

ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

----- * -----

BÀI TẬP LỚN MÔN HỌC
NHẬP MÔN HỌC MÁY VÀ KHAI PHÁ DỮ LIỆU

PHÂN LOẠI CHỦ ĐỀ TIN TỨC TIẾNG VIỆT
SỬ DỤNG CÁC PHƯƠNG PHÁP HỌC MÁY

Giảng viên hướng dẫn: TS. Nguyễn Nhật Quang

Mã lớp: 136805

Sinh viên thực hiện:

Nguyễn Tiên Nam 20194337

Võ Quang Đức 20194252

Lã Thị Hà 20194266

Nguyễn Thị Quỳnh Anh 20194223

Hà Nội, 2023

Mục lục

Danh mục hình ảnh	4
Danh mục bảng biểu	4
A. Lời mở đầu.....	5
I. Đặt vấn đề.....	5
II. Xác định bài toán.....	6
B. Tập dữ liệu bài toán.....	6
C. Các phương pháp	9
I. Tiền xử lý dữ liệu.....	9
1. Tokenizer.....	9
1.1. Maximum match:.....	9
1.2. Maximum entropy:.....	10
1.3. CRF	11
2. Loại bỏ “stop word”	11
II. Trích chọn đặc trưng.....	12
1. TF-IDF.....	12
1.1. TF:.....	13
1.2. IDF	13
1.3. Triển khai thuật toán tính TF-IDF	14
1.4. Nhược điểm của TF-IDF.....	16
2. SVD.....	17
2.1. Cơ sở toán học	17
2.2. Áp dụng cho bài toán giảm chiều dữ liệu:.....	19
III. Xây dựng mô hình.....	20
1. Naive-Bayes	20
1.1. Định lý Bayes.....	20
1.2. Mô hình phân lớp Naive Bayes	21

1.3.	Mô hình Multinomial Naive Bayes	23
1.4.	Triển khai mô hình Multinomial Naive Bayes	25
2.	Mạng nơ-ron nhân tạo	25
2.1.	Mạng nơ-ron nhân tạo (Neural Network) là gì?	25
2.2.	Hoạt động của các nơ-ron	26
2.3.	Mô hình Mạng nơ-ron nhân tạo tổng quát.....	26
2.4.	Hàm kích hoạt (activation function)	28
3.	K-Nearest-Neighbor	31
3.1.	K-Nearest-Neighbor là gì?.....	31
3.2.	Thí nghiệm để tìm tham số k tốt nhất	31
4.	Tiêu chí đánh giá	32
D.	Kết quả thí nghiệm.....	33
I.	Thử nghiệm mô hình Multinomial Naive Bayes	33
II.	Thử nghiệm mô hình Deep Neural Network	34
III.	Thử nghiệm với mô hình K-Nearest-Neighbor.....	36
IV.	Confusion matrix	36
E.	Kết luận và cải tiến.....	37
	Danh mục tài liệu tham khảo	38

Danh mục hình ảnh

Hình 1. Biểu đồ minh họa phân bố nhãn chủ đề.....	8
Hình 2. Phân bố độ dài văn bản sau khi loại bỏ stop word.....	12
Hình 3. Xây dựng tập từ điển từ tập dữ liệu	14
Hình 4. Hàm tính số lượng văn bản mà một từ ‘word’ có trong	14
Hình 5. Hàm tính tần số xuất hiện của một từ trong văn bản	15
Hình 6. Hàm tính IDF cho một từ.....	15
Hình 7. Hàm tính TF-IDF cho văn bản.....	15
Hình 8. Biểu diễn dạng vector của một văn bản ngắn	16
Hình 9. Source code tính TF-IDF cho tập dữ liệu văn bản.....	16
Hình 10. Source code giảm chiều vector TF-IDF bằng hàm TruncatedSDV	19
Hình 11. Source code tìm α tốt nhất để accuracy lớn nhất	25
Hình 12. Source code triển khai mô hình Multinomial Naive Bayes	25
Hình 13. Mô hình giải phẫu thần kinh nơ-ron	26
Hình 14. Kiến trúc mạng nơ-ron nhân tạo	27
Hình 15. Hình ảnh minh họa cho hoạt động của các node trong mạng nơ-ron	28
Hình 16. Đồ thị hàm Sigmoid	29
Hình 17. Đồ thị hàm Tanh	30
Hình 18. Đồ thị hàm ReLU.....	30
Hình 19. Kết quả thử nghiệm mô hình kNN.....	32
Hình 20. Kết quả thu được với mô hình Naive Bayes trên tập validation.....	33
Hình 21. Kết quả thử nghiệm mô hình Multinomial Naive Bayes	34
Hình 22. Đồ thị accuracy theo từng epoch	35
Hình 23. Đồ thị loss theo từng epoch	35
Hình 24. Ma trận nhầm lẫn của mô hình Naive Bayes	37

Danh mục bảng biểu

Bảng 1. Chi tiết tập dữ liệu	6
------------------------------------	---

A. Lời mở đầu

Ngày nay, trí tuệ nhân tạo và học máy đang dần len lỏi vào từng khía cạnh khác nhau của cuộc sống đời thường. Trí tuệ nhân tạo giúp công việc con người trở nên dễ dàng và tự động hơn.

