	0.370	0.254	0.040
InferSent	k-means++	58.930s	0.144	0.147	0.145	0.075	0.037
	aleatória	45.625s	0.129	0.129	0.129	0.077	0.046
	baseada em PCA	25.131s	0.142	0.158	0.150	0.055	0.082
Sentence-BERT	k-means++	21.491s	0.336	0.319	0.327	0.262	0.039
	aleatória	13.250s	0.333	0.316	0.324	0.222	0.056
	baseada em PCA	3.479s	0.333	0.316	0.324	0.222	0.040
Universal Sentence Encoder	k-means++	9.567s	0.431	0.409	0.420	0.335	0.028
	aleatória	9.593s	0.418	0.400	0.409	0.313	0.052
	baseada em PCA	2.740s	**0.433**	**0.410**	**0.421**	**0.336**	0.038

Tabela 5 – Métricas do experimento 1.

Nota-se a redução de tempo considerável que a aplicação de PCA nos dados causa, levando a redução de tempo que variaram de 2,34 até 6,17 vezes mais rápido.

O Universal Sentence Encoder com inicialização baseada em PCA teve o melhor desempenho baseado no V-Measure e IRA e foi o único modelo que performou melhor com a aplicação de PCA no dados baseado no V-Measure, IRA e Silhouette. Tanto o Universal Sentence Encoder como o Sentence-BERT, com exceção do tempo de execução, não apresentaram resultados significamente diferentes com diferentes métodos de inicialização.

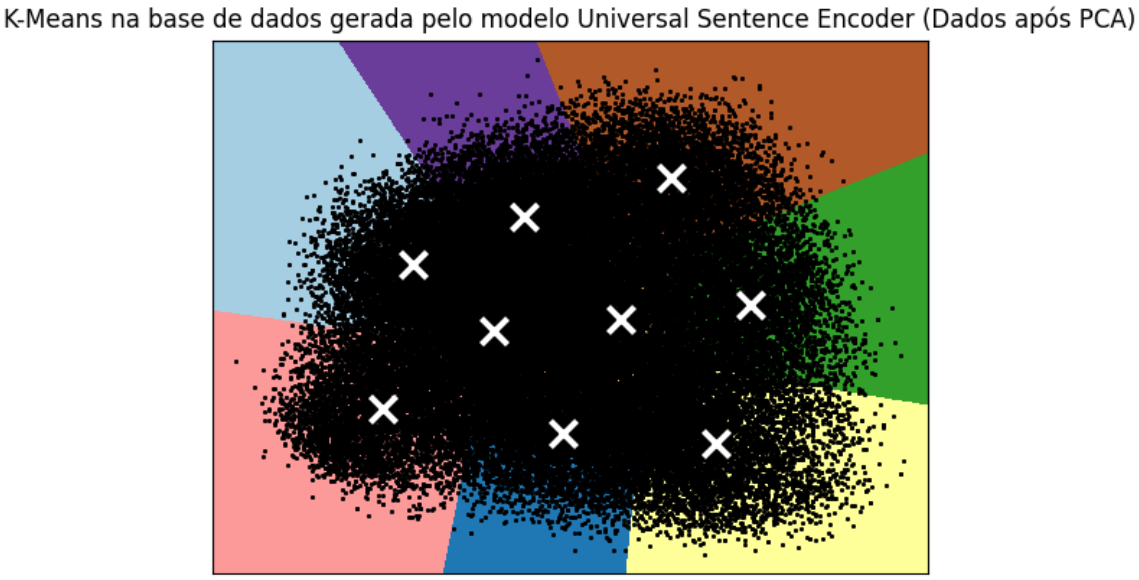


Figura 10 – Experimento 1 - Distribuição dos dados após PCA com algoritmo K-Means usando modelo Universal Sentence Encoder.

Na figura 10 consta a distribuição dos embeddings do modelo do Universal Sentence Encoder, os centroides estão marcados com um "X"e as áreas dos grupos são delimitadas por cor. Nota-se que os grupos formados pelo algoritmo K-Means são esparsos, todavia foram encontradas fronteiras bem definidas entre os grupos.

4.3.3 Experimento 2 - Treinamento com toda a base de dados balanceada

Neste experimento, dividido em 2 cenários, a base de dados foi uniformizada, a uniformização da base geralmente resulta em melhores resultados pois retira o viés que algum grupo com um maior número de amostras pode causar. Sendo o objetivo entender o desempenho de cada modelo em cenários de base uniformizada.

