Disciplina Integradora I - Recuperação de informação

Pedro de Castro e Igor Dourado

9 de julho de 2021

Resumo

A internet a cada dia gera um volume extremamente grande de dados. Sem o uso adequado de algoritmos de buscas de informações seria extremamente frustrante, senão impossível, realizar a construção de ferramentas que realizem estas funções. O intuito deste artigo é verificar possíveis soluções para o problema em que precisamos buscar trabalhos de conclusão a partir de suas palavras-chave. Serão analisados três algoritmos, começando do mais elementar, até o mais usual que utiliza uma estrutura de dados própria chamada "índices invertidos". Serão apresentados tanto o pseudocódigo da implementação quanto o código na linguagem de programação Python para resolver o problema.

Introdução

Durante anos bibliotecas criaram formas de indexar e classificar livros, textos e diversos artigos para assim existir uma forma de recuperação e fácil acesso a tais dados. Com a criação do computador e por sequência da Internet, o número de arquivos e dados cresceram de forma exponencial gerando assim a necessidade de existir novas formas de recuperação de dados, que agora estão armazenados de formas digital. Algoritmos de buscas de informação, do inglês *Information Retrieval* (IR), são formas de implementar sistemas de buscas a partir de palavras-chave e entregar como solução o dado requirido para o usuário que fez a requisição. Algoritmos de IR, na sua grande parte, lidam com dados não estruturados, que são segundo [1] dados em que

existam indicações limitadas do seu tipo. Como um exemplo de dados não estruturados são documentos de texto que contenham informações úteis dentro dele. Neste artigo como iremos lidar com uma coleção de trabalhos de conclusão os mesmo podem ser considerados como dados não estruturados portanto, será focado apenas em soluções de IR que lidem com dados não estruturados.

Sistemas em IR podem ser caracterizados em diferentes níveis: por tipos de usuários, tipos de data e pelo tipo de informação que necessita. Diversos tipos de sistemas em IR são desenvolvidos para lidarem com a combinação destas três características descritas como:

- **Tipos de usuários**. O usuário pode ser um especialista (como por exemplo, um curador ou um bibliotecário), o qual busca por informações específicas com palavras-chave relevantes para aquela tarefa, ou uma pessoa com informações genéricas e palavras-chave genéricas que deseja resolver sua tarefa (estudantes buscando informações sobre algum tópico, pesquisadores, etc).

- **Tipos de dado**. Sistemas de busca podem ser otimizados para buscar tipos específicos de dados. Como exemplo neste artigo será necessário buscar trabalhos de conclusão a partir de suas palavras-chave, o qual pode ser feito por força bruta em cima do conteúdo de cada documento ou, de forma mais otimizada, o programa simplesmente procura dentro da seção "palavras-chave"e confere se existe alguma ocorrência de busca. Sistemas em IR que utilizam da otimização de busca em documentos específicos também são chamados de *domain-specific* ou *vertical IR systems*.

- **Tipos de informação**. Mais ligada ao contexto de buscas na Internet, informações dos usuários podem ser definidas como navegacional, informacional ou transacional [1] sendo: Navegacional se referindo a uma informação em particular que o usuário necessita rapidamente. Informacional sendo uma busca relacionada a algum tópico em específico. Transacional se referindo a uma busca que levem a sites que gerem mais informações para o usuário buscar mais sobre o assunto (redes sociais, lojas e-commerce, etc).

Dentro do contexto da ciência da computação as disciplinas de banco de dados e IR são bastante interligadas. Banco de dados lidam com recuperação

de dados estruturados, com uma linguagem bem definida tal qual representa formas de buscas otimizadas conhecida como SQL (*Structured Query Language*). Já sistemas IR, lidam com dados não estruturados como comentado anteriormente, o que permite existir uma linguagem sem uma ordem bem definida a qual não irá garantir uma consistência em suas buscas. Abaixo será mostrado as principais diferenças entre sistemas de banco de dados e sistemas IR segundo [1]

Sistemas de banco de dados:

- Dados estruturados.
- Controlados por esquemas.
- Modelo relacional (orientado a objetos, hierárquico, não relacional, etc) predominante.
- Modelo de consulta estruturada.
- Operações ricas com metadados.
- Consulta retorna dados.
- Resultados são baseados em combinações exatas (está sempre correto)

Sistemas IR:

- Dados não estruturados.
- Sem esquema fixo, vários modelos de dados (e.g. espaço de vetores).
- Não existe uma linguagem específica de busca.
- Pode ser utilizado várias noções de estatísticas para complementar buscas.
- Busca irá devolver uma lista ou um ponteiro específico para o documento
- Resultados são baseados em aproximações correspondentes. Podendo ser também utilizado medidas de "rank"para buscas.

