Avaliação de Modelos de Recuperação de Informação Utilizando Biblioteca de Busca por Similaridade FAISS

José Carlos Ferreira Neto[1]

[1]Programa de Pós-Graduação em Engenharia de Sistemas e Automação – Universidade Federal de Lavras (UFLA)

Abstract. *Information Retrieval (IR) is an area of computer science that explores techniques of representation, storage, organization and access to information. RI models seek to satisfy user needs by retrieving documents that meet their request. This work presents a comparison of IR modeling techniques, combining vector representation generated through TF-IDF and BERT. These strategies are evaluated with the Cystic Fibrosis (CF) collection.*

Resumo. *A Recuperação de Informação (RI) é uma área da computação que explora técnicas de representação, armazenamento, organização e acesso a informação. Os modelos de RI buscam satisfazer as necessidades do usuário, recuperando documentos que atendem a sua solicitação. Este trabalho apresenta uma comparação de técnicas de modelagem de RI, combinando representação vetorial gerada através do TF-IDF e do BERT. Estas estratégias são avaliadas com a coleção Cystic Fibrosis.*

1. Introdução

bla bla bla.

2. Fundamentação Teórica

Os tópicos a seguir abordarão conceitualmente os algoritmos e as técnicas que foram utilizadas no decorrer deste trabalho.

2.1. Representação Vetorial

Várias são as formas de se gerar representações vetoriais de palavras e textos, desde estratégias relacionado a frequência dos termos em um documento até estratégias baseadas em aprendizado de máquina.

O *Term Frequency – Inverse Document Frequency* (TF-IDF) é uma técnica dentro da modelagem vetorial, que consiste em produzir uma representação vetorial de um texto, e esta estratégia é baseada em pesos. O componente TF baseia-se na frequência de cada palavra em cada documento da coleção, e o componente IDF está relacionado com a quantidade de documentos que uma palavra aparece [Croft et al. 2010].

O documento pode ser representado pelo vetor $\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{i,j})$, sendo que o componente tf do termo k_i no documento d_j pode ser obtido por:

$$tf_{i,j} = \begin{cases} 1 + log f_{i,j} & se\ f_{i,j} > 0 \\ 0 & caso\ contrário \end{cases} \quad (1)$$

Onde $f_{i,j}$ representa a frequência de ocorrência de um termo k_i no documento d_j. Já o IDF pode ser obtido por:

$$idf_i = log\frac{N}{n_i} \tag{2}$$

Onde N é o total de documentos em uma coleção e n_i o número de documentos que o termo k_i aparece. Portanto, a combinação dos componentes TF e IDF produz pesos conforme a equação a seguir:

$$w_{i,j} = \begin{cases} (1 + log f_{i,j}) \times log\frac{N}{n_i} & se\ f_{i,j} > 0 \\ 0 & caso\ contrário \end{cases} \tag{3}$$

Outra forma de modelagem é a modelagem semântica. Uma das técnicas desse tipo de modelagem, sendo esta mais recente, é utilizar o modelo BERT (*Bidirectional Encoder Representations from Transformers*) para produzir vetores para as palavras captando o seu sentido no contexto a qual esta está inserida.

O BERT é um modelo pré-treinado desenvolvido pela Google utilizado para tarefas de Processamento de Linguagem Natural (PLN). Os dados de pré-treinamento utilizados extraídos do *BooksCorpus* (800 milhões de palavras) e do *Wikipedia* (2.500 milhões de palavras) [Devlin et al. 2018].

Entretanto, o BERT é restrito a vetorização de palavras, por conta disso, pensando em escalar o mesmo conceito e gerar vetores (*embeddings*) para toda uma sentença, em [Reimers and Gurevych 2019] é proposto uma modificação deste modelo, dando origem ao *sentece*-BERT. Os autores explicam que, o *sentece*-BERT utiliza um codificador que pode converter passagens mais longas de texto em vetores. Este sistema é bastante útil para busca semântica de documentos, uma vez que sentenças semelhantes semanticamente estão próximas umas das outras no espaço vetorial.

2.2. *Grid Search*

Nesta técnica, faz-se uma busca exaustiva pelo melhor valor de um parâmetro específico do algoritmo. Para isso, defini-se uma série de valores com que cada um destes parâmetros podem assumir.