O primeiro cenário possui a condição de 500 amostras por classe, ou seja, 4500 amostras de treino. O segundo cenário possui a condição de 50 amostras por classe, ou seja, 450 amostras de treino.

4.3.3.1 Cenário 1 - 500 amostras por classe

Neste experimento a base de dados foi uniformizada, 500 amostras por classe, para treinamento dos algoritmos de K- Means. Sendo o objetivo identificar se os modelos possuem ganho de desempenho em uma base uniformizada.

Modelo	Inicialização	Tempo	Homo	Compl	V-Measure	IRA	Silhouette
Doc2Vet	k-means++	0.058s	0.477	0.484	0.481	0.387	0.049
	aleatória	0.037s	0.479	0.486	0.483	0.393	0.052
	baseada em PCA	0.014s	0.420	0.432	0.426	0.331	0.037
InferSent	k-means++	2.939s	0.233	0.263	0.247	0.111	0.066
	aleatória	2.356s	0.237	0.263	0.249	0.116	0.070
	baseada em PCA	1.380s	0.179	0.215	0.195	0.065	0.109
Sentence-BERT	k-means++	0.700s	0.463	0.479	0.471	0.378	0.071
	aleatória	0.535s	0.465	0.481	0.473	0.379	0.070
	baseada em PCA	0.162s	0.390	0.402	0.396	0.303	0.059
Universal Sentence Encoder	k-means++	0.370s	0.559	0.577	0.568	0.470	0.064
	aleatória	0.285s	**0.561**	**0.580**	**0.571**	**0.472**	0.052
	baseada em PCA	0.105s	0.506	0.532	0.519	0.409	0.038

Tabela 6 – Métricas do experimento 2 - base equalizada, 500 amostras por classe.

A uniformização da base de dados teve um impacto positivo em todos os modelos. O Universal Sentence Encoder com inicialização aleatória teve o melhor desempenho baseado em V-Measure e IRA, apesar de diferenciar pouco dos valores com inicialização k-means++ esse resultado é contra intuitivo pois é esperado que a inicialização k-means++ resulte em centroides que gerem grupos melhores.

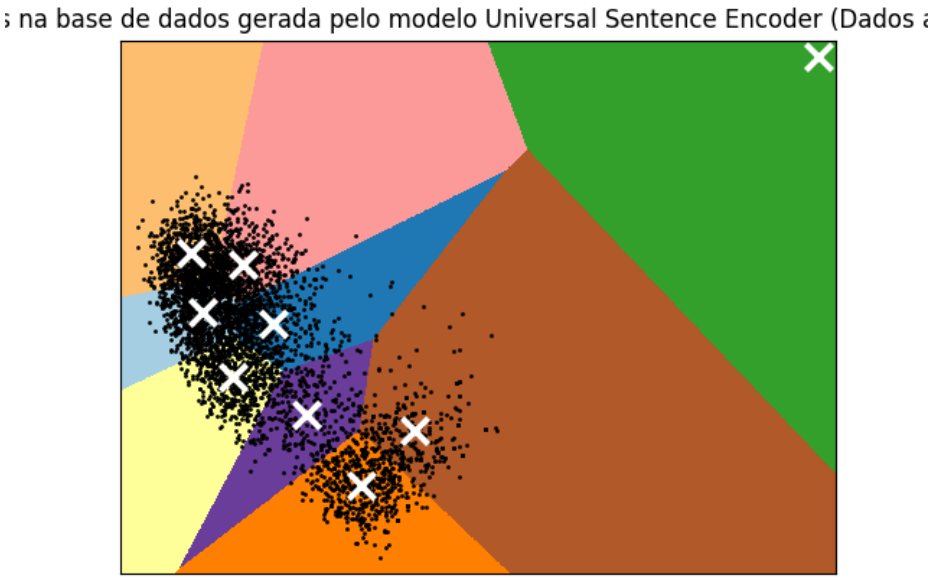


Figura 11 – Experimento 2 - Distribuição dos dados após PCA com algoritmo K-Means usando modelo Universal Sentence Encoder.

Na figura 11 consta a distribuição dos embeddings do modelo do Universal Sentence Encoder, nota-se que os grupos formados pelo algoritmo K-Means são mais densos se comparado com a distribuição da figura 10.

4.3.3.2 Cenário 2 - 50 amostras por classe

Neste experimento a base de dados foi uniformizada, 50 amostras por classe, para treinamento dos algoritmos de K- Means. Sendo o objetivo identificar se os modelos possuem ganho de desempenho com uma base reduzida.