Problema computacional

Na seção anterior foi definido IR como o processo de busca de documentos não estruturados dentro de uma coleção, tal qual melhor combine com a palavra-chave buscada pelo usuário. Uma palavra-chave pode ser usada como

uma busca específica de algum conteúdo (termos como "Sistemas operacionais","Linguagens de programação"podem ser usados para referirem a buscas relacionadas a Ciência da Computação). Segundo [1] existem dois modos de interação sobre sistemas IR, são eles: *Retrieval* (de recuperação) e *Browsing* (de navegação), os dois são similares no que tange ao seus objetivos, porém o de recuperação concentra seus esforços em extrair informações relevantes de um repositórios de documentos a partir de uma busca de palavra-chave. Já o de navegação se contenta em mostrar documentos que sejam similares ou relacionados baseados em relevância de buscas do usuário (bastante utilizado por navegadores de internet onde páginas da Web levam a diversos outros lugares).

Existem duas abordagens principais para IR: a estatística e a semântica. Em uma abordagem estatística, os documentos são analisados e desmembrados em trechos de texto (palavras, frases, palavras-chave), e cada palavra ou frase é contada sua frequência, pesada e medida por sua relevância ou importância. Essas palavras agora serão comparadas com os termos de consulta em grau de combinação em potencial para assim produzir uma lista pontuada de documentos resultantes que contêm as palavras buscadas. Existem três técnicas estatísticas, a booleana, espaço de vetores e probabilística. Por outro lado a abordagem semântica em IR usam técnicas de recuperação baseadas em conhecimento sintático, léxico, sentencial, baseado em discurso e pragmático do entendimento do conhecimento.

Este artigo tem o enfoque em sistemas de IR com modos de interação em Retrieval (de recuperação), iremos utilizar também uma abordagem estatística. Em um primeiro momento iremos utilizar as palavras-chave já contidas nos trabalhos de conclusão, favorecendo-nos de conhecer nosso problema, e em um segundo momento iremos generalizar a construção das palavras-chave utilizando métodos estatísticos que serão apresentados posteriormente.

O problema a ser resolvido pode ser exemplificado como: "Dado uma coleção de trabalhos de conclusão e uma palavra-chave do usuário, o software responsável pela recuperação de informação deve devolver todos os documentos relacionados aquela palavra-chave, ou documentos que contenham enfoques semelhantes ao procurado". A imagem abaixo representa uma possível resposta que o software irá devolver para o usuário que fará uma busca a partir de uma palavra-chave, também chamada de "query de busca". Inicialmente o software irá receber um pedido de busca de um usuário com uma palavra-chave, o software irá procurar em todos seus documentos utilizando

força bruta, se for optado pela primeira opção de resolução, ou irá gerar a estrutura própria de pesquisa (índices invertidos) e por fim devolverá uma lista com documentos como resultado.

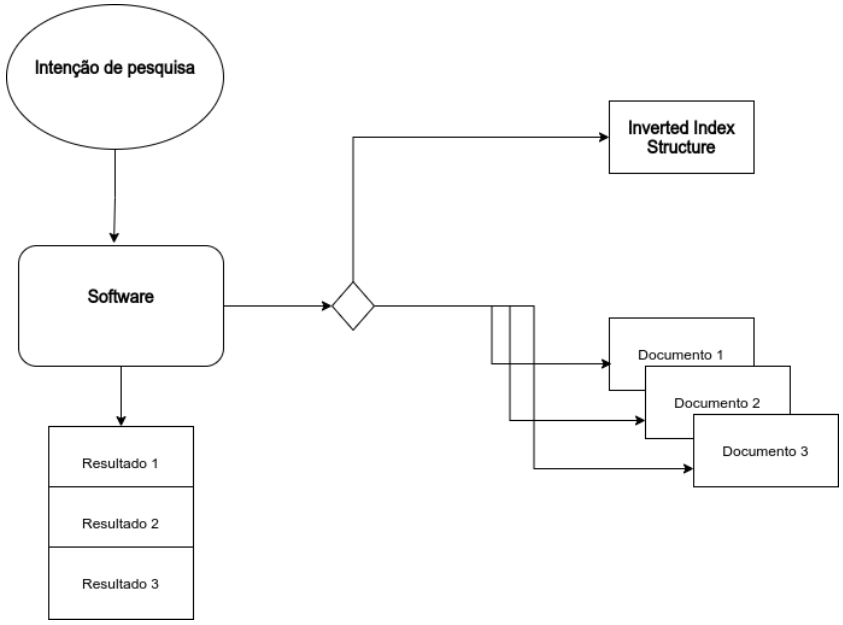

Figura 1: Exemplo de execução do problema.