Với mong muốn tìm hiểu và ứng dụng các mô hình Học máy đã được ở môn học, chúng em quyết định lựa chọn đề tài “Phân loại văn bản tiếng Việt sử dụng mô hình học máy”. Với đề tài này, chúng em mong muốn thử nghiệm và đưa ra mô hình có khả năng phân loại tốt cho văn bản tiếng Việt.

Trong dự án này, chúng em sẽ tiếp cận các phương pháp và thuật toán Học máy khác nhau. Báo cáo được chia làm các phần: Phân tích bộ dữ liệu, Các phương pháp xử lý và xây dựng mô hình, Kết quả thí nghiệm và Kết luận cải tiến.

Chúng em xin cảm ơn thầy Nguyễn Nhật Quang đã hỗ trợ bọn em trong quá trình thực hiện đề tài.

I. Đặt vấn đề

Đi cùng với sự phát triển loài người là sự bùng nổ của thông tin dữ liệu. Trong kho dữ liệu lớn đó, lượng dữ liệu văn bản chiếm một phần rất lớn, không ngừng tăng trưởng và cần được xử lý, khai thác. Điển hình như hiện nay, các trang báo điện tử ra đời nhằm cung cấp thông tin, kiến thức cho người đọc. Bên cạnh những công việc như tự động cập nhật, tổng hợp tin tức mới hàng ngày thì những trang báo này phải tổ chức, sắp xếp và phân loại những tin tức này vào đúng chủ đề của chúng. Tuy nhiên, mỗi ngày báo điện tử cung cấp hàng trăm tin tức, cho nên việc phân loại bằng tay sẽ vô cùng tốn kém và độ chính xác có thể không cao. Điều này được lý giải như sau: Văn phong tiếng Việt đa hình đa nghĩa, vô cùng phong phú, việc đọc hết một tin tức dài và phân loại nó vào đúng chủ đề là rất khó và có thể mất đến vài phút, cộng thêm việc thế giới quan của mỗi người khác nhau nên kết quả phân loại cũng sẽ khác nhau. Trong trường hợp này, kết quả phân loại sẽ mang tính chủ quan cao và chắc chắn độ chính xác cũng không cao. Tóm lại, việc giải quyết công việc phân loại tin tức một cách thủ công có hai vấn đề lớn đó là độ chính xác thấp và thời gian thực hiện lâu, trong khi đó, báo điện tử phải đáp ứng yêu cầu “nhanh” và chính xác”.

Nhận thấy công việc này có thể giải quyết bằng các mô hình học máy, nhóm chúng em sẽ quyết định xây dựng, tìm ra một mô hình có thể phân loại tin tức dựa trên nội dung của tin tức đó. Việc phân loại sử dụng các thuật toán học máy có thể cải thiện

đáng kể độ chính xác, hơn thế nữa, một mô hình học máy có thể tiến hành phân loại hàng trăm tới hàng nghìn tin tức mỗi giây, nhanh hơn rất nhiều so với các làm thủ công. Đặc biệt, con người cần hoạt động theo chu kỳ sinh học nhất định thì máy có thể hoạt động cả ngày đêm và hoàn toàn tự động. Tóm lại, việc ứng dụng học máy sẽ cho kết quả và hiệu năng vượt bậc với cách thông thường.

II. Xác định bài toán

Bài toán cần lấy bộ dữ liệu từ các chủ đề khác nhau. Sau đó chúng em tiến hành xử lý dữ liệu thô (dạng text) để trích chọn những đặc trưng khác nhau của văn bản.

Các mô hình học máy được nhóm sử dụng: k-Nearest-Neighbor, Naïve Bayes và Neural Network để phân lớp dự đoán

B. Tập dữ liệu bài toán

Dữ liệu huấn luyện cho bài toán được thu thập tại github:

<https://github.com/duyvuleo/VNTC>

Ta sẽ chia bộ dữ liệu (dataset) thành 2 phần: data **train** và data **test**. Bài toán phân lớp này sẽ chia thành 2 công việc:

- Huấn luyện:

Dùng bộ Training Text (data train) rút trích thành các bộ Features Vector đưa vào Machine Learning Algorithm từ đó thành một Predictive Model dùng để phân loại sau này, cuối cùng kết quả sẽ thu được Expect Label tên lớp cần phân loại.

- Hiện thực kết quả:

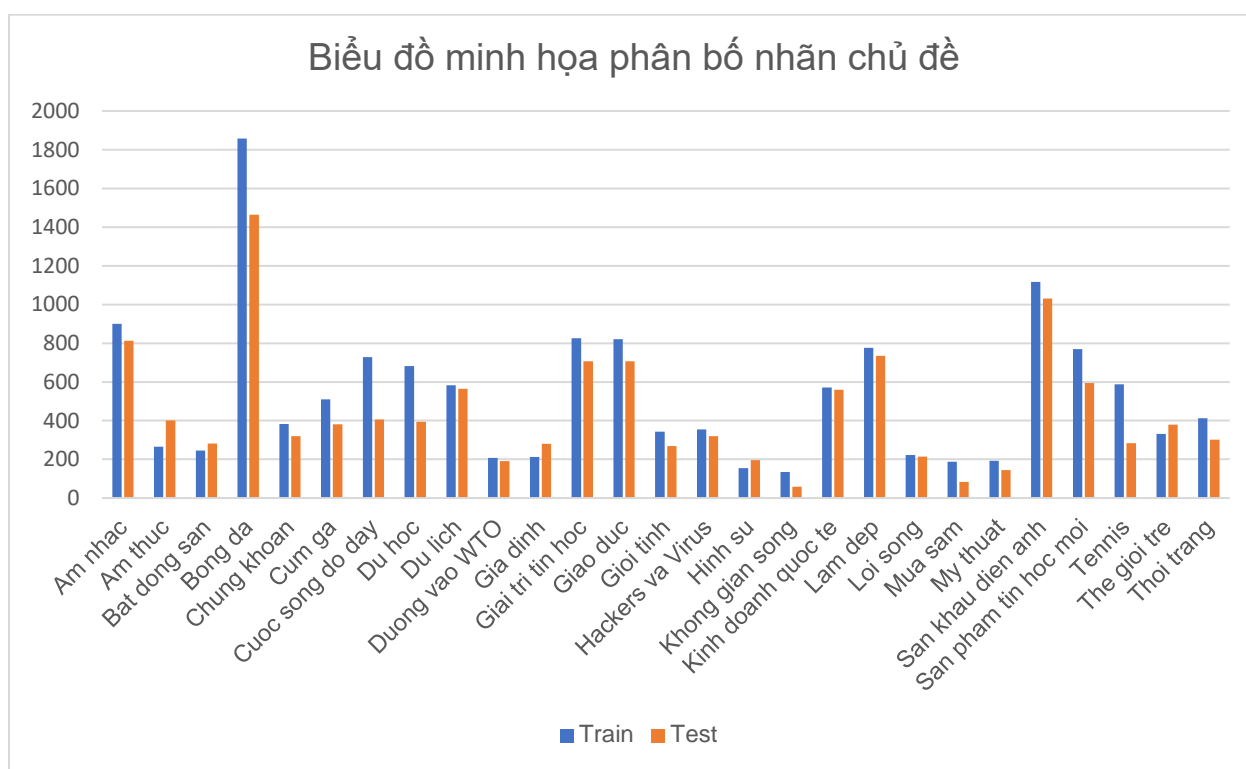
Dùng Text (data test) rút trích thành các bộ Features Vector đưa vào Predictive Model sẽ nhận được Expect Label tên lớp cần phân loại.

Bảng 1. Chi tiết tập dữ liệu

	Train	Test
Am nhạc	900	813
Am thực	265	400

Bat dong san	246	282
Bong da	1857	1464
Chung khoan	382	320
Cum ga	510	381
Cuoc song do day	729	405
Du hoc	682	394
Du lich	582	565
Duong vao WTO	208	191
Gia dinh	213	280
Giai tri tin hoc	825	707
Giao duc	821	707
Gioi tinh	343	268
Hackers va Virus	355	319
Hinh su	155	196
Khong gian song	134	58
Kinh doanh quoc te	571	559
Lam dep	776	735
Loi song	223	214
Mua sam	187	84
My thuat	193	144

San khau dien anh	1117	1030
San pham tin hoc moi	770	595
Tennis	588	283
The gioi tre	331	380
Thoi trang	412	302



Hình 1. Biểu đồ minh họa phân bố nhãn chủ đề

C. Các phương pháp

I. Tiền xử lý dữ liệu

1. Tokenizer

Tokenization (tách từ) là một trong những bước quan trọng nhất trong quá trình tiền xử lý văn bản. Tokenization là quá trình tách một cụm từ, câu, đoạn văn, một hoặc nhiều tài liệu văn bản thành các đơn vị nhỏ hơn. Mỗi đơn vị nhỏ hơn này được gọi là Tokens.

Có thể coi tokens là các khối xây dựng của NLP và tất cả các mô hình NLP đều xử lý văn bản thô ở cấp độ các Tokens. Chúng được sử dụng để tạo từ vựng trong một kho ngữ liệu (một tập dữ liệu trong NLP). Từ vựng này sau đó được chuyển thành số (ID) và giúp chúng ta lập mô hình.

Một điểm đặc biệt trong văn bản tiếng Việt đó là đơn vị từ từ bao gồm từ đơn (một từ chỉ gồm một tiếng) và từ ghép (một từ có thể được kết hợp bởi nhiều tiếng khác nhau), ví dụ như: thủ tướng, bắt đầu,... khác với tiếng Anh và một số ngôn ngữ khác, các từ được phân cách nhau bằng khoảng trắng. Vì vậy, chúng ta cần phải tách từ và nối các từ ghép với nhau bằng dấu “_” để có thể đảm bảo ý nghĩa của từ được toàn vẹn.