A busca exaustiva se dá pelo treinando de diversos modelos, combinando os valores definidos para cada um dos parâmetros a serem otimizados, e pela avaliação de cada um destes modelos. A melhor configuração, ou seja, a melhor combinação resultará no modelo com as melhores métricas e por consequência este será o modelo selecionado para o treinamento final, conforme demonstra o fluxo da Figura 1.

2.3. Validação Cruzada

A validação cruzada é uma técnica de amostragem que utiliza diferentes partes dos dados para treinar e testar um modelo em diferentes iterações. Utiliza-se esta técnica principalmente para avaliar o modelo sob diferentes amostragens, estimando assim a sua precisão para prever novos valores sobre dados não vistos durante a etapa de treinamento.

Existem várias abordagens para aplicar validação cruzada, são elas: *leave-p-out*, *leave-one-out*, *holdout*, *k-fold* entre outras. Neste trabalho abordaremos apenas a *k-fold*.

O *k-fold* consiste em dividir os dados disponíveis em k partições (*folds*), instanciar k modelos idênticos e treiná-los em $k-1$ partições enquanto os avalia na partição restante. A validação do modelo é feita utilizando a média da(s) métrica(s) obtida(s) nas k iterações [Chollet 2021].

2.4. kNN

A classificação por método de vizinhança não exige uma etapa de treinamento, como normalmente outros classificadores exigem. Isto é, no kNN esta etapa serve para armazenar as instâncias dos dados de treinamento. Para que um novo registro de dado seja classificado, os seus k vizinhos mais próximos são recuperados, formando a vizinhança deste registro. É atribuído a este registro a classe de dados que tem o maior número de representantes dentre a vizinhança, esta última etapa é denominada de votação majoritária [Guo et al. 2003].

Para produzir a vizinhança, necessita-se definir o tipo de cálculo de distância utilizado. A implementação clássica do algoritmo é a utilização da distância euclidiana, conforme a equação a seguir:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{4}$$

Onde, $\mathbf{x}$ e $\mathbf{y}$ são os vetores, cada um representando um registro, x_i e y_i são valores da coordenada i e n o número de coordenadas, isto é, o comprimento do vetor.

Para além da distância euclidiana, pode-se utilizar a similaridade do cosseno, conforme equação a seguir:

$$sim(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}} \tag{5}$$

A votação para a definição da classe a ser atribuída a um novo registro pode ser por voto majoritário simples, isto é, todos os k vizinhos mais próximos possuem o mesmo peso de voto, quanto por voto majoritário ponderado pela distância, isso significa que, vizinhos mais próximos possuem maior influência para definir a classe do novo registro do que vizinhos mais distantes.

2.5. SVM

A ideia principal de uma máquina de vetor de suporte é construir um hiperplano como superfície de decisão de tal forma que a margem de separação entre exemplos de classes diferentes seja máxima [Haykin 2001].

Para conjuntos de dados que não são linearmente separáveis, existem algumas complicações, visto que o objetivo é separar as classes sem violação da margem.

Para contornar este problema, podemos adotar classificação de margem suave, isto é, permitir que pontos de dados violem a margem. Com isto, produz-se um modelo

flexível, capaz de equilibrar a largura da margem para que seja máxima e permitir que alguns pontos violem a margem [Géron 2022].

Outra abordagem é aumentar o número de dimensões do conjunto de dados, de tal forma que, em uma alta dimensão os dados possam ser separados. Porém, ao aumentar a dimensionalidade do problema, o modelo se torna mais lento computacionalmente falando. Podemos mapear estes dados em alta dimensão sem a necessidade de adicionar mais características utilizando o troque do *kernel*. Este mapeamento é feito por meio de uma função, os *kernels* podem ser: linear, polinomial, RBF e sigmoide [Géron 2022].

2.6. PCA

Existem diversos casos onde as instâncias de dados possuem centenas, milhares, ou até mais, características. Diante disso, os algoritmos de aprendizado de máquina tornam-se lentos, podem apresentar dificuldade de encontrar boas soluções para os problemas, ou pior, se tornam intratáveis [Géron 2022].

Para contornar este problema, podemos fazer uso de técnicas de redução de dimensionalidade. Estas técnicas podem ter um custo associado, que é o de perda de informação, em contrapartida, ganhamos principalmente em tempo de treinamento.