Aplicabilidade do problema

Como principal referência de aplicabilidade nós temos todas as ferramentas de buscas de documentos online. Podem elas serem encontradas em bibliotecas online, ferramentas mais conhecidas como Google Scholar[1], periódicos da CAPES[2], bibliotecas de faculdades como da PUCRS[3] e grandes aplicabilidades também em navegadores da Internet.

[1]https://scholar.google.com.br/
[2]https://www-periodicos-capes-gov-br.ezl.periodicos.capes.gov.br/
[3]https://biblioteca.pucrs.br/

Definição de estruturas

A forma mais simples de procurar ocorrências de termos de consulta em coleção de documentos pode ser realizada ao varrer o texto sequencialmente, porém este tipo de consulta somente é apropriado quando temos uma coleção pequena de textos. Algoritmos de IR tendem então a criar uma estrutura própria para a otimização de suas buscas, esta estrutura é chamada de "índices invertidos". Uma estrutura de dados índices invertidos segundo [1] compreende informações do vocabulário do texto e seu referente ponteiro (ou qualquer forma de fácil identificação) para o documento (Chamamos de vocabulário um conjunto de termos distintos referentes ao documento encontrado), cada termo do vocabulário pode conter também ponteiros para o documento específico, a frequência da ocorrência do vocabulário, e o quanto é necessário para nos deslocarmos até chegarmos a ocorrência (em termos de quantas palavras até chegarmos a ocorrência). As diferentes etapas envolvidas na construção dos índices invertidos podem ser resumidas da seguinte forma:

1. Divida os documentos em termos de vocabulário ao criar "tokens", limpar, remover *Stopwords*[4] definir a raiz das palavras (Mormefa nuclear).

2. Reunir estatísticas de documento e armazená-las em uma tabela de pesquisa de documento.

3. Inverter o fluxo documento-termo para o um fluxo termo-documento, junto com as estatísticas obtidas anteriormente (criação dos índices invertidos).

A imagem abaixo, retirada de [1] exemplifica uma possível construção de uma estrutura a partir de três documento, a estrutura gerada pode ser vista a esquerda onde ela contém informações sobre a identificação do documento, o termo (vocabulário, palavra-chave) e a posição onde foi encontrada a ocorrência do vocabulário.

[4]Stopwords são palavras importantes na criação de sentenças porém não agregam nenhum valor semântico para a mesma, como exemplo *de, o, e, para, n ao, uma, sim*

ID	Term	Document: position
1.	example	1:2, 1:5
2.	inverted	1:8, 2:1
3.	index	1:9, 2:2, 3:3
4.	market	3:2, 3:13

Figura 2: Exemplo de uma estrutura índices invertidos.

Nas próximas seções serão analisadas formas de resolvermos o problema de recuperação de documentos utilizando desde a forma mais simples de busca (força bruta), até uma das mais utilizadas que se apropria de estruturas de dados próprias para otimizar a busca em documentos de texto, chamada de índices invertidos (utilizando as palavras-chave já contidas nos trabalhos de conclusão), e em um momento final será utilizada uma forma genérica de extração de palavras-chave de qualquer documento de texto utilizando métodos de abordagem estatística e algoritmos de aprendizado de máquina.

Primeira solução

Para a primeira possível solução do problema de buscar trabalho de conclusão, iremos usar a forma mais simples de buscar sequencialmente palavras-chave dentro de um documento, a força bruta. Algoritmos de força bruta são

conhecidos como algoritmos triviais, porém geral no que consiste verificar todas as possibilidades de resposta. A principal desvantagem de algoritmos desta forma são as chamadas explosões combinatórias, que acontecem quando existe um número elevado de candidatos para a solução (o que em problemas reais geralmente será verdade). Abaixo podemos ver um pseudocódigo baseado em uma implementação de força bruta de [3] para resolver o problema de buscar uma palavra-chave dentro de uma coleção de documentos.