Modelo	Inicialização	Tempo	Homo	Compl	V-Measure	IRA	Silhouette
Doc2Vet	k-means++	0.025s	0.479	0.486	0.483	0.353	0.050
	aleatória	0.017s	0.472	0.487	0.480	0.360	0.062
	baseada em PCA	0.006s	0.482	0.490	0.486	0.385	0.056
InferSent	k-means++	0.349s	0.270	0.297	0.283	0.112	0.059
	aleatória	0.270s	0.214	0.258	0.234	0.090	0.065
	baseada em PCA	0.090s	0.168	0.204	0.184	0.039	0.127
Sentence-BERT	k-means++	0.071s	0.418	0.438	0.428	0.292	0.074
	aleatória	0.051s	0.408	0.427	0.417	0.298	0.061
	baseada em PCA	0.020s	0.328	0.341	0.334	0.233	0.057
Universal Sentence Encoder	k-means++	0.041s	**0.524**	**0.548**	**0.536**	**0.398**	0.077
	aleatória	0.030s	0.471	0.492	0.481	0.331	0.069
	baseada em PCA	0.017s	0.512	0.526	0.519	0.401	0.062

Tabela 7 – Métricas do experimento 2 - base equalizada, 50 amostras por classe.

A uniformização da base com 50 amostras por classe não apresentou resultou em ganho de desempenho, apresentando até casos de perda de desempenho, quando comparado com cenário de uniformização da base com 500 amostras por classe. O Universal Sentence Encoder com inicialização aleatória teve o melhor desempenho baseado em V-Measure e IRA.

4.3.4 Experimento 3 - Treinamento com apenas 2 classes

Neste experimento, dividido em 3 cenários, a base de dados é limitada a duas classes, a binarização do problema facilita o processo de agrupamento. Visa-se analisar o comportamento de cada modelo em classes de áreas convergentes e áreas divergentes. Sendo o objetivo entender o desempenho de cada modelo em cenários binários.

4.3.4.1 Cenário 1 - Apenas as classes 'Visão Computacional' e Política'

Neste experimento são usadas apenas duas classes: 'Visão Computacional' e 'Política' o qual são assuntos divergentes. Sendo o objetivo identificar se os modelos possuem alto desempenho com assuntos divergentes, o que é o esperado.

Modelo	Inicialização	Tempo	Homo	Compl	V-Measure	IRA	Silhouette
Doc2Vet	k-means++	0.020s	0.743	0.744	0.744	0.832	0.111
	aleatória	0.019s	0.743	0.744	0.744	0.832	0.116
	baseada em PCA	0.006s	0.743	0.744	0.744	0.832	0.112
InferSent	k-means++	0.188s	0.011	0.011	0.011	0.014	**0.354**
	aleatória	0.177s	0.011	0.011	0.011	0.014	0.351
	baseada em PCA	0.110s	0.011	0.011	0.011	0.014	0.345
Sentence-BERT	k-means++	0.047s	0.914	0.914	0.914	0.956	0.095
	aleatória	0.050s	0.908	0.909	0.908	0.953	0.095
	baseada em PCA	0.028s	0.908	0.909	0.908	0.953	0.096
Universal Sentence Encoder	k-means++	0.035s	**0.928**	**0.928**	**0.928**	**0.964**	0.114
	aleatória	0.039s	0.928	0.928	0.928	0.964	0.102
	baseada em PCA	0.020s	0.928	0.928	0.928	0.964	0.110

Tabela 8 – Métricas do experimento 3 - Apenas as classes 'Visão Computacional' e 'Política'.

Com exceção do modelo InferSent todos modelos apresentaram alto desempenho esperada. Os modelo Sentence-BERT e Universal Sentence Encoder possuiram valores notáveis e para todos os modelos o método de inicialização resultou em variações insignificantes. O modelo InferSent apresentou péssimos valores, sendo praticamente incapaz de realizar um agrupamento neste cenário, o coeficiente Silhouette é alto no caso do InferSent, mas com atribuições praticamente aleatórias a densidade dos grupos formados não é uma característica relevante.