O PCA é um dos algoritmos mais populares para redução de dimensionalidade. Ele identifica o hiperplano mais próximo dos dados originais que preserva a maior quantidade de variância e os projeta neste hiperplano. Ao escolher o hiperplano que mantém a maior quantidade de variância possível, é provável que a quantidade de informação perdida seja a mínima possível [Géron 2022].

3. Estratégias Adotadas

Nos tópicos a seguir serão abordadas as estratégias de pré-processamento e modelagem utilizadas neste trabalho.

3.1. Pré-processamento

Para o pré-processamento foi adotado os seguintes procedimentos no corpo dos textos dos documentos da coleção:

- Foram removidos tags HTML e XML;
- Foram substituídas todas as sequências de número por 0;
- Foram removidos todos os múltiplos espaços em branco;
- Remoção de *stopwords* apenas para o TF-IDF.

Para todos os casos citados foi utilizado expressões regulares para encontrar os padrões desejados e fazer as devidas correções.

Para gerar as representações vetoriais dos documentos da coleção, as duas abordagens discutidas na seção 2.1 foram adotadas. Para produzir os vetores baseados no TF-IDF utilizou-se a biblioteca *Scikit-Learn* e para a geração dos *embeddings* via BERT, utilizou-se o modelo pré-treinado *multi-qa-mpnet-base-dot-v1*[1]. Este último foi ajustado para busca semântica utilizando 215 milhões de pares de pergunta e resposta de diversas fontes, este suporta sentenças de até 512 palavras, gerando um *embedding* de 768 dimensões.

[1]Disponível em: https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1

Por conta do tamanho dos vetores produzidos, isto é, o número de dimensões, tanto por TF-IDF quanto pelo *sentenceBERT*, foi necessário aplicar redução de dimensionalidade por PCA para tornar o tempo de treinamento mais rápido.

O número de componentes selecionado está relacionado com a manutenção de 75% da taxa de variância explicada. Sendo que, a taxa de variância explicada é proporção de variância do conjunto de dados que cada componente carrega.

3.2. Modelagem

Para construção dos classificadores, foram utilizados dois algoritmos, kNN e SVM. Isto significa que, iremos comparar 4 classificadores: kNN com TF-IDF, kNN com *sentenceBERT*, SVM com TF-IDF e SVM com *sentenceBERT*. Para definir a melhor configuração para cada um destes classificadores, utilizou-se a técnica de *Grid Search* e validação cruzada, conforme a Figura 1.

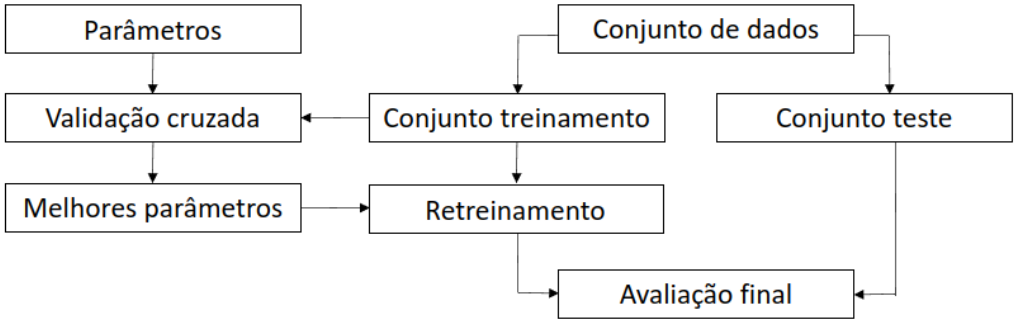

Figura 1. Fluxo Adotado de Treinamento e Avaliação dos Algoritmos

Os parâmetros testados com *Grid Search* para o algoritmo kNN são:

- Valor de k (*n_neighbors*): variando de 3 a 15;
- Tipo de ponderação dos votos (*weights*): *uniform* e *distance*;
- Cálculo de distância (*metric*): *cosine* e *euclidean*.

Os parâmetros testados com *Grid Search* para o algoritmo SVM são:

- Parâmetro de regularização (C): 0.1, 1 e 10;
- Kernel (*kernel*): *linear* e *rbf*;
- Coeficiente para o *kernel rbf* (*gamma*): 1, 0.1, 0.01 e 0.001.

Tanto para o kNN quanto para o SVM, foi utilizado os algoritmos implementados na biblioteca *sklearn*.