Một số phương pháp phổ biến để tách từ tiếng Việt như:

1.1. *Maximum match:*

Ý tưởng của thuật toán này là duyệt một câu từ trái qua phải và chọn từ có nhiều tiếng nhất mà có mặt trong từ điển tiếng Việt. Nội dung thuật toán này dựa trên thuật toán đã được ChihHao Tsai giới thiệu năm 1996, gồm 2 dạng:

- Dạng đơn giản: Giả sử có một chuỗi các tiếng trong câu là t_1, t_2, \dots, t_N . Thuật toán sẽ kiểm tra xem t_1 có mặt trong từ điển hay không, sau đó kiểm tra tiếp t_1-t_2 có trong từ điển hay không. Tiếp tục như vậy cho đến khi tìm được từ có nhiều tiếng nhất có mặt trong từ điển, và đánh dấu từ đó. Sau đó tiếp tục quá trình trên với tất cả các tiếng còn lại trong câu và trong toàn bộ văn bản. Dạng này khá đơn giản nhưng nó gặp phải rất nhiều nhập nhằng trong tiếng Việt
- Dạng phức tạp: dạng này có thể tránh được một số nhập nhằng gặp phải trong dạng đơn giản. Đầu tiên thuật toán kiểm tra xem t_1 có mặt trong từ điển không, sau đó kiểm tra tiếp t_1-t_2 có mặt trong từ điển không. Nếu t_1-t_2 đều có mặt

trong từ điển thì thuật toán thực hiện chiến thuật chọn 3-từ tốt nhất. Tiêu chuẩn chọn 3 từ tốt nhất là:

- Độ dài trung bình của 3 từ là lớn nhất
- Sự chênh lệch độ dài của 3 từ là ít nhất

Ưu điểm của phương pháp: đơn giản, dễ hiểu và chạy nhanh.

Nhược điểm của phương pháp: nó không giải quyết được 2 vấn đề quan trọng nhất của bài toán phân đoạn từ tiếng Việt: thuật toán gặp phải nhiều nhập nhằng, hơn nữa nó hoàn toàn không có chiến lược gì với những từ chưa biết.

1.2. Maximum entropy:

Ý tưởng chính của Maximum Entropy là “ngoài việc thỏa mãn một số ràng buộc nào đó thì mô hình càng đồng đều càng tốt”. Ta xem xét bài toán toán phân lớp gồm có 4 lớp. Ràng buộc duy nhất mà chúng ta biết là trung bình 40% các tài liệu chứa từ “professor” thì nằm trong lớp faculty. Trực quan cho thấy nếu có một tài liệu chứa từ “professor” chúng ta có thể nói có 40% khả năng tài liệu này thuộc lớp faculty, và 20% khả năng cho các khả năng còn lại (thuộc một trong 3 lớp còn lại). Mặc dù maximum entropy có thể được dùng để ước lượng bất kỳ một phân phối xác suất nào, chúng ta cần tập trung vào việc học ra phân phối điều kiện của chuỗi nhãn tương ứng với chuỗi (xâu) đầu vào cho trước

Trong maximum entropy, người ta dùng dữ liệu huấn luyện để xác định các ràng buộc trên phân phối điều kiện. Mỗi ràng buộc thể hiện một đặc trưng nào đó của dữ liệu huấn luyện. Maximum Entropy cho phép chúng ta giới hạn các phân phối mô hình lý thuyết gần giống nhất các giá trị kì vọng cho các đặc trưng này trong dữ liệu huấn luyện D. Người ta mô hình hóa xác suất $P(o | s)$ (o là chuỗi đầu vào và s là chuỗi nhãn đầu ra):

$$P(o | s) = \frac{1}{Z(o)} \exp \left(\sum_i \lambda_i f_i(o, s) \right)$$

Trong đó:

- $P(o | s)$ là một đặc trưng

- λ_i là một tham số cần phải ước lượng
- $Z(o)$ là thừa số chuẩn hóa nhằm đảm bảo tính đúng đắn của định nghĩa xác suất

1.3. CRF

CRF là một khung để xây dựng các mô hình xác suất giúp phân đoạn và gán nhãn dữ liệu tuần tự hay các thực thể định danh. Các thực thể đó biểu thị cho các từ trong câu đại diện cho các đối tượng như tên người, tổ chức, địa điểm,... CRF là thuật toán xác suất có điều kiện $P(y|x)$ với xác suất của vector đầu ra y của một biến ngẫu nhiên được cung cấp bởi một vector đặc trưng x .

Để dự đoán chuỗi thích hợp thì ta phải tối đa hóa xác suất có điều kiện và ta lấy chuỗi với xác suất lớn nhất:

$$\hat{y} = \operatorname{argmax}_y P(y|x)$$

Một cách tổng quát, mô hình CRF sử dụng cho việc gán nhãn thực thể, giải quyết nhược điểm sai lệch nhãn do các nhãn độc lập với nhau của mô hình Markov ẩn. CRF trước tiên xác định các hàm tính năng cần thiết, khởi tạo trọng số lambda cho các giá trị ngẫu nhiên và sau đó áp dụng phương pháp gradient descent lặp đi lặp lại cho đến khi các giá trị tham số hội tụ.