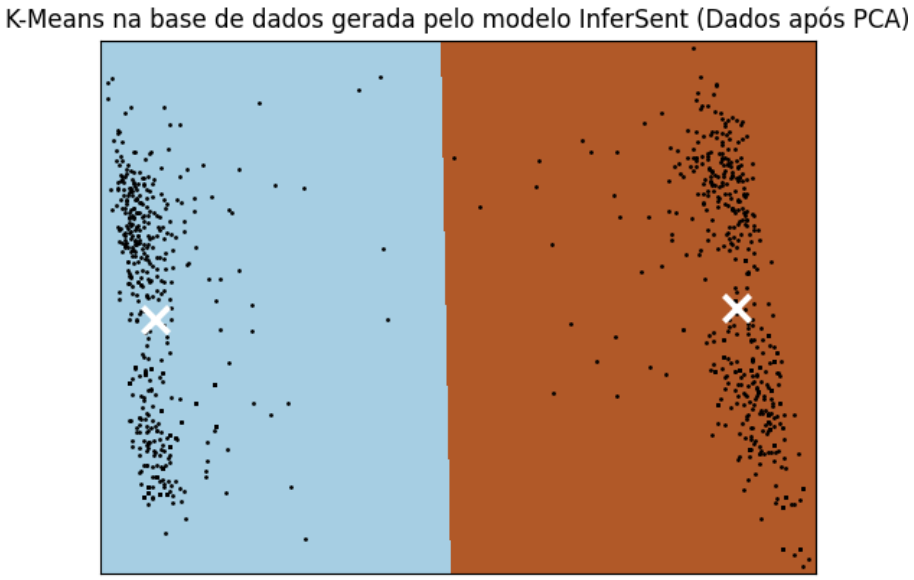


Figura 12 – Experimento 3 - Distribuição dos dados após PCA com algoritmo K-Means usando modelo InferSent.

Na figura 12 consta a distribuição dos embeddings do modelo do InferSent, nota-se que os grupos formados pelo algoritmo K-Means possuem amostras mais concentradas, ou seja, são facilmente classificados, porém a relação que foi encontrada no treinamento não apresenta relação com os rótulos dos embeddings.

4.3.4.2 Cenário 2 - Apenas as classes 'Visão Computacional' e 'Veículo Autônomo'

Neste experimento são usadas apenas duas classes 'Visão Computacional' e Veículo Autônomo' o qual são assuntos convergentes. Sendo o objetivo identificar se os modelos possuem uma queda de desempenho em relação ao cenário anterior o que é esperado e identificar qual modelo desempenha melhor nestas condições.

Modelo	Inicialização	Tempo	Homo	Compl	V-Measure	IRA	Silhouette
Doc2Vet	k-means++	0.021s	0.473	0.475	0.474	0.571	0.096
	aleatória	0.021s	0.486	0.488	0.487	0.586	0.078
	baseada em PCA	0.007s	0.483	0.484	0.483	0.586	0.082
InferSent	k-means++	0.168s	0.004	0.004	0.004	0.005	0.356
	aleatória	0.204s	0.004	0.004	0.004	0.005	0.359
	baseada em PCA	0.102s	0.004	0.004	0.004	0.005	0.351
Sentence-BERT	k-means++	0.054s	0.529	0.535	0.532	0.621	0.069
	aleatória	0.054s	**0.529**	**0.535**	**0.532**	**0.621**	0.076
	baseada em PCA	0.028s	0.526	0.530	0.528	0.617	0.075
Universal Sentence Encoder	k-means++	0.043s	0.472	0.473	0.473	0.577	0.055
	aleatória	0.043s	0.474	0.475	0.474	0.577	0.053
	baseada em PCA	0.018s	0.452	0.454	0.453	0.553	0.050

Tabela 9 – Métricas do experimento 3 - Apenas as classes 'Visão Computacional' e 'Veículo Autônomo'

Todo os modelos apresentaram queda de desempenho comparado ao cenário anterior e o modelo Sentence-BERT com inicialização aleatória teve o melhor desempenho baseado em V-Measure e IRA.