Foi utilizado k-*fold* com $k = 5$ e os resultados obtidos foram avaliados de acordo com as métrica F1 micro e F1 macro. Os melhores parâmetros, isto é, os parâmetros que produziram os melhores resultados para cada um dos 4 classificadores foram os selecionados para o retreinamento.

O retreino é a produção final do modelo, desta vez treinado com todos os dados de treino e com os parâmetros selecionados a partir do *Grid Search*. Este modelo por fim é avaliado sobre os dados de teste.

4. Experimentos

Todos as bibliotecas mencionadas na seção 3 estão disponíveis para linguagem Python, e portanto, todos os experimentos foram executados utilizando esta linguagem. Assim sendo, a configuração da máquina utilizada é: processador (CPU) Ryzen 5 3600, 6-CORE, 12-THREADS, velocidade de 3.6GHz, GPU GTX 1660TI com 6Gb de memória e duas memórias (RAM) DDR4 de 16GB e velocidade de 2666MHz.

Todos os códigos utilizados para executar os experimentos mencionados neste trabalho podem ser acessados no GitHub[2].

5. Resultados

Os vetores produzidos pelo *sentenceBERT* possuem 768 dimensões, enquanto que os vetores produzidos pelo TF-IDF possuem 28.659 dimensões. Com a redução de dimensionalidade aplicada, foi possível reduzir os vetores para 87 dimensões para o *sentenceBERT* e 1.515 para o TF-IDF, em ambas a taxa de variância explicada é de 75% em relação ao conjunto de dados original. E para que os treinamentos se tornem mais rápido, foi adotado os vetores reduzidos na sequência deste trabalho.

Na execução do *Grid Search* para o algoritmo kNN foram testadas 52 combinações diferentes de parâmetros, conforme indicado na seção 3.2. Para a escolha da melhor configuração, foi ponderado F1 micro por F1 macro, ou seja, multiplicou-se um pelo outro, sendo a configuração com maior valor a selecionada para treinar o modelo final.

A Tabela 1 apresenta os resultados do *Grid Search* com validação cruzada para o algoritm kNN com a representação vetorial via *sentence*BERT das cinco melhores combinações dentre as 52 possíveis, ordenado pela ponderação entre F1 micro e F1 macro. Em termos de tempo de execução, as diferentes combinações não produziram nenhuma diferença significativa. A configuração que produziu os melhores resultados segundo a ponderação das métricas de F1 foi a combinação com $k = 11$, tipo de ponderação de votação *uniform* e cálculo de distância euclidiana. Dentre as cinco melhores configurações, todas possuem a mesmo tipo de ponderação de votações e cálculo de distância.

Tabela 1. *Grid Search* - Algoritmo kNN com *sentenceBERT*

k - weights - metric	Tempo de Execução (s)	F1 Micro	F1 Macro	F1 Micro x F1 Macro
11 - uniform - euclidean	0.340	0.728	0.318	0.232
5 - uniform - euclidean	0.337	0.706	0.327	0.231
9 - uniform - euclidean	0.340	0.724	0.317	0.230
6 - uniform - euclidean	0.337	0.711	0.320	0.227
8 - uniform - euclidean	0.338	0.720	0.315	0.227

A Tabela 2 apresenta os resultado do *Grid Search* com validação cruzada para o algoritmo kNN com a representação vetorial via TF-IDF das cinco melhores combinações dentre as 52 possíveis, ordenado pela ponderação entre F1 micro e F1 macro. Semelhante ao que foi apresentado nos resultados anteriores, o tempo de execução para as diferentes

[2]Códigos disponíveis em: https://github.com/jcfneto/document-classification

combinações não produziram diferença significativa. A configuração que produziu os melhores resultados segundo a ponderação das métricas de F1 foi a combinação com $k = 14$, tipo de ponderação de votação *uniform* e cálculo da similaridade do cosseno. Sendo que, dentre as cinco melhores configurações, todas possuem os mesmos parâmetros exceto pelo número de k.