Trong CRF, dữ liệu đầu vào là tuần tự và chúng ta phải dựa vào ngữ cảnh trước đó để đưa ra dự đoán về một điểm dữ liệu

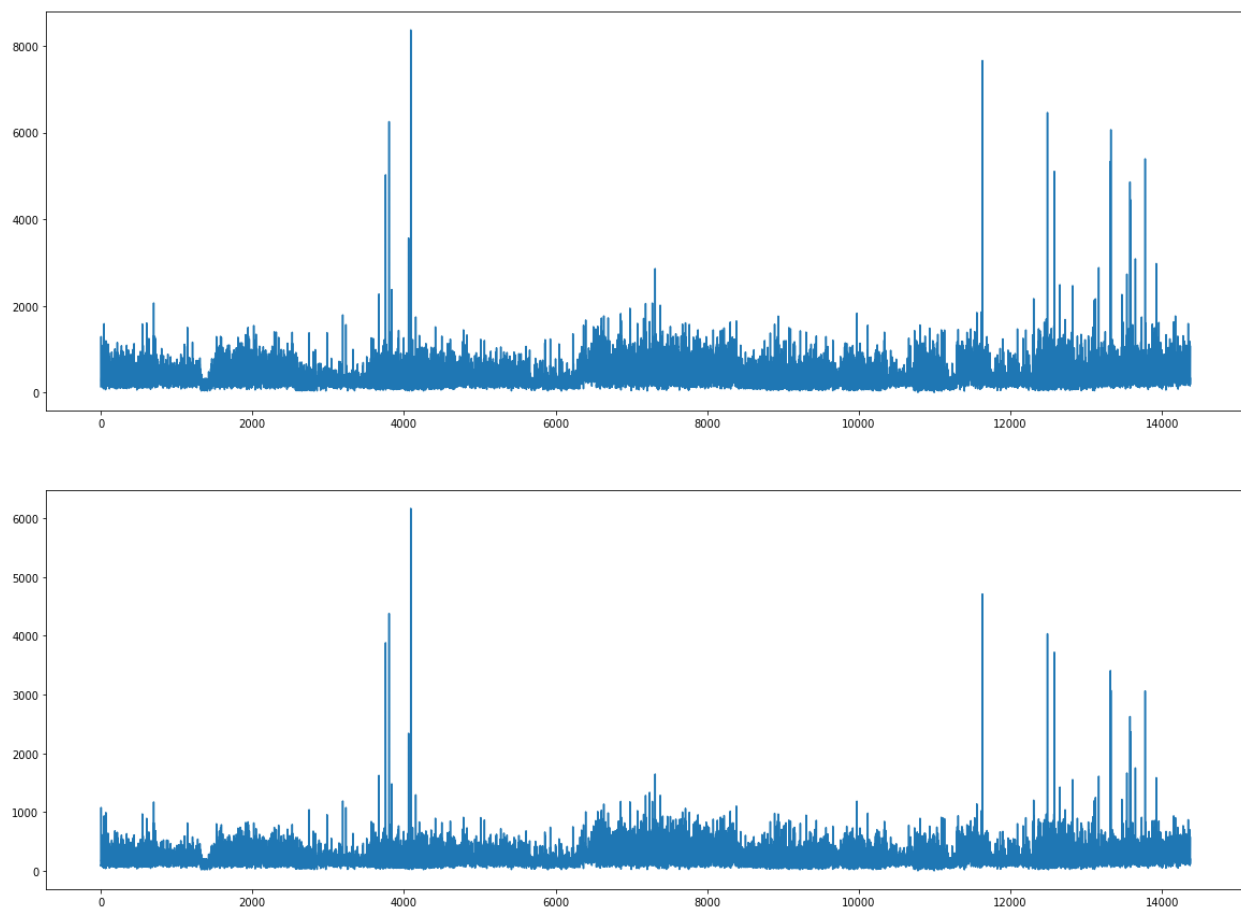
Hiện nay, có khá nhiều thư viện mã nguồn mở của bài toán này. Với ngôn ngữ Python, có 2 thư viện phổ biến là [underthesea](#) hoặc [pyvi](#).

2. Loại bỏ “stop word”

Bên cạnh các xử lý trên, chúng ta còn phải cần quan tâm đến 1 xử lý khác nữa. Đó là việc loại bỏ stop word. Bước xử lý này góp phần giúp giảm số lượng từ, loại bỏ các từ không mang ý nghĩa, tăng tốc độ học và xử lý, và nâng cao kết quả của mô hình.

Stopwords là những từ xuất hiện nhiều trong ngôn ngữ tự nhiên, tuy nhiên lại không mang nhiều ý nghĩa. Ở tiếng Việt Stopwords là các từ nối (của, là, có, được, những...). Danh sách stopwords phải được xây dựng từ bộ dữ liệu văn bản lớn, của

toàn bộ dữ liệu. Để loại bỏ các stopwords, ta sẽ làm theo phương pháp dựa trên tần suất xuất hiện của từ. Ta tiến hành đếm số lần xuất hiện của từng từ trong dataset. Dữ liệu về các tần suất xuất hiện các từ sẽ được lưu vào 1 cấu trúc dictionary với key là chính từ ấy, value là số lần xuất hiện. Dữ liệu sau đó được sắp xếp theo chiều giảm dần value, tức là các từ có tần suất xuất hiện càng cao thì xếp trên đầu và nhiều khả năng là các stopwords. Ta lấy top 100 (có thể lấy nhiều hoặc ít hơn) của dữ liệu tần suất vừa thu được, sẽ là các từ stopwords cần loại bỏ.



