Universidade Federal de Uberlândia - Campus Monte Carmelo
Faculdade de Computação - Sistemas de Informação
GSI521 – Organização e Recuperação da Informação

Prof. Dr. Murillo G. Carneiro
Phelipe Rodovalho Santos

Trabalho 2: Modelo Vetorial

Visão Geral

A eficiência de um sistema de recuperação da informação é diretamente associada ao modelo que se utiliza, apesar dos primeiros modelos terem sido idealizados e criados nos anos 60, seus principais mecanismos ainda estão presentes em grande parte dos sistemas de recuperação atuais. A ponderação TF-IDF pode ser definida como cálculo da relevância de uma palavra em um conjunto de documentos. O significado aumenta proporcionalmente ao número de vezes que uma palavra aparece no texto, mas é compensado pela frequência da palavra no documento. O modelo de busca vetorial representa cada documento como um vetor de pesos tf-idf.
TF – privilegia os termos que mais aparecem no documento.
IDF – privilegia os termos que mais aparecem na base de documentos.
Fórmula: TF-IDF(t, d) = TF(t, d) * IDF(t)

No contexto deste trabalho será implementado um modelo vetorial para consultas, que recebe uma chave de consulta com um ou mais termos, e representa como um vetor a partir da ponderação TF-IDF, calculando a similaridade do cosseno entre o vetor de consulta e aqueles referentes à coleção de documentos. O algoritmo ranqueia os documentos em relação à similaridade com a consulta e retorna os melhores resultados ao usuário.

Algoritmo

O algoritmo implementado possui 5 funções e o método principal, será explicado detalhadamente cada uma delas:

1. def main()
2. def peso_idf(document, num_d)
3. def consulta_modelo_vetorial(pesquisa)
4. def criar_indice_invertido(document, num)
5. def prepare_doc(file)
6. def read_documents()

Entendendo melhor cada uma das funções:

def main( ):

```
def main():
    # função que lê e preenche as variaveis globais que serão usadas por todo o código
    read_documents()
    # preparando os documentos retirando pontuações e stopwords
    document01 = prepare_doc('doc1_patinho_feio.txt')
    document02 = prepare_doc('doc2_joao_maria.txt')
    document03 = prepare_doc('doc3_pinoquio.txt')
    document04 = prepare_doc('doc4_branca_neve.txt')
    document05 = prepare_doc('doc5_cinderela.txt')

    # ordenando os documentos por ordem alfabetica
    document01.sort()
    document02.sort()
    document03.sort()
    document04.sort()
    document05.sort()

    # criando indice invertido de cada um dos documentos e salvando no dicionario global dict_terms
    criar_indice_invertido(document01, 1)
    criar_indice_invertido(document02, 2)
    criar_indice_invertido(document03, 3)
    criar_indice_invertido(document04, 4)
    criar_indice_invertido(document05, 5)

    peso_idf(document01, 1)
    peso_idf(document02, 2)
    peso_idf(document03, 3)
    peso_idf(document04, 4)
    peso_idf(document05, 5)
    pprint.pprint(tfidf)

    pesquisa = str(input("Informe os termos da pesquisa:"))
    # convertendo para minusculo e separando os termos
    pesquisa = pesquisa.lower().split()

    consulta_modelo_vetorial(pesquisa)
```

def peso_idf(document, num_d):

Função responsável por calcular a ponderação TF-IDF dos termos de cada documento, recebe o documento e o número do documento.

```
def peso_idf(document, num_d):
    numTermsDocument = len(document)
    for t in document:
        # numero de arquivos que termo aparece
        numFilesAppear = len(set(dict_terms.get(t)))
        idff = (1 + math.log(numFilesAppear) *
                math.log(numTermsDocument/numFilesAppear))
        tfidf[t] = {idff, num_d}
```

def consulta_modelo_vetorial(pesquisa):

Função responsável por implementar a consulta com modelo vetorial e apresentar os resultados com melhor ranqueamento, recebe os termos da pesquisa.

```
def consulta_modelo_vetorial(pesquisa):
    docs_pesquisa = []
    for p in pesquisa:  # percorrendo os termos de pesquisa fornecidos pelo usuário
        if dict_terms.get(p) is not None:  # se o termo existir
            #print(p + " : " + dict_terms.get(p))
            for i in dict_terms.get(p):
                # inserindo o numero dos documentos de cada termo na lista
                if i not in docs_pesquisa:
                    docs_pesquisa.append(i)

    if len(docs_pesquisa) == 0:  # caso nenhum termo for encontrado nos termos mapeados, encerra
        print("Nenhum termo de pesquisa valido foi encontrado")
        exit()

    sim = []
    dict_pondera = {}
    for p in pesquisa:
        try:
            sim.append(list(tfidf.get(p)))
        except:
            continue

    for x, y in sim:
        print("\n", x)
        print("\n", y)
        dict_pondera[x] = {y}

    q = dict_pondera.keys()
    if(len(dict_pondera) > 1):
        print("A pesquisa retornou os seguintes documentos: ",
              q)
    elif(len(dict_pondera) == 1):
        print("A pesquisa retornou o seguinte documento: ",
              q)
    else:
        print("A pesquisa não retornou nenhum documento")
```