Algorithm 1 Brute-Force-Search

```
1: List = []
2: for i = 1, 2 ..., N do
3:     for text in documents.list[i] do
4:         if Query == text then
5:             List.append(documents.list[i].id)
6:         end if
7:     end for
8: end for
9: return List
```

O algoritmo começa quando existe um número N que será a quantidade de documentos existentes na coleção, e para cada iteração existe um "objeto"chamado "documents"que contém informações sobre a lista de documentos existentes, o "id"de cada documento da lista, seu tamanho total, e o conteúdo textual de cada documento. A cada iteração é comparado a "Query"feita pelo usuário com o objeto "documents", se existir uma igualdade é inserida em outro objeto chamado "List", tal objeto será a resposta final para nosso algoritmo o mesmo irá conter todos os documentos que irão ter a ocorrência da palavra-chave feita pelo usuário.

Análise da solução

Nosso algoritmo não é viável para uma busca em uma coleção grande de documentos visto que, é muito provável que exista uma possível explosão combinatória. Uma análise do pior caso de execução do algoritmo se comporta em um tempo $\mathcal{O}(N * m * s)$, sendo "N"a quantidade de documentos existentes na coleção e "m"o tamanho do texto em cada documento e "s"sendo o tamanho de cada palavra presente no texto.

Existem algumas possíveis otimizações para melhorar o número de comparações possíveis dentro do algoritmo de força bruta, umas delas como dito anteriormente poderia ser ao invés de trabalharmos com todo os dados textuais brutos poderíamos utilizar de uma etapa anterior sendo ela um pré-processamento dos dados e removendo então todas as stopwords antes de iniciarmos a busca.

Problemas no algoritmo

Como alguns problemas dentro deste algoritmo é possível citar alguns deles: O primeiro como visto acima e dito anteriormente é o de análise de tempo no pior caso, sua execução para um número grande de documentos se torna inviável já que lidamos com algoritmos próximos de ordens quadráticas. Como um outro problema podemos relatar a possibilidade de dar falsos negativos, ou seja, uma busca não encontrar em algum texto uma palavra-chave referente ao assunto, porém o documento conter sim um assunto referente aquela busca.

Sugestão de otimização

Em relação ao primeiro problema, o caso da análise do tempo de execução do pior caso, será necessário reformular o algoritmo utilizando novas ideias de armazenamento e buscas em estruturas apropriadas para o problema. Já o problema de nossa solução nos gerar falsos negativos podem existir algumas melhoras em que nosso algoritmo, como parte do pré processamento, gera uma lista de sinônimos para a palavra-chave de busca feita pelo usuário e assim, além de buscarmos pela palavra chave inicial, iremos também buscar por todos os seus sinônimos. Visto que esta primeira solução foi utilizada para nos dar um primeiro entendimento do assunto, não iremos seguir com implementações de otimização e/ou análise de métricas de desempenho porém nas soluções subsequentes irão ser apresentas suas devidas análises.

Segunda solução

Nossa primeira solução nos gera um problema de otimização, o que nos leva a recorrer para uma análise maior de nossas possibilidades. Uma delas, como dito anteriormente, é a utilização de "índices invertidos"uma estrutura

de dados muito usada para problemas ligados a IR, que nos permite uma flexibilidade e um fácil retorno e acesso a dados contidos nos documentos previamente processados. Esta estrutura é baseada em vocabulários como índices e documentos como valores referentes aos mesmos, uma forma de entendermos a estrutura de dados em termos de estruturas computacionais conhecidas, seria um "dicionário"de palavras-chave como "chaves"e ponteiros, ou o próprio nome do documento, como "valores".

Para começarmos nossa segunda solução iremos precisar de etapas inicias que iremos chamar de pré-processamento. As etapas do pré-processamento, segundo [1], podem ser listadas como:

1. Extração das palavras-chave dentro de toda a coleção de trabalhos de conclusão.

2. Remoção das pontuações e stopwords.

3. Transformar as palavras-chave em mormefas nucleares.

4. Gerar uma lista de "tokens"com as palavras-chave.

Para o primeiro item da lista, como iremos trabalhar somente em um contexto aonde nossos documentos serão trabalhos de conclusão, uma forma de resolver esta tarefa utilizando um menor esforço computacional possível será procurar diretamente dentro da seção onde o autor explícita quais serão as palavras-chave referente ao seu trabalho de conclusão. Tendo em mãos uma lista pronta de palavras-chave recém extraídas do documento, iremos partir para as próximas etapas. A remoção de pontuações e stopwords podem ser feitas por um simples algoritmo que percorre toda a lista e irá retirar o que não será utilizando futuramente, já a transformação das palavras em morfemas nucleares (raízes das palavras) será necessário a utilização de um dicionário tal qual tenha como chaves as possíveis palavras geradas pela raiz, e como valores suas palavras "filhas".

