TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

KHOA KHOA HỌC MÁY TÍNH

TRUY XUẤT THÔNG TIN

ஜ★ரு

Mô Hình Vector - Space

GV hướng dẫn: **NGUYỄN TRỌNG CHỈNH**

Nhóm thực hiện:

17520898 – Võ Lê Phong

17520644 – Phạm Hoàng Đăng Khoa

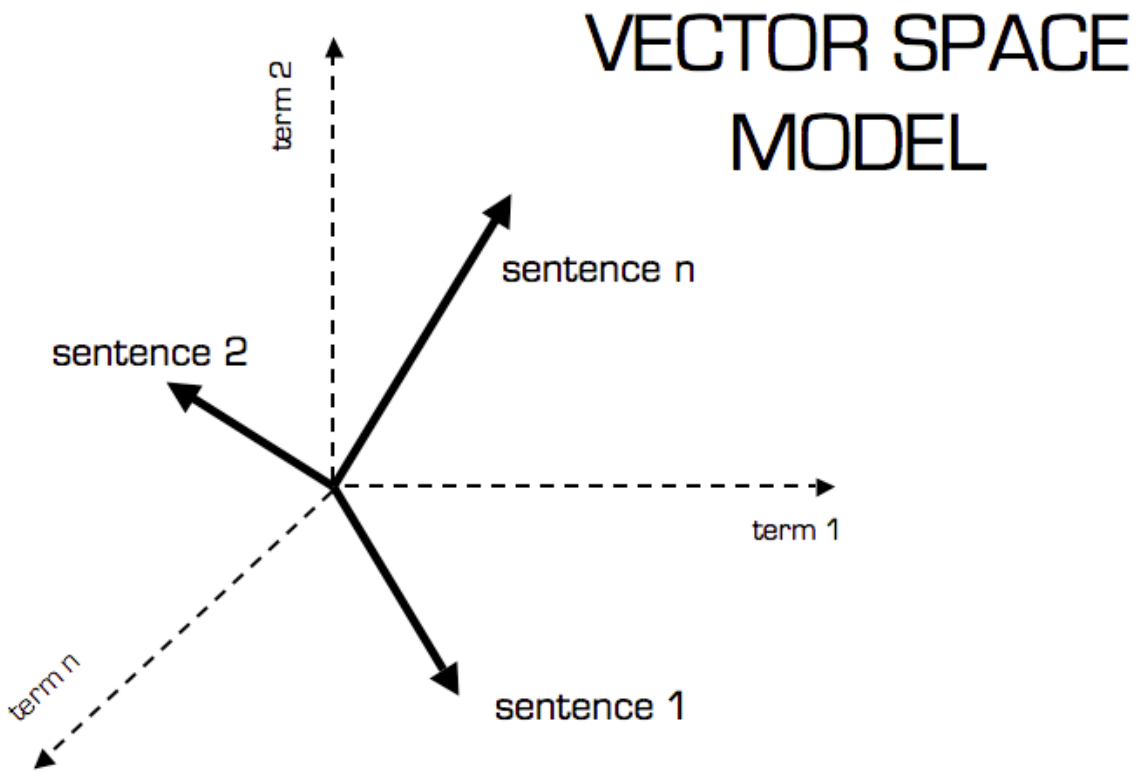
17520148 – NGUYỄN DUY HOÀI SƠN

TP.HCM, Ngày 07 tháng 1 năm 2021

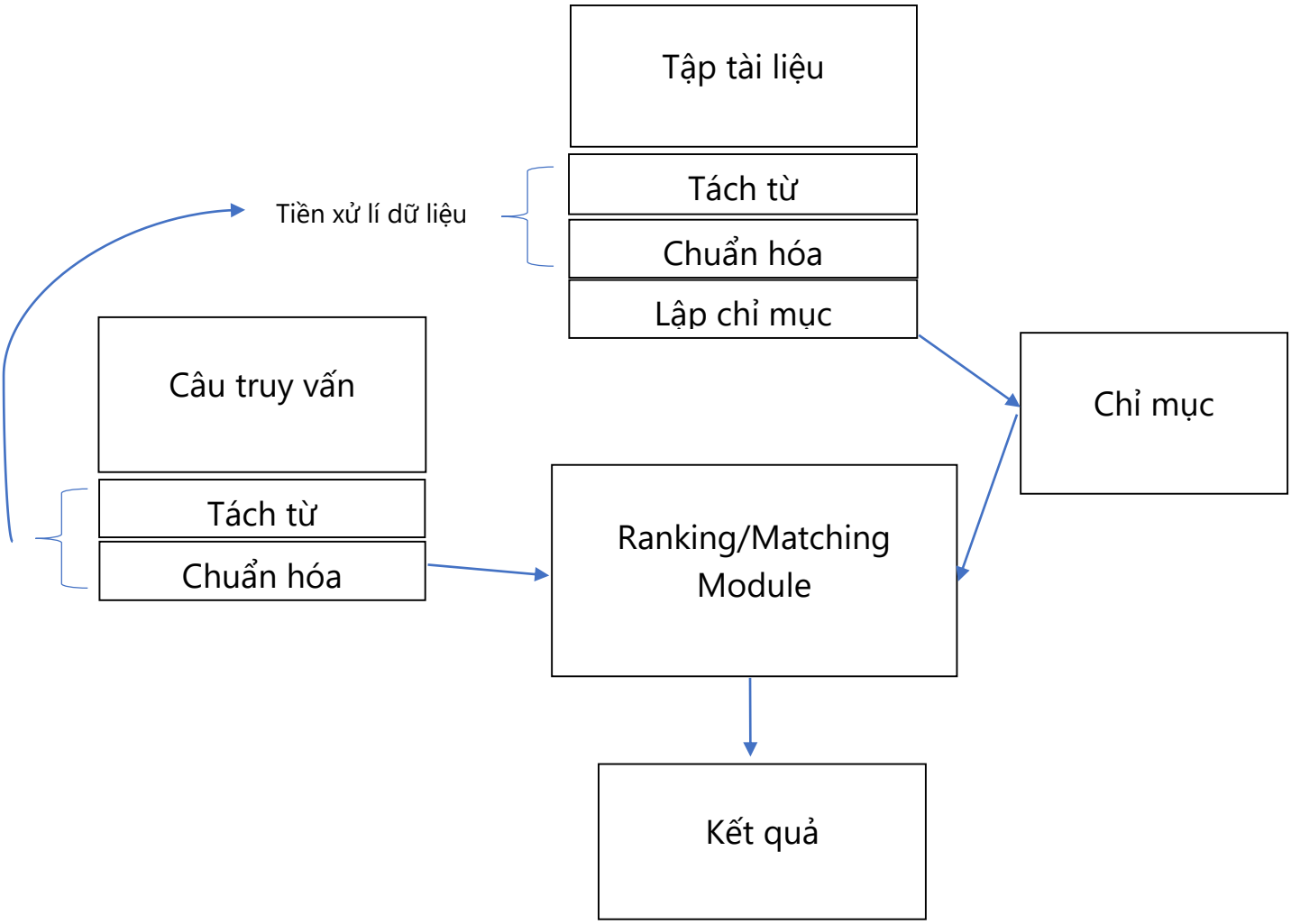
I, GIỚI THIỆU CHUNG

Nói một cách ngắn gọn, Vector space model (Mô hình không gian vector) là một mô hình đại số (algebraic model) thể hiện thông tin văn bản như một vector, các phần tử của vector này thể hiện mức độ quan trọng của một từ và cả sự xuất hiện hay không xuất hiện (Bag of words) của nó trong một tài liệu.

Mô hình này biểu diễn văn bản như những điểm trong không gian Euclid n-chiều, mỗi chiều tương ứng với một từ trong tập hợp các từ. Phần tử thứ i, là di của vector văn bản cho biết số lần mà từ thứ i xuất hiện trong văn bản. Sự tương đồng của hai văn bản được định nghĩa là khoảng cách giữa các điểm, hoặc là góc giữa những vector trong không gian.