Tabela 2. *Grid Search* - Algoritmo kNN com TF-IDF

k - weights - metric	Tempo de Execução	F1 Micro	F1 Macro	F1 Micro x F1 Macro
14 - uniform - cosine	0.564	0.702	0.302	0.212
12 - uniform - cosine	0.562	0.695	0.304	0.211
9 - uniform - cosine	0.558	0.691	0.304	0.210
13 - uniform - cosine	0.561	0.699	0.299	0.209
15 - uniform - cosine	0.571	0.697	0.299	0.208

Comparativamente, os resultados com *sentenceBERT* apresentaram tempo de execução menor, e isso é justificado pelo comprimento dos vetores, neste caso o vetor do *sentenceBERT* é menor, gerando um custo computacional menor e consequentemente uma execução mais rápida.

Na execução do *Grid Search* para o algoritmo SVM foram testadas 24 combinações diferentes de parâmetros, conforme indicado na seção 3.2. A escolha da melhor configuração segue o mesmo critério mencionada anteriormente para o kNN.

A Tabela 3 apresenta os resultados para o algoritmo SVM com *sentenceBERT* para as cinco melhores configurações dentre as 24 possíveis. Aumentar valores de C treinamentos mais lentos, esse aumento foi ainda mais relevante quando se utilizou *kernel* rbf e valores maires para *gamma*. Isso é percebido ao se comparar as seguintes duas configurações, por exemplo:

- $C = 10$ e *kernel* linear o tempo de execução médio foi de 4.76s;
- $C = 10$ e *kernel* rbf o tempo de execução médio foi:
  - Para $gamma = 0.001$: 3.89s;
  - Para $gamma = 0.01$: 3.86s;
  - Para $gamma = 0.1$: 15s;
  - Para $gamma = 1$: 53.05s;

Tabela 3. *Grid Search* - Algoritmo SVM com *sentenceBERT*

C - gamma - kernel	Tempo de Execução	F1 Micro	F1 Macro	F1 Micro x F1 Macro
0.1 - x - linear	2.752	0.752	0.366	0.275
10 - 0.01 - rbf	3.857	0.729	0.356	0.260
1 - x - linear	2.911	0.712	0.349	0.249
10 - x - linear	4.625	0.698	0.347	0.242
10 - 0.001 - rbf	3.887	0.759	0.289	0.219

A melhor configuração é com $C = 0.1$ e *kernel* linear. O fator *gamma* não está presente para *kernel* linear. Comparando com os resultados obtidos pelas diferentes combinações de parâmetros, o SVM produzem resultados menos equilibrados comparando entre si em relação aos resultados do kNN.

A Tabela 4 apresenta os resultados para o algoritmo SVM com TF-IDF para as cinco melhores configurações dentre as 24 possíveis. Assim como foi observado com o algoritmo kNN, o tempo de execução com os vetores produzidos via TF-IDF é maior que os produzidos via *sentenceBERT*. A configuração com melhor resultado foi com $C = 10$ e *kernel* linear.

Tabela 4. *Grid Search* - Algoritmo SVM com TF-IDF

C - gamma - kernel	Tempo de Execução	F1 Micro	F1 Macro	F1 Micro x F1 Macro
10 - x - linear	56.787	0.699	0.341	0.239
1 - x - linear	60.174	0.736	0.324	0.238
10 - 0.1 - rbf	69.842	0.717	0.324	0.232
10 - 1 - rbf	105.103	0.695	0.326	0.227
1 - 1 - rbf	105.339	0.723	0.314	0.227

A Figura 2 apresenta os resultados de F1 micro e F1 macro médios de todas as configurações testadas. Nota-se que, tanto para F1 micro quanto para F1 macro, o algoritmo SVM apresentou valores máximos maiores se comparado ao kNN. Entretanto, nota-se também que, existem algumas configurações do algoritmo SVM que apresentaram resultados bem inferiores ou até discrepantes das demais configurações do SVM, a maior parte delas possuem em comum o *kernel* rbf.