Para resolver estes problemas de pré-processamento foi utilizado bibliotecas padrão em processamento de linguagens naturais como a *NLTK*[5], tais bibliotecas nos fornecem tanto métodos para removermos as stopwords, quanto um método também para gerarmos as raízes das palavras-chave. Outras bibliotecas também serão utilizadas e consequentemente também serão mostradas no decorrer das soluções.

[5]https://www.nltk.org/

Implementações

Para demonstrarmos as implementações feitas, tanto para a segunda implementação quanto para a terceira, iremos utilizar a linguagem de programação Python e dentro desta linguagem também será utilizada bibliotecas como: NLTK, pdfplumber[6], Scikit-learn[7] e módulos como Pickle[8] para salvar serialização de objetos.

Implementação em Python

Para nossa implementação em Python foram desenvolvidas duas classes diferentes: *InvertedIndex* que representa nosso objeto central para solução do problema e *Documents* que irá representar o documento em sua forma textual, tanto ele de forma bruta, quanto suas palavras-chave já extraídas e pré-processadas.

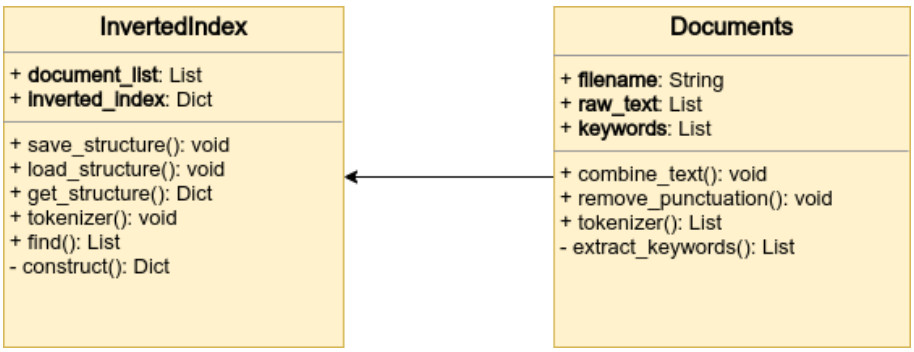

Figura 3: UML de implementação das classes.

Como o foco de nossa implementação são as partes principais da resolução do problema, iremos focar em apenas três métodos dentre as classes existentes. O primeiro será o *extract-keywords* da classe "documents", este método contempla a parte do algoritmo que faz a extração das palavras-chave do

[6]https://github.com/jsvine/pdfplumber
[7]https://scikit-learn.org/stable/
[8]https://docs.python.org/3/library/pickle.html

trabalho de conclusão, e após a extração das palavras-chave o mesmo parte para as etapas de retirada de pontuações, stopwords e lematização (processo de transformar as palavras em suas raízes).

```
def extract_keywords(self):
    with pdfplumber.open(self.filename) as pdf:
        for pdf_page in pdf.pages:
            single_page = pdf_page.extract_text()
            if single_page:
                self.raw_text.append(single_page)

        for i, text in enumerate(self.raw_text):
            position = self.raw_text[i]
                .find("Palavras-chave")
            if position > 0:
                keywords = self.raw_text[i]
                    .split("Palavras-chave")[1]
                break

            position = self.raw_text[i]
                .find("PALAVRAS-CHAVE")

            if position > 0:
                keywords = self.raw_text[i]
                    .split("PALAVRAS-CHAVE")[1]
                break

    keywords = keywords.replace("\n", "")
    sw = stopwords.words("portuguese")
    keywords = self.combine_text(keywords).lstrip()
    keywords = self.remove_punctuation(keywords)
    keywords = keywords.lstrip()
    keywords = self.tokenizer(keywords)
    keywords_clean = list()

    for k in keywords:
```

```
        if k not in sw:
            keywords_clean.append(k)

    return keywords_clean
```

O algoritmo começa abrindo um arquivo "pdf"e extraindo o conteúdo de cada página, neste primeiro passo é utilizado o auxilio de bibliotecas que realizam estas funções como a "pdfplumber"com os métodos *extract_text*. Após a extração do texto é utilizado um loop para procurar a ocorrência da String "Palavras-chave:"ou "PALAVRAS-CHAVE", nos retornando assim todas as palavras-chave referentes ao documento e por fim terminamos pré-processando todas nossos dados com a ajuda de bibliotecas de processamento de linguagem natural.