II, MÔ HÌNH BÀI TOÁN



III, TIỀN XỬ LÍ DỮ LIỆU

- Tách các từ.

```
def load_data_in_a_directory(data_path):
    file_paths = glob.glob(data_path)
    lst_contents = []

    for file_path in file_paths:
        f = open(file_path, 'r', encoding='utf-8')
        str = f.read()
        f.close()
        words = str.replace('"', '').replace('”', '').replace('"', '').replace('.', '').replace("'","").replace(':', '').split()
        lst_contents.append(words)
    return (lst_contents, file_paths)
```

- Xóa các stopword và xóa các dấu và các số.

```
stoplist = stopwords.words("english")
punct = re.compile("[\.\?'!\$\(\)\*\^\+,\|/:0-9=><`~]|Oh")
def stopremoval(tokens):
    ret = []
    for token in tokens:
        m = punct.search(token)
        if m:
            continue
        if not token in stoplist:
            token = token.lower()
            ret.append(token)
    return ret
```

- Chuyển các từ về viết thường và nguyên mẫu (Stemming).

```
def stemming(tokens):
    ret = []
    ps = PorterStemmer()
    for token in tokens:
        ret.append(ps.stem(token))

    return ret
```

VD:

Doc1 = “what similarity laws must be obeyed when constructing aeroelastic models of heated high speed aircraft .”

Doc2 = “what are the structural and aeroelastic problems associated with flight of high speed aircraft .”

Doc3 = “what problems of heat conduction in composite slabs have been solved so far .”

- Tách từ:

Doc1 = ['what', 'similarity', 'laws', 'must', 'be', 'obeyed', 'when', 'constructing', 'aeroelastic', 'models', 'of', 'heated', 'high', 'speed', 'aircraft', '.']

Doc2 = ['what', 'are', 'the', 'structural', 'and', 'aeroelastic', 'problems', 'associated', 'with', 'flight', 'of', 'high', 'speed', 'aircraft', '.']

Doc3 = ['what', 'problems', 'of', 'heat', 'conduction', 'in', 'composite', 'slabs', 'have', 'been', 'solved', 'so', 'far', '.']

- Xóa stopword và đấu:

Doc1 = ['similarity', 'laws', 'must', 'obeyed', 'constructing', 'aeroelastic', 'models', 'heated', 'high', 'speed', 'aircraft']

Doc2 = ['structural', 'aeroelastic', 'problems', 'associated', 'flight', 'high', 'speed', 'aircraft']

Doc3 = ['problems', 'heat', 'conduction', 'composite', 'slabs', 'solved', 'far']

- Chuyển hóa từ:

Doc1 = ['similar', 'law', 'must', 'obey', 'construct', 'aeroelast', 'model', 'heat', 'high', 'speed', 'aircraft']

Doc2 = ['structur', 'aeroelast', 'problem', 'associ', 'flight', 'high', 'speed', 'aircraft']

Doc3 = ['problem', 'heat', 'conduct', 'composit', 'slab', 'solv', 'far']

IV, LẬP CHỈ MỤC

Từ ví dụ trên, ta tổng hợp lại các từ có trong cả 3 tập dữ liệu:

{ 'aeroelast', 'aircraft', 'associ', 'conduct', 'composit', 'construct', 'far', 'flight', 'high', 'heat', 'model', 'must','law', 'obey', 'problem', 'similar', 'slab', 'solv', 'structur', 'speed'}