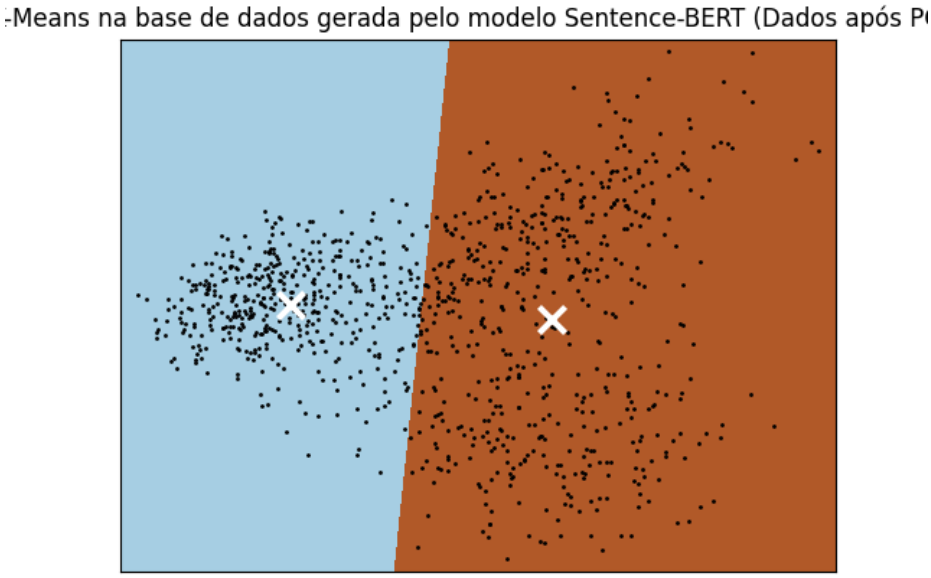


Figura 13 – Experimento 3 - Distribuição dos dados após PCA com algoritmo K-Means usando modelo Sentece-BERT.

Na figura 13 consta a distribuição dos embeddings do modelo do Sentece-BERT,

nota-se que os grupos formados pelo algoritmo K-Means possuem amostras esparsas, porém obteve resultados de 0.532 e 0.621 de V-Measure e IRA em um cenário de difícil agrupamento.

4.3.4.3 Cenário 3 - Apenas as classes 'Política' e 'Ansiedade'

Neste experimento são usadas apenas duas classes 'Política' e 'Ansiedade' o qual são assuntos divergentes e fora do escopo de Engenharia. Sendo o objetivo identificar se os modelos possuem alto desempenho com assuntos divergentes e identificar qual modelo desempenha melhor nestas condições.

Modelo	Inicialização	Tempo	Homo	Compl	V-Measure	IRA	Silhouette
Doc2Vet	k-means++	0.024s	0.393	0.413	0.403	0.438	0.137
	aleatória	0.020s	0.403	0.422	0.412	0.451	0.136
	baseada em PCA	0.007s	0.772	0.773	0.772	0.850	0.095
InferSent	k-means++	0.232s	0.010	0.010	0.010	0.012	0.398
	aleatória	0.182s	0.010	0.010	0.010	0.012	0.383
	baseada em PCA	0.121s	0.010	0.010	0.010	0.012	0.382
Sentence-BERT	k-means++	0.085s	0.128	0.246	0.169	0.054	0.368
	aleatória	0.046s	0.128	0.246	0.169	0.054	0.378
	baseada em PCA	0.027s	0.128	0.246	0.169	0.054	0.407
Universal Sentence Encoder	k-means++	0.036s	0.133	0.250	0.174	0.058	0.228
	aleatória	0.032s	0.133	0.250	0.174	0.058	0.243
	baseada em PCA	0.026s	**0.852**	**0.853**	**0.852**	**0.914**	0.084

Tabela 10 – Métricas do experimento 3 - Apenas as classes 'Política' e 'Ansiedade'

O Universal Sentence Encoder e Doc2Vet apresentaram bons desempenhos com a inicialização baseada em PCA baseado em V-Measure e IRA e o Sentence-BERT e o InferSent possuiram baixo desempenho baseado em V-Measure e IRA. O Universal Sentence Encoder com inicialização baseada em PCA teve o melhor desempenho baseado em V-Measure e IRA. O impacto do PCA nas métricas para os modelos Universal Sentence Encoder e Doc2Vet são notáveis, principalmente o do Universal Sentence Encoder que alcançou IRA melhor em 15,7 vezes.