Hình 2. Phân bố độ dài văn bản sau khi loại bỏ stop word

II. Trích chọn đặc trưng

1. TF-IDF

Mục đích: chuyển hóa văn bản đã tách từ thành các vector đặc trưng

Sử dụng TF-IDF (term frequency – inverse document frequency) là trọng số của một từ trong văn bản thu được qua thống kê thể hiện mức độ quan trọng của từ này trong một văn bản, mà bản thân văn bản đang xét nằm trong một tập hợp các văn bản. Trọng số này được sử dụng để đánh giá tầm quan trọng của một từ trong một văn bản. Giá trị cao thể hiện độ quan trọng cao (phụ thuộc vào số lần từ xuất hiện trong văn bản), nhưng bù lại bởi tần suất của từ đó trong tập dữ liệu (có những từ như: và, nhưng,... mặc dù xuất hiện nhiều trong văn bản, nhưng lại xuất hiện nhiều trong tập dữ liệu nên mức độ quan trọng sẽ được giảm đi).

Cách tính TF-IDF

1.1. TF:

Dùng để ước lượng tần xuất xuất hiện của từ trong văn bản. Tuy nhiên với mỗi văn bản thì có độ dài khác nhau, vì thế số lần xuất hiện của từ có thể nhiều hơn . Vì vậy số lần xuất hiện của từ sẽ được chia độ dài của văn bản (tổng số từ trong văn bản đó)

Công thức:

$$TF(t, d) = (\text{số lần từ } t \text{ xuất hiện trong văn bản } d) / (\text{tổng số từ trong văn bản } d)$$

1.2. IDF

Dùng để đo mức độ quan trọng của một từ trong một số lượng lớn các văn bản. Ví dụ, một số từ nói: và, nhưng, ở,... xuất hiện nhiều trong tất cả các văn bản nhưng thường không có ý nghĩa quan trọng. Vì vậy ta cần giảm đi mức độ quan trọng của những từ đó bằng cách sử dụng IDF

$$IDF(t, D) = \log(\text{Tổng số văn bản trong tập mẫu } D / \text{Số văn bản có chứa từ } t)$$

$$\text{Giá trị của TF-IDF} = TF(t, d) * IDF(t, D)$$

Trong đó: t là một từ riêng lẻ

d là một văn bản

D là một tập văn bản

1.3. Triển khai thuật toán tính TF-IDF

B1: Tiền xử lý

```
#preprocessing the text data
sentences = []
word_set = []

for sent in text:
    x = [i.lower() for i in sent if i.isalpha()]
    sentences.append(x)
    for word in x:
        if word not in word_set:
            word_set.append(word)
word_set = set(word_set)
total_documents = len(sentences)
index_dict = {}
i = 0
for word in word_set:
    index_dict[word] = i
    i += 1
```

Hình 3. Xây dựng tập từ điển từ tập dữ liệu

Chú thích:

- sentences là tập các bài báo (D)
- word_set là tập các từ xuất hiện trong bài báo d (các từ trong word_set là khác nhau)
- text: mảng chứa tất cả các văn bản

B2: Tính số văn bản mà mỗi từ trong word_set xuất hiện

```
def count_dict(sentences):
    word_count = {}
    for word in word_set:
        word_count[word] = 0
        for sent in sentences:
            if word in sent:
                word_count[word] += 1
    return word_count
word_count = count_dict(sentences)
```

Hình 4. Hàm tính số lượng văn bản mà một từ 'word' có trong

Chú thích: word_count là một dict với key là một từ riêng lẻ value là số văn bản mà từ đó xuất hiện

B3: Tính tần suất xuất hiện của một từ word trong một văn bản document (TF)

```
def termfreq(document, word):  
    N = len(document)  
    occurrence = len([token for token in document if token == word])  
    return occurrence/N
```

Hình 5. Hàm tính tần số xuất hiện của một từ trong văn bản

B4: Tính IDF

```
def inverse_doc_freq(word):  
    try:  
        word_occurrence = word_count[word] + 1  
    except:  
        word_occurrence = 1  
    return np.log(total_documents/word_occurrence)
```

Hình 6. Hàm tính IDF cho một từ

B5: Tính TF-IDF cho mỗi văn bản d

```
[20] def tf_idf(sentence):  
    tf_idf_vec = np.zeros((len(word_set),))  
    for word in sentence:  
        tf = termfreq(sentence, word)  
        idf = inverse_doc_freq(word)  
        value = tf*idf  
        tf_idf_vec[index_dict[word]] = value  
    return tf_idf_vec
```

Hình 7. Hàm tính TF-IDF cho văn bản

Kết quả: Thu được biểu diễn dạng vector của văn bản d

```
▶ vectors = [];  
for sent in sentences:  
    vec = tf_idf(sent)  
    vectors.append(vec)  
print(vectors[0])
```

```
[0.         0.         0.         0.         0.         0.  
 0.         0.         0.         0.         0.         0.42374168  
 0.         0.         0.         0.         0.         0.  
 0.         0.         0.         0.         0.         0.  
 0.         0.         0.         0.         0.         0.  
 0.         0.         0.         0.         0.         0.  
 0.4540656  0.         0.         0.         0.         0.  
 0.         0.         0.         0.         0.         0.  
 0.87165326 0.         ]
```