def criar_indice_invertido(document, num):

Função responsável por implementar o mapeamento dos termos do documento preenchendo um dicionário global de termos, recebe o nome do documento e o número do mesmo para ser usado como índice.

1. Percorre o documento recebido *"document"* comparando se cada termo deste documento existe no dicionário de termos *"dict_terms",* se não existir então insere o termo no *dicionário {key : value}* sendo:
    a. key = termo;
    b. value = índice referente ao número do documento;

2. Se o termo já existir no dicionário de termos *"dict_terms",* e o índice referente ao número do documento não existir no value do termo, então é feito um update no valor do value do termo adicionando o índice do documento atual.

```
def criar_indice_invertido(document, num):  # criando indice invertido
    for d in document:  # percorrendo o documento
        if d not in dict_terms:
            dict_terms.setdefault(d, [])
            dict_terms[d].append(num)
        # se o termo não esta no dict e não é um termo repetido
        if d in dict_terms and num not in dict_terms[d]:
            # x = str(dict_terms[d])+" "
            # atualiza a lista de documentos do termo
            # dict_terms.update({d: x+num})
            dict_terms[d].append(num)

    pprint.pprint(dict_terms)
```

def prepare_doc(file):

Função responsável por separar o conteúdo de cada documento em uma lista realizando operações no conteúdo:

1. Retirando quebra de linha “\n” e convertendo todos os caracteres para minúsculos.
2. Percorrendo o vetor de pontuação e retirando a pontuação da lista *“document”*.
3. Inserindo na lista final somente os termos do documento que não constam na lista de stopwords, ou seja retirando as stopwords dos termos do documento.
4. Retorna o resultado final da lista “*document”* com os termos do documento em minúsculo, sem pontuação e sem stopwords.

```
def prepare_doc(file):
    # passando o conteudo de cada arquivo para uma lista de string
    document = str(docs.get(file))
    # convertendo tudo para minusculo e retirando o \n
    document = document.lower().replace("\\n", "")

    for p in punctuation:  # percorrendo o vetor de pontuação
        document = document.replace(p, "")  # retirando a pontuação da string

    # separando no espaço todas as palavras e inserindo em uma lista
    document = document.split()
    # inserindo na lista somente as palavras que NAO constam na lista de stopwords
    document = [d for d in document if not d in stopwords]

    return document
```

def read_documents( ):

Função responsável por ler os documentos no diretório especificado na variável *path* , lê também os arquivos de pontuação e stopwords inserindo o conteúdo em variáveis globais do tipo lista. Acredito que os comentários no código abaixo complementam a explicação.

```
def read_documents():
    # definindo variaveis globais
    global punctuation
    global stopwords
    global dict_terms
    global docs

    path = "collection_docs"  # caminho do diretório que contem a coleção de documentos
    docs = {}  # criando dicionário vazio que irá armazenar o conjunto de documentos
    # criando dicionario vazio que irá armazenas os termos do conjunto de documentos
    dict_terms = {}

    # lendo o arquivo de pontuação
    with open('punctuation.txt', 'r',  encoding="utf8") as f:
        punctuation = f.read()
    # separando todas as pontuações e inserindo em formato de lista em punctuation
    punctuation = punctuation.split()

    # lendo o arquivo de stopwords
    with open('stopwords_ptbr.txt', 'r', encoding="utf8") as f:
        stopwords = f.read()
    # separando todas as stopwords e inserindo em formato de lista em stopwords
    stopwords = stopwords.split("\n")

    # lendo todos os arquivos dentro do diretorio no caminho PATH
    for filename in os.listdir(path):
        # abrindo arquivo com encoding utf8 para receber acentos e caracteres especiais adequadamente
        with open(os.path.join(path, filename), 'r', encoding="utf8") as f:
            # salvando conteudo dos arquivos na lista de documentos
            docs[filename] = f.readlines()
```