O segundo método que iremos mostrar será o de construção dos índices invertidos, dentro da classe "invertedIndex". Nosso objeto "inverted index"contém a lista de todas as palavras-chave dos documentos e seus respectivos nomes de arquivos, então basta o método criar um dicionário de palavras-chave como "chaves"e uma lista de nomes de arquivos como valores.

```
def construct(self):
    data_combined = {}
    for document in self.documents_list:
        for keyword in document.keywords:
            if keyword not in data_combined.keys():
                data_combined[keyword] = list()
            data_combined[keyword]
                .append(document.filename)

    return data_combined
```

O algoritmo percorre toda a lista de documentos fornecida e por cada atributo "keywords"pertencentes a cada documento. Se a "keyword"do documento não existir dentro do dicionário anteriormente é criado uma lista

referente aquela posição do dicionário, se a "keyword"já tiver uma posição dentro do dicionário é apenas inserido dentro da lista um novo nome de arquivo.

Por fim, o terceiro método que iremos falar sobre será o *find* dentro da classe "invertedIndex". Este método recebe como argumentos uma palavra-chave fornecida pelo usuário e irá buscar dentro da estrutura já criada por ocorrências da mesma, se for encontrado a palavra-chave dentro da estrutura, o método irá devolver o conjunto referente aos documentos.

```
def find(self, query):
    query = self.tokenizer(query)
    query = "".join(query)
    try:
        response = self.inverted_index[query]
        return response
    except KeyError as e:
        # Handle error
```

O terceiro algoritmo começa criando um "token"da palavra-chave que o usuário esta tentando procurar, é necessário simplesmente procurar dentro da nossa estrutura se a chave existe, é importante lembrar que se a chave não existir dentro da estrutura o erro deverá ser tratado.

Por último existem outros dois métodos que servem apenas para a criação e armazenamento da estrutura, esta forma de armazenamento nos garante que não será necessário utilizar o método "construct"toda vez que o usuário tiver a intenção de pesquisa e também nos garante uma otimização de busca já que a estrutura sempre estará pronta para uso, se ela existir. Para a serialização do objeto foi utilizado a biblioteca "pickle"do Python nos métodos "save_structure"e "load_structure"na classe "invertedIndex".

Análise da solução

A segunda solução nos garante uma busca otimizada no momento em que existe a estrutura gerada para cada documento presente na coleção de documentos. Existe uma diferença drástica em relação ao primeiro algoritmo de

força bruta, o mesmo garantia no pior caso uma busca de $\mathcal{O}(N * m * s)$ uma complexidade beirando ser quadrática, enquanto uma busca em nossa estrutura de dados nos garante uma complexidade linear no pior caso de $\mathcal{O}(N)$ sendo "N"o número de elementos pertencentes ao conjunto das "chaves"do dicionário em Python[9].

Problemas no algoritmo

A segunda solução nos fornece uma ideia geral de como a estrutura índices invertidos pode ser utilizada para resolvermos o problema de buscas em uma coleção de documentos. Dois problemas podem ser citados nesta solução, como um deles sendo na parte do pré processamento dos dados. Como é utilizado apenas uma biblioteca de extração de dados em um arquivo "pdf"ocorre de que em alguns arquivos existirem palavras não codificadas corretamente e/ou pontuações que não são reconhecidas pelo algoritmo, o que acaba gerando problemas futuros como a perda de algumas palavras-chave e também poderá gerar de falsos negativos. Como um segundo problema podemos visualiza-lo no gráfico abaixo que demonstra os tempos de execução do método "extract_keyword"da classe "documents".

[9]Estas informações sobre a complexidade dos tempos foram retiradas em https://wiki.python.org/moin/TimeComplexity.