Hình 8. Biểu diễn dạng vector của một văn bản ngắn

Ngoài cách trên, chúng em sử dụng hàm `TfidfVectorizer()` do thư viện `sklearn` cung cấp để tính nhanh TF-IDF của tập văn bản

```
▶ from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer  
tfidf_vect = TfidfVectorizer(ngram_range=(1, 2), max_df=0.5, analyzer='word', max_features=30000)  
tfidf_vect.fit(X_data) # learn vocabulary and idf from training set
```

Hình 9. Source code tính TF-IDF cho tập dữ liệu văn bản

- Tham số `ngram_range=(min_n, max_n)`: giá trị nhỏ nhất và lớn nhất của số lượng từ tạo nên đặc trưng.
- Tham số `max_df`: một số thực là giá trị lớn nhất của DF, đối với những từ có DF lớn hơn giá trị `max_df` thì sẽ lấy giá trị là `max_df`
- Tham số `analyzer`: nhậm một trong 3 giá trị {'word', 'char', 'char_wb'}, tương ứng với việc coi những đặc trưng (các từ trong từ điển) một từ, ký tự hay n_từ
- Tham số `max_feature`: số lượng đặc trưng tối đa

1.4. Nhược điểm của TF-IDF

Sau khi thực hiện TF-IDF, chúng ta dễ dàng nhận thấy rằng, ma trận mà chúng ta thu được có kích thước rất lớn, và việc xử lý tính toán với ma trận này đòi hỏi thời gian và bộ nhớ khá tốn kém. Giả sử, chúng ta có 100.000 văn bản và bộ từ điển bao gồm 50000 từ, khi đó ma trận mà chúng ta thu được sẽ có kích thước là $100000 \times$

50000. Giả sử mỗi phần tử được lưu dưới dạng float32 - 4 byte, bộ nhớ mà chúng ta cần sử dụng là:

$$100000 \times 50000 \times 4 = 20000000000 \text{ byte}$$

tức là chúng ta tốn tầm 18.63GB bộ nhớ, khó có thể lưu hết vào RAM để thực hiện tính toán.

2. SVD

Trong thực tế, với số lượng văn bản khổng lồ và từ điển lên đến hàng trăm nghìn từ, bộ nhớ mà chúng ta sử dụng còn tốn kém hơn rất nhiều. Để xử lý vấn đề này, chúng ta sẽ sử dụng thuật toán SVD (**singular value decomposition**) nhằm mục đích giảm chiều dữ liệu của ma trận mà chúng ta thu được sau khi tính TF-IDF, với SVD ta sẽ loại bỏ được những “stop word” không mang nhiều ý nghĩa mà vẫn giữ nguyên được các thuộc tính quan trọng của ma trận gốc ban đầu.

2.1. Cơ sở toán học

Một ma trận $A_{m \times n}$ bất kỳ đều có thể phân tích thành dạng:

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} (V_{n \times n})^T$$

Trong đó:

U, V : là các ma trận trực giao,

Σ : là ma trận *đường chéo* với các phần tử trên đường chéo $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq 0 = 0 = \dots = 0$

Mặc dù, Σ không phải ma trận vuông, ta vẫn có thể coi nó là ma trận chéo nếu các thành phần khác không của nó chỉ nằm ở vị trí *đường chéo*, tức tại các vị trí có chỉ số hàng và chỉ số cột là như nhau.

Số lượng các phần tử khác 0 trong Σ chính là rank của ma trận A .

- Compact SVD

A có thể được biểu diễn bằng tổng của các ma trận có rank bằng 1

$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T$$

Trong cách biểu diễn này, ma trận A chỉ phụ thuộc vào r cột đầu tiên của U, V và r giá trị khác 0 trên đường chéo của ma trận Σ . Vì vậy ta có một cách phân tích gọn hơn và gọi là *compact SVD*:

$$\mathbf{A} = \mathbf{U}_r \Sigma_r (\mathbf{V}_r)^T$$

Với $\mathbf{U}_r, \mathbf{V}_r$ lần lượt là ma trận được tạo bởi r cột đầu tiên của U và V.

Σ_r là ma trận con được tạo bởi r hàng đầu tiên và r cột đầu tiên của Σ .

- Truncated SVD

Trong ma trận Σ , các giá trị trên đường chéo ($\sigma_1, \sigma_2 \dots, \sigma_r$) là không âm và giảm dần. Khi đó có thể xấp xỉ ma trận A bằng tổng của $k < r$ ma trận có rank 1

$$\mathbf{A} \approx \mathbf{A}_k = \mathbf{U}_k \Sigma_k (\mathbf{V}_k)^T = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

Sai số do cách xấp xỉ trên là căn bậc hai của tổng bình phương của các singular values mà ta đã bỏ qua ở phần cuối của Σ . Ở đây sai số được định nghĩa là Frobenius norm của hiệu hai ma trận:

$$\|\mathbf{A} - \mathbf{A}_k\|_F^2 = \sum_{i=k+1}^r \sigma_i^2$$

Có thể kết luận rằng: sai số do xấp xỉ càng nhỏ nếu phần singular values bị *truncated* có giá trị càng nhỏ so với phần singular values được giữ lại.