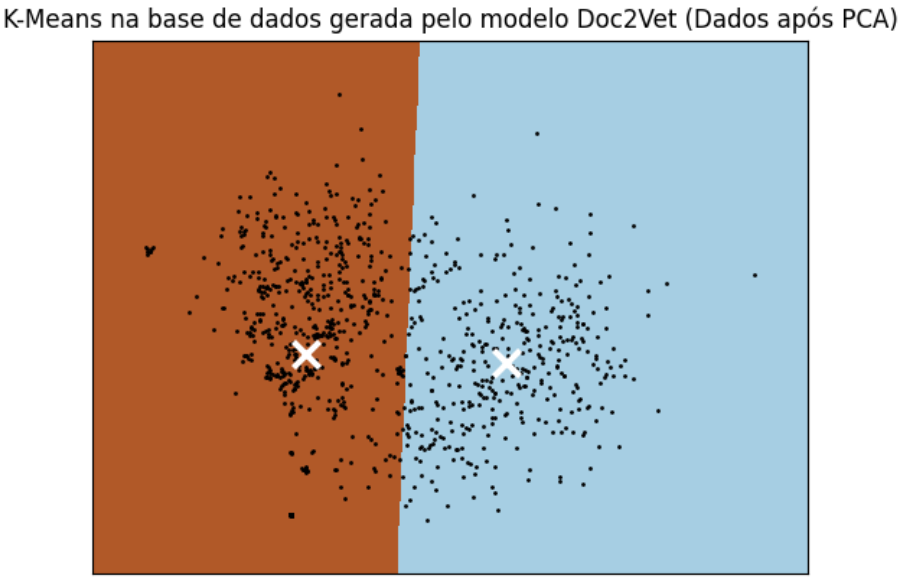


Figura 14 – Experimento 3 - Distribuição dos dados após PCA com algoritmo K-Means usando modelo Doc2Vet.

Na figura 14 consta a distribuição dos embeddings do modelo do Doc2Ve, nota-se que os grupos formados pelo algoritmo K-Means possuem amostras esparsas, porém obteve resultados de 0.772 e 0.850 de V-Measure e IRA.

4.3.5 Considerações sobre os resultados

Observando os experimentos propostos o Universal Sentence Encoder se apresenta como o modelo mais estável, garantido uma qualidade maior de agrupamento que os outros modelos nos diversos cenários. InferSent se apresentou como pior modelo para o problema de agrupamento tendo o pior desempenho em todos os cenários.

É também possível dizer que em alguns cenários os agrupamentos realizados com inicialização baseada em PCA tiveram uma grande melhora nos resultados. Como é o caso do Universal Sentece Encoder no experimento 3 cenário 3.

A aplicação de PCA nos dados para a inicialização do centroides dos grupos do algoritmo K-Means resultou ou em impactos negativos mínimos nas métricas se comparada com a inicialização k-means++ ou aleatória ou em impactos positivos

significativos, concluindo que para esse problema é válido a inicialização dos centroides baseada em PCA. De modo geral foi possível entender o impacto da escolha do modelo de *Sentence Embedding* para cenários prováveis.

5

Considerações finais

Um sistema de análise de desempenho de modelos de *Sentence Embedding* para agrupamento de artigos foi desenvolvido e explanado. O processo de criação de um sistema de WebScrap foi descrito, as fontes de dados que nesse caso foram IEEE, Frontiers e University of Chicago Press as quais serviram para a construção de uma base de dados de 58716 amostras divido em 9 temas com tempo total de 32h de execução.

Para as comparações 4 modelos de *Sentence Embedding* foram selecionados: Doc2Vet, Sentence-BERT, Universal Sentence Encoder e InferSent. Alguns modelos passaram por definição de hiperparâmetros e treinamento enquanto outros já se encontravam pré-treinados e pronto para uso.

Foram definidos 6 cenários diferentes para a análise de desempenho dos modelos. O modelo que apresentou melhor desempenho para o problema de agrupamento de artigos de maneira geral foi o Universal Sentence Encoder, que se apresentou mais estável que os outros modelos em relação as variações de cenários. No experimento 3 cenário 3 o Universal Sentence Encoder alcançou 0.852 e 0.914 na métricas, respectivamtene, V-Measure e IRA. Enquanto o InferSent, modelo cujo foi treinado com a base de dados de artigos coletados, diferentemente do Universal Sentence Encoder, apresentou valores praticamente aleatório em alguns cenários.

Após os experimentos é possível ter um conceito de desempenho dos modelos utilizados para a classificação de artigos. Essa ideia pode ser refinada a fim de se desenvolver um mecanismo de busca de artigos que usa como pivô características semânticas.

Para trabalhos futuros seria conveniente analisar o desempenho dos modelos de *Sentence Embedding* para a similaridade de textos, ou repetir os experimentos realizados com uma base de dados com outro aspecto que não seja resumos de artigos.

REFERÊNCIAS

CER, D. et al. *Universal Sentence Encoder*. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1803.11175>. 20

CHAPAGAIN, A. *Hands-On Web Scraping with Python: Perform advanced scraping operations using various Python libraries and tools such as Selenium, Regex, and others*. [S.l.]: Packt Publishing Ltd, 2019. 14

CLUSTERING Performance Evaluation. 2022. <https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation>. [Accessed 15-Sep-2022]. 21

CONNEAU, A. et al. *Supervised Learning of Universal Sentence Representations from Natural Language Inference Data*. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1705.02364>. 19

EMBEDDINGS. 2022. <https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture>. [Accessed 07-Sep-2022]. 16