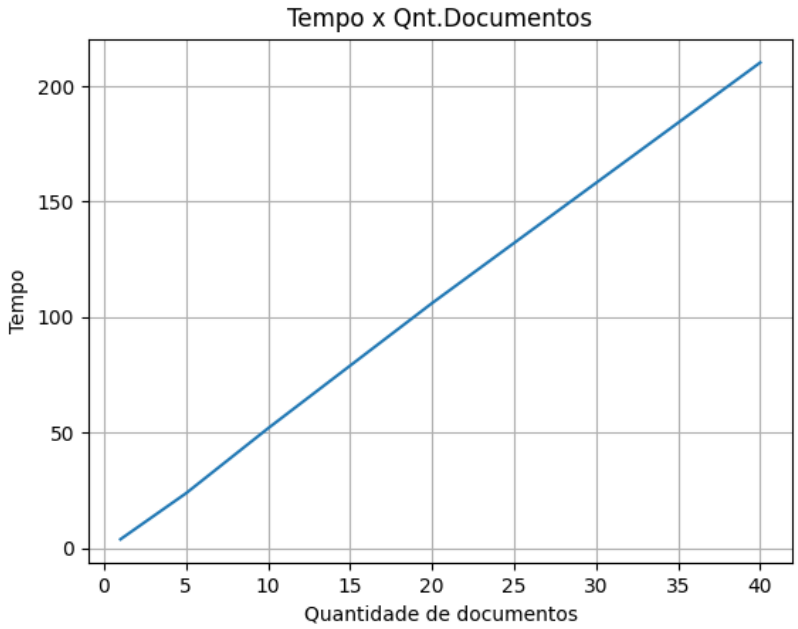

Figura 4: Gráfico de análise dos tempos de execução.

A análise após algumas simulações de tempos no método de extração de palavras-chave nos mostra que o algoritmo roda em um tempo linear, o que pode causar problemas em uma coleção de dados de um tamanho grande, o que em uma situação de mundo real será comum existirem grandes volumes de dados.

Sugestão de otimização

O primeiro problema de extração de dados a partir de um documento "pdf"pode gerar "ruídos"que levam nosso algoritmo a interpretar as palavras presentes no documento de forma errada. Como uma sugestão de resolução do problema podemos utilizar serviços em nuvem específicos para estas tarefas, como um "Amazon Textract"[10] que utiliza algoritmos de OCR (*Optical character recognition*) para identificar e extrair da melhor forma os dados textuais do documento. Em relação a segunda solução, uma forma possível de otimização seria a implementação do algoritmo "extract_keywords"da classe

[10]https://aws.amazon.com/pt/textract/

"documents"de forma que ele seja multithread ganhando assim uma melhor performance em relação ao algoritmo "single thread"já que o mesmo se prevalece dos vários núcleos dentro de um computador.

Terceira solução

Após a segunda solução já temos todos os passos necessários para implementar um software que utiliza estruturas e algoritmos de IR. Como uma última solução, e para possíveis trabalhos futuros, podemos ter a necessidade de gerar palavras-chave a partir de uma coleção de documentos possibilitando assim uma forma de generalizar a construção e a busca de documentos de diversas formas (não só apenas trabalhos de conclusão), já que os algoritmos apresentados até agora se baseavam apenas na ideia de conhecer previamente o domínio de problemas que estamos lidando.

Uma forma de gerarmos palavras-chave a partir de uma coleção de documentos utiliza métodos estatísticos como a frequência das palavras ou o mais conhecido TFIDF (*Term Frequency Inverse Document Frequency*). Segundo [1] extração de palavras-chave a partir de uma coleção de documentos segue um modelo chamado "espaço de vetores", que é uma estrutura em que os documentos são representados como um vetor n-dimensional aonde os recursos[11] e seus respectivos pesos (seguindo medidas estatísticas citadas acima) são linhas e colunas. A medida TFIDF foi inventada para resolver problemas em IR, a medida é uma extensão do cálculo de frequência dos termos existentes em uma coleção de documentos. Ela utiliza o inverso da frequência que as palavras que aparecem entre os documentos, garantindo assim que se a coleção de documentos se trata sobre "ciências jurídicas", podemos ter repetidas palavras falando sobre "processos", "direito", "justiça"porém não significa que serão importantes para relacionar as palavras-chave referentes a uma nova entrada de um documento específico, ou seja, a medida irá por mais peso somente em palavras-chaves que aparecem com maior quantidade em um documento específico porém este termo não será o termo que mais aparece mais vezes em diversos documentos. Abaixo podemos ver a fórmula para o cálculo da medida.