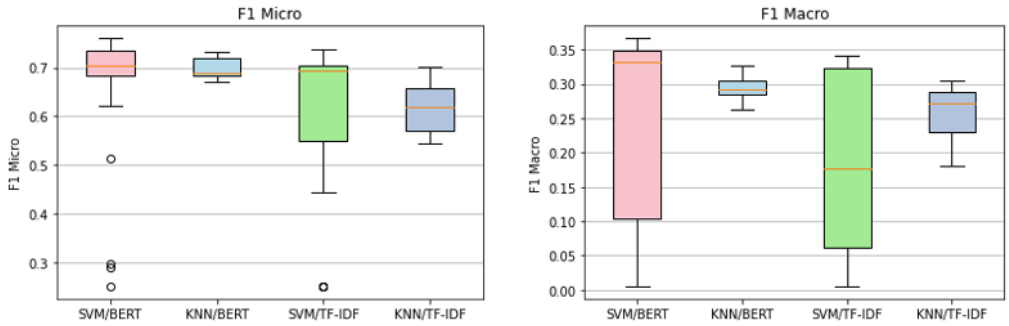

Figura 2. F1 Micro e F1 Macro

Fez-se uso do teste-T unilateral para as médias de duas amostras independentes para identificar se há diferença estatística para os experimentos executados. Com base no que foi observado para os resultados obtidos, as hipóteses alternativas levantadas são:

1. H1: média F1 micro do SVM/BERT $>$ média F1 micro KNN/BERT;
2. H1: média F1 micro do SVM/TF-IDF $>$ média F1 micro KNN/TF-IDF;
3. H1: média F1 macro do SVM/BERT $<$ média F1 macro KNN/BERT;
4. H1: média F1 macro do SVM/TF-IDF $<$ média F1 macro KNN/TF-IDF;
5. H1: média F1 micro do SVM/BERT $>$ média F1 micro SVM/TF-IDF;
6. H1: média F1 micro do KNN/BERT $>$ média F1 micro KNN/TF-IDF;
7. H1: média F1 macro do SVM/BERT $>$ média F1 macro SVM/TF-IDF;
8. H1: média F1 macro do KNN/BERT $>$ média F1 macro KNN/TF-IDF.

Ressaltando que, para cada hipótese alternativa, existe uma hipótese nula que é exatamente oposta a hipótese alternativa postulada. Considerando um intervalo de

confiança de 95%, para todos os testes, exceto para o item 8, foi observado um valor-P < 0.05. Isso significa que para as hipóteses do item 1 à 7, podemos rejeitar a hipótese nula.

Tendo selecionado as configurações para cada um dos classificadores, estes foram treinados com todos os dados de treinamento e avaliados sobre os dados de treino.

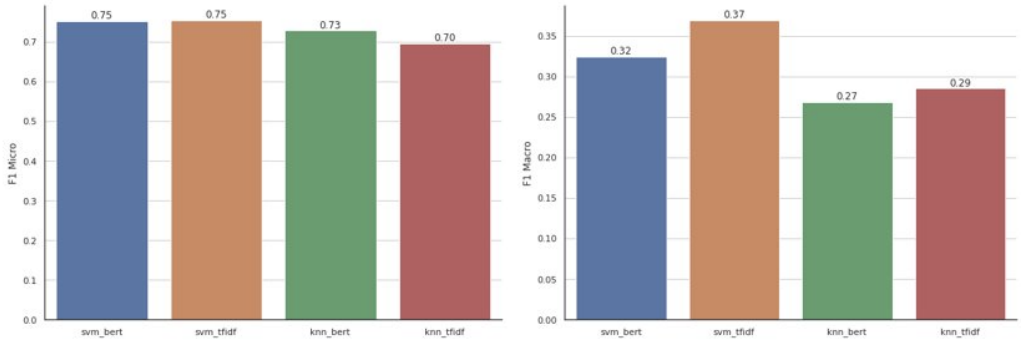

Figura 3. F1 Micro e F1 Macro dos Modelos Ajustados

A Figura 3 apresenta os resultados dos modelos treinados com todos os dados de treino e com os melhores parâmetros selecionados. Em termos de F1 micro, os dois classificadores com algoritmo SVM possuem o mesmo resultado enquanto que o classificador com kNN com vetor *sentenceBERT* é superior ao kNN com TF-IDF. Ao olhar para F1 macro, os classificadores com vetor TF-IDF apresentam resultados superiores.

6. Conclusão

Referências

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Croft, W. B., Metzler, D., and Strohman, T. (2010). *Search engines: Information retrieval in practice*, volume 520. Addison-Wesley Reading.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. "O'Reilly Media, Inc.".

Guo, G., Wang, H., Bell, D., Bi, Y., and Greer, K. (2003). Knn model-based approach in classification. In *OTM Confederated International Conferences"On the Move to Meaningful Internet Systems*
, pages 986–996. Springer.

Haykin, S. (2001). *Redes neurais: princípios e prática*. Bookman Editora.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.