FRONTIERS. 2022. Disponível em: <https://www.frontiersin.org/>. 24

GÉRON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019. ISBN 9781492032618. Disponível em: <https://books.google.com.br/books?id=HHetDwAAQBAJ>. 15, 21

HAPKE, H.; HOWARD, C.; LANE, H. *Natural Language Processing in Action: Understanding, analyzing, and generating text with Python*. [S.l.]: Simon and Schuster, 2019. 16

IEEE Xplore. 2022. Disponível em: <https://ieeexplore.ieee.org/Xplore/home.jsp>. 24

LE, Q. V.; MIKOLOV, T. *Distributed Representations of Sentences and Documents*. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1405.4053>. 17

MIKOLOV, T. et al. *Efficient Estimation of Word Representations in Vector Space*. arXiv, 2013. Disponível em: <https://arxiv.org/abs/1301.3781>. 17

REIMERS, N.; GUREVYCH, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv, 2019. Disponível em: <https://arxiv.org/abs/1908.10084>. 19

REIS, J.; HOUSLEY, M. *Fundamentals of Data Engineering: Plan and Build Robust Data Systems*. O'Reilly Media, Incorporated, 2022. ISBN 9781098108304. Disponível em: <https://books.google.com.br/books?id=Z_TFzgEACAAJ>. 15

UNIVERSITY of Chicago Press. 2022. Disponível em: <https://press.uchicago.edu/index.html>. 24

ŽIŽKA, J.; DAŘENA, F.; SVOBODA, A. *Text mining with machine learning: principles and techniques*. [S.l.]: CRC Press, 2019. 13, 14

A

CÓDIGO PARA USO DO DOC2VET

```
import pickle
import random
import numpy as np

# Get preprocess sentences
with open('../tokens', 'rb') as pickle_file:
    tokenized_sent = pickle.load(pickle_file)
random.shuffle(tokenized_sent)

# Prepare data to train
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
tagged_train_data = [TaggedDocument(token[2], [i]) for i, token in
    enumerate(tokenized_sent)]

# Train doc2vec model
model = Doc2Vec(tagged_train_data, vector_size = 30, min_count = 3,
    epochs = 100)
```

B

Código para uso do Sentence-BERT

```
import pickle
import random
import numpy as np

# Get preprocess sentences
with open('../tokens', 'rb') as pickle_file:
   tokenized_sent = pickle.load(pickle_file)
random.shuffle(tokenized_sent)

# Prepare data to train
train_tokens = [' '.join(token[2]) for token in tokenized_sent]

# Install library to use Sentence-BERT
!pip install sentence-transformers

# Import pre trained Sentence-BERT
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('bert-base-nli-mean-tokens')
```

C

CÓDIGO PARA USO DO INFERSENT

```
import pickle
import random
import numpy as np
import torch

from models import InferSent

# Get preprocess sentences
with open('../tokens', 'rb') as pickle_file:
   tokenized_sent = pickle.load(pickle_file)
random.shuffle(tokenized_sent)

# Prepare data to train
train_tokens = [' '.join(token[2]) for token in tokenized_sent]

# Setting hyperparameters
V = 2
MODEL_PATH = 'encoder/infersent%s.pkl' % V
params_model = {'bsize': 64, 'word_emb_dim': 300, 'enc_lstm_dim':
   2048,
            'pool_type': 'max', 'dpout_model': 0.0, 'version': V}
model = InferSent(params_model)
model.load_state_dict(torch.load(MODEL_PATH))
```

```
W2V_PATH = 'GloVe/glove.840B.300d.txt'
model.set_w2v_path(W2V_PATH)

# Training model
model.build_vocab(train_tokens)
```

D

CÓDIGO PARA USO DO UNIVERSAL SENTENCE ENCODER

```
import pickle
import random
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub

# Get preprocess sentences
with open('../tokens', 'rb') as pickle_file:
   tokenized_sent = pickle.load(pickle_file)
random.shuffle(tokenized_sent)

# Prepare data to train
train_tokens = [' '.join(token[2]) for token in tokenized_sent]

# Install library to use Universal Sentence Encoder
!pip3 install --upgrade tensorflow-gpu
!pip3 install tensorflow-hub

# Import pre-trained model
module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
model = hub.load(module_url)
```