- Tính trọng số TF:

$$tf_{dk} = \frac{f(d,k)}{f_d}$$

$f(d,k)$: Tần số xuất hiện của từ k trong d

f_d: Số lượng từ trong d

```python
def calc_tf_weighting(vocab, contents):
    TF = dict()
    # TF = np.zeros((len(vocab), len(contents))
    for i, word in enumerate(vocab):
        list_f = []
        list_df = []
        for j, content in enumerate(contents):
            count = content.count(word)
            if count > 0 :
                list_f.append(count / len(content))
                list_df.append(j)
        TF[word] = []
        TF[word].append(list_f)
        TF[word].append(list_df)
    return TF
```

Từ	Doc	TF
aeroelast	1	0.09
	2	0.125
aircraft	1	0.09
	2	0.125
Associ	2	0.125
conduct	3	0.14
composit	3	0. 14
construct	1	0.09
Far	3	0.14
Flight	2	0.125
High	1	0.09
	2	0.125
Heat	1	0.09
	3	0.14
Model	1	0.09
Must	1	0.09
Law	1	0.09
Obey	1	0.09
problem	2	0.125
	3	0.14
Similar	1	0.09
Slab	3	0.14

Solv	3	0.14
structur	2	0.125
Speed	1	0.09
	2	0.125

- Tính trọng số IDF:

$$idf_k = \log(\frac{N}{n_k})$$

N: Tổng số tài liệu

n_k: số tài liệu chứa term thứ k

Từ	IDF
aeroelast	0.4054
aircraft	0.4054
Associ	1.0986
conduct	1.0986
composit	1.0986
construct	1.0986
Far	1.0986
Flight	1.0986
Heat	0.4054
High	0.4054
Law	1.0986
Model	1.0986
Must	1.0986
Obey	1.0986
problem	0.4054
Similar	1.0986
Slab	1.0986
Solv	1.0986
Speed	0.4054
structur	1.0986

```
def calc_idf(TF, N):
    IDF = dict()
    for word in TF.keys():
        IDF[word] = np.log(N / len(TF[word][0]))
    return IDF
```

V, ĐỘ ĐO SIMILARITY

Query = “aeroelastic problems associated”

- Tiền xử lí câu query ta được:

['aeroelast', 'problem', 'associ']

- Tính TF và W của câu query:

```
def get_query_vector(splited_query, IDF):
    query_vector = np.zeros(len(splited_query))
    #get query TF
    query_TF = dict()
    for i, word in enumerate(splited_query): #Tính TF của Query
        count = splited_query.count(word)
        if word in query_TF.keys():
            continue
        query_TF[word] = count/len(splited_query)

    for termIndex, term in enumerate(splited_query): # Tính W của Query
        if term in IDF:
            query_vector[termIndex] = IDF[term]*query_TF[term]
    return query_vector
```

['aeroelast', 'problem', 'associ']

TF = [0.33 , 0.33 , 0.33]

W_Query = [0.13515504, 0.13515504, 0.3662041] (Vector của câu Query)

- Dựa vào TF và IDF, tính vector của các doc theo chiều của Vector Query:

```
def get_doc_vector(splited_query, TF, IDF):
    doc_vector = dict()
    for termIndex, term in enumerate(splited_query): # Tạo chiều cho các Doc
        if term not in TF.keys():
            continue
        for docIndex, docID in enumerate(TF[term][1]):
            if docID in doc_vector.keys():
                continue
            doc_vector[docID] = np.zeros(len(splited_query))

    for termIndex, term in enumerate(splited_query): #Tính W cho Vector_Doc
        if term not in TF.keys():
            continue
        for docIndex, docID in enumerate(TF[term][1]):
            doc_vector[docID][termIndex] = IDF[term]*TF[term][0][docIndex]

    return doc_vector
```

Doc1 = [0.03686046, 0. , 0.]

Doc2 = [0.05068314, 0.05068314, 0.05068314]

Doc3 = [0. , 0.05792359, 0.]