[11] **Recursos** são um subconjunto dos termos em um conjunto de documentos considerados mais relevantes para a pesquisa

$$TF_{ij} = f_{ij} \Big/ \sum_{i=1 \text{ to } |V|} f_{ij}$$

$$IDF_i = \log\left(N / n_i\right)$$

Figura 5: Fórmula TFIDF.

Nessas fórmulas, o significado dos símbolos são:

- TF_{ij} é a frequência do termo normalizado.
- f_{iJ} É o número de ocorrência do termo i.
- IDF é o peso de frequência do documento inverso para o termo i.
- N é o número de documentos na coleção.
- n_i é o número de documentos em que o termo i ocorre.

Observe que, se um termo i ocorre em todos os documentos, então $n_i = N$ e, portanto $IDF = log(1)$ tornando-se zero, e anulando sua importância garantindo assim que termos que aparecem múltiplas vezes em diversos documentos serão menos importantes.

Os passos para gerarmos a matriz de documentos são:

1. Pré processar os documentos fazendo a remoção de pontuação, acentos, stopwords.
2. Gerar uma matriz onde as linhas serão os documentos e as colunas suas frequências de termos normalizados.
3. Gerar a matriz da coleção de documentos com os pesos referentes a fórmula TFIDF (já que foi aprendido anteriormente a frequência normalizada dos termos).
4. Prevêr o resultado para outro documento não pertencente a matriz TFIDF feita.
5. Ordernar e retirar as "n"palavras chaves obtidas pelo algoritmo de predição.

Para resolvermos estas etapas foram utilizadas as bibliotecas Sckit-Learn com as classes "CountVectorize"e "TfidfTransformer", sendo a primeira referente a gerar a matriz de frequência normalizada e a segunda para transformar a matriz de frequência em uma matriz TFIDF.

Após a criação das palavras-chave referentes ao nosso novo documento testado podemos seguir as etapas de criação da estrutura índices invertidos que nem foi apresentado na solução dois.

Problema no algoritmo

O algoritmo acima por ser da classe de algoritmos de aprendizado de máquina, necessita de uma quantidade de dados grande e que os dados estejam com uma qualidade boa para a resolução do problema. Como citado em outras soluções nesta, utilizando aprendizado de máquina, também cai no mesmo problema como já diz o ditado em inglês: *Garbage in, garbage out* nosso pré processamento gera dados que em grande parte são úteis porém também temos erros de codificação, palavras juntas por problemas no algoritmo de extração de texto do "pdf"que, na etapa de pre processamento, acabam sendo cortas da matriz de treino.

Sugestão de otimização

O problema citado acima acaba nos gerando um problema de predição na hora de extrairmos as palavras-chave de um novo documento a partir da matriz de treino. Como uma sugestão para resolver este problema é aumentar o número de palavras-chave pertencentes ao documento tornando assim possível ele abranger um número maior de termos referentes ao documento minimizando assim o problema de ruídos interferirem no algoritmo.

Conclusão

Neste trabalho foram apresentadas três formas de solução para o problema de buscas de informações em uma base de dados de trabalhos de conclusão, uma delas sendo de força bruta, a segunda utilizando a estrutura própria que é utilizada para algoritmos em IR e a terceira que nos ajuda a generalizar e criar, a partir do conteúdo textual dos documentos, palavras-chave.

Após esta pesquisa conseguimos identificar e entender formas utilizadas para resolver problemas em uma área que abrange diversos sistemas, desde busca em bibliotecas digitais quanto em pesquisas em navegadores que utilizamos no nosso cotidiano. A área de IR continua crescendo e irá continuamente necessitar de novas pesquisas e algoritmos para suprir uma demanda de dados que cresce a cada segundo, e a possibilidade de integração com áreas de aprendizado de máquina a qual possibilita diversas otimizações de buscas, sumarização e extração de índices próprios.

Referências

[1] Ramez Elmasri and Sham Navathe. *Fundamentals of database systems*, volume 7. Pearson, 2017.

[2] Kavita Ganesan. Tutorial: Extracting keywords with tf-idf and python's scikit-learn. `https://kavita-ganesan.com/extracting-keywords-from-text-tfidf/#.YOcd8XVKgUS`, 2020.

[3] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Addison Wesley, 2nd edition, 2006.

[4] NLKT Python. Natural language toolkit website. `https://www.nltk.org/`, 2020.

[5] Scikit-Learn Python. Scikit-learn website. `https://scikit-learn.org/stable/`, 2020.