E

CÓDIGO PARA USO DO K-MEANS

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

from sklearn.preprocessing import StandardScaler
from time import time
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt


path="../dataframes"
csvs = os.listdir(path)

datasets = []
for csv_name in csvs:
dataset = pd.read_csv(f'{path}/{csv_name}', header=None)
datasets.append(dataset)


themes_selected = ['cybersecurity', 'anxiety']
```

```
restriction = False
equalize = False
max_samples_per_label = 500

def sample_restristion(sample: str):
if sample in themes_selected:
   return True
else:
   return False

def equilize_dataset(labels_count, sample: int):
if labels_count[sample] < max_samples_per_label:
   labels_count[sample] += 1
   return labels_count, True
else:
   return labels_count, False

def pre_process_dataset(dataset):
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values

if restriction:
   indexes = [i for i, sample in enumerate(y) if
      sample_restristion(sample)]
   X = X[indexes]
   y = y[indexes]

X = StandardScaler().fit_transform(X)
y = pd.factorize(y)[0]



if equalize:
   labels_count = [0] * 9
   indexes = []
   for i, sample in enumerate(y):
```

```
        labels_count, check = equilize_dataset(labels_count, sample)
        if check:
            indexes.append(i)

    X = X[indexes]
    y = y[indexes]


data, labels = X, y
(n_samples, n_features), n_digits = data.shape,
    np.unique(labels).size

print(f"# labels: {n_digits}; # samples: {n_samples}; # features
    {n_features}")
return data, labels, n_digits

def bench_k_means(kmeans, name, data, labels):
"""Benchmark to evaluate the KMeans initialization methods.

Parameters
----------
kmeans : KMeans instance
    A :class:`~sklearn.cluster.KMeans` instance with the
        initialization
    already set.
name : str
    Name given to the strategy. It will be used to show the results
        in a
    table.
data : ndarray of shape (n_samples, n_features)
    The data to cluster.
labels : ndarray of shape (n_samples,)
    The labels used to compute the clustering metrics which requires
        some
    supervision.
```

```
"""
t0 = time()
estimator = make_pipeline(StandardScaler(), kmeans).fit(data)
fit_time = time() - t0
results = [fit_time]

# Define the metrics which require only the true labels and estimator
# labels
clustering_metrics = [
   metrics.homogeneity_score,
   metrics.completeness_score,
   metrics.v_measure_score,
   metrics.adjusted_rand_score,
]
results += [m(labels, estimator[-1].labels_) for m in
   clustering_metrics]

# The silhouette score requires the full dataset
results += [
   metrics.silhouette_score(
      data,
      estimator[-1].labels_,
      metric="euclidean",
      sample_size=300,
   )
]

# Show the results
formatter_result = (
   "{:.3f}s\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}"
)
print(formatter_result.format(*results))
return results

def get_results(data, labels, n_digits):
```

```
results = []
print(82 * "_")
print("time\thomo\tcompl\tv-meas\tARI\tsilhouette")

kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4,
    random_state=0)
results.extend(bench_k_means(kmeans=kmeans, name="k-means++",
    data=data, labels=labels))

kmeans = KMeans(init="random", n_clusters=n_digits, n_init=4,
    random_state=0)
results.extend(bench_k_means(kmeans=kmeans, name="random",
    data=data, labels=labels))

pca = PCA(n_components=n_digits).fit(data)
kmeans = KMeans(init=pca.components_, n_clusters=n_digits, n_init=1)
results.extend(bench_k_means(kmeans=kmeans, name="PCA-based",
    data=data, labels=labels))

print(82 * "_")
return results

def plot_pca(data, labels, n_digits, model_name):
reduced_data = PCA(n_components=2).fit_transform(data)
kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4)
kmeans.fit(reduced_data)

# Step size of the mesh. Decrease to increase the quality of the VQ.
h = 0.02 # point in the mesh [x_min, x_max]x[y_min, y_max].

# Plot the decision boundary. For that, we will assign a color to
    each
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:,
    0].max() + 1
```

```
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:,
    1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
    y_max, h))

# Obtain labels for each point in mesh. Use last trained model.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower",
)

plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.", markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="w",
    zorder=10,
)
plt.title(
```

```
    f"K-Means na base de dados gerada pelo modelo {model_name} (Dados
        aps PCA)"
)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.savefig(f'{model_name}.png')
plt.show()

def pipeline(model_name, dataset):
data, labels, n_digits = pre_process_dataset(dataset)
results = get_results(data, labels, n_digits)
plot_pca(data, labels, n_digits, model_name)
print(results)
return results

results = []
models_name = ['Doc2Vet', 'InferSent', 'Sentence-BERT', 'Universal
    Sentence Encoder']
for i, dataset in enumerate(datasets):
model_name = models_name[i]
results.extend(pipeline(model_name, dataset))
```