1, Độ đo COSINE:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

```
def cosine_similarity(query_vector, doc_vector): # Tính ranking theo cosine
    result = []
    for docID in doc_vector.keys():
        result.append((docID, dot(query_vector, doc_vector[docID]) / (norm(query_vector) * norm(doc_vector[docID]))))
    return rank_sort(result, True)
```

Độ đo của các Doc là:

Doc 1 = $\frac{0.03686046*0.33}{\sqrt{0.33^2+0.33^2+0.33^2}*\sqrt{0.03686046^2}}$ = 0.32718457421366

Doc 2 = $\frac{0.05068314*0.33 + 0.05068314*0.33+ 0.05068314*0.33}{\sqrt{0.33^2+0.33^2+0.33^2}*\sqrt{0.05068314^2+ 0.05068314^2\ 0.05068314^2}}$ = 1.0

Doc 3 = $\frac{0.05792359*0.33}{\sqrt{0.33^2+0.33^2+0.33^2}*\sqrt{0.05792359^2}}$ = 0.32718457421365993

Thứ tự là Doc2, Doc1, Doc3.

Nhược điểm: Các Doc_Vector cùng phương với nhau sẽ cho ra cùng kết quả.

2, Độ đo EUCLID:

$$d(\mathbf{d}, \mathbf{q}) = \sqrt{(d_1 - q_1)^2 + (d_2 - q_2)^2 + \cdots + (d_n - q_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (d_i - q_i)^2}.$$

```
def euclid_distance(query_vector, doc_vector):
    result = []
    for docID in doc_vector.keys():
        result.append((docID, np.linalg.norm(query_vector - doc_vector[docID])))
    return rank_sort(result)
```

Độ đo của các Doc là:

Doc 1 = $\sqrt{(0.33 - 0.03686046)^2 + (0.33 - 0)^2 + (0.33 - 0)^2}$ = 0.4025346527

Doc 2 = $\sqrt{(0.33 - 0.05068314)^2 + (0.33 - 0.05068314)^2 + (0.33 - 0.05068314)^2}$

= 0.25817811773537924

Doc 3 = $\sqrt{(0.33 - 0)^2 + (0.33 - 0.05792359)^2 + (0.33 - 0)^2}$ = 0.3979158461

Thứ tự là Doc2, Doc3, Doc1.

Nhược điểm: Docs dài hơn sẽ bị giảm hạn do chứa nhiều từ khác gây nhiễu, khiến khoảng cách xa hơn.

VI, CHẠY TRÊN BỘ DỮ LIỆU CRANFIELD

Recall	Precision Cosine	Precision Euclid
0%	54.386857443406264	67.89103117136389
10%	51.85004188384061	63.589029371993675
20%	41.02574511858882	50.805205296996455
30%	30.87532044476465	41.3285142395388
40%	25.42231091362718	33.19305815294673
50%	22.397205753632324	29.730301603320473
60%	16.147294104170555	22.71314438569602
70%	11.254869045924139	16.626174892642393
80%	8.95150011301367	13.723574935103153
90%	5.878261222851852	8.764585403823663
100%	5.480452181089815	7.96960730906635
Map	0.24879078020446344	0.3239402061477195

VII, NHẬN XÉT MÔ HÌNH

a. Ưu điểm

- Dễ hiểu và dễ cài đặt
- Đã được nghiên cứu từ lâu và thực nghiệm cho kết quả tốt và khả thi
- Hiện tại là phương pháp được sử dụng rộng rãi
- Có nhiều công thức tính TF.IDF khác nhau.

b. Nhược điểm

- Để cho kết quả đúng đắn, ta giả định:
    - Các câu là độc lập với nhau
    - Các query và documents cùng loại với nhau
- Có nhiều tham số để hiệu chỉnh
    - Bộ từ điển
    - Tham số trong các hàm Normalize
    - Threshold để chọn ra top kết quả

VIII, LỜI CẢM ƠN

Nhóm chúng em xin chân thành cảm ơn những kiến thức bổ ích từ thầy Nguyễn Trọng Chỉnh. Đã giúp chúng em có thể đạt được kết quả ngày hôm nay. Tuy nó không quá hoàn hảo để có thể xứng đáng với những gì mà bọn em được truyền đạt từ phía thầy.