Tổng quát: nếu ta muốn giữ lại ít nhất a% lượng thông tin trong A, trước hết ta cần chọn k là số nhỏ nhất sao cho:

$$\frac{\sum_{i=1}^k \sigma_i^2}{\sum_{j=1}^r \sigma_j^2} \geq a$$

Ưu điểm: Giảm chi phí tính toán, có lợi về lưu trữ nếu $k \ll m, n$. Để lưu ma trận với Truncated SVD, ta sẽ lưu các ma trận U_k, Σ_k, V_k . Khi đó tổng số phần tử phải lưu là $k(m+n+1)$, với chú ý rằng ta chỉ cần lưu các giá trị trên đường chéo của Σ_k

Giả sử mỗi phần tử được lưu bởi một số thực 4 byte, thì số byte cần lưu trữ là $4k(m+n+1)$. Nếu so giá trị này với ảnh gốc có kích thước mn mỗi giá trị là 1 số nguyên 1 byte, tỉ lệ nén là:

$$\frac{4k(m+n+1)}{mn}$$

Tỉ lệ này nhỏ hơn 1

2.2. Áp dụng cho bài toán giảm chiều dữ liệu:

Áp dụng cho ma trận TF-IDF của dữ liệu, m cột của U đại diện cho một cơ sở trực giao cho không gian vector kéo dài bởi các hàng của A . Với giả thiết, A là ma trận TF-IDF, các cột của U sẽ biểu thị một cơ sở trực giao cho không gian văn bản tương quan. Tương tự, n cột của V đại diện cho một cơ sở trực giao cho không gian vector mở rộng bởi các cột của A . Vì các cột của ma trận TF-IDF tương ứng với các bài báo, nên V sẽ cung cấp một cơ sở trực giao cho không gian từ trong từ điển.

Thư viện Sklearn cung cấp hàm `TruncatedSVD()` để tính truncatedSVD cho ma trận văn bản được tạo ra từ phương pháp TF-IDF.

```
svd = TruncatedSVD(n_components=300, random_state=42)
svd.fit(X_data_tfidf)
```

Hình 10. Source code giảm chiều vector TF-IDF bằng hàm `TruncatedSDV`

- Tham số `n_components` là kích thước mong muốn của dữ liệu đầu ra
- Tham số `random_states` là số lượng kết quả có thể tái sinh qua nhiều lần gọi hàm

III. Xây dựng mô hình

1. Naive-Bayes

1.1. Định lý Bayes

Định lý Bayes cho phép tính xác suất xảy ra của một sự kiện ngẫu nhiên A khi biết sự kiện liên quan B đã xảy ra. Xác suất này được ký hiệu là $P(A|B)$, và đọc là “xác suất của A khi có B”. Đại lượng này được gọi xác suất có điều kiện hay xác suất hậu nghiệm vì nó được rút ra từ giá trị được cho của B hoặc phụ thuộc vào giá trị đó.

Theo định lí Bayes, xác suất xảy ra A khi biết B sẽ phụ thuộc vào 3 yếu tố:

- Xác suất xảy ra A của riêng nó, không liên quan đến B. Ký hiệu là $P(A)$. Đây được gọi là xác suất biên duyên hay xác suất tiên nghiệm, nó là “tiên nghiệm” theo nghĩa rằng nó không quan tâm đến bất kỳ thông tin nào về B.
- Xác suất xảy ra B của riêng nó, không liên quan đến A. Ký hiệu là $P(B)$. Đại lượng này còn gọi là hằng số chuẩn hóa (normalising constant), vì nó luôn giống nhau, không phụ thuộc vào sự kiện A đang muốn biết.
- Xác suất xảy ra B khi biết A xảy ra. Ký hiệu là $P(B|A)$ - “xác suất của B nếu có A”. Đại lượng này gọi là khả năng (likelihood) xảy ra B khi biết A đã xảy ra.

Công thức:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Công thức chỉ ra xác suất của A xảy ra nếu B cũng xảy ra, ta viết là $P(A|B)$. Và nếu ta biết xác suất của B xảy ra khi biết A, ta viết là $P(B|A)$ cũng như xác suất độc lập của A và B.

- $P(A|B)$ là “xác suất của A khi biết B”
- $P(A)$ là “xác suất xảy ra của A”
- $P(B|A)$ là “xác suất của B khi biết A”
- $P(B)$ là “xác suất xảy ra của B”

Như vậy, định lý Bayes sẽ giúp chúng ta tính toán được khả năng xảy ra của một giả thuyết bằng việc thu thập những bằng chứng nhất quán hay không nhất quán với một giả thuyết nào đó. Khi các bằng chứng được thu thập, mức độ tin tưởng đối với một

giả thuyết sẽ thay đổi. Khi có đủ bằng chứng, mức độ tin tưởng này thường trở nên rất cao hay cực thấp, và xác suất khả năng xảy ra của giả thuyết sẽ thay đổi nếu các bằng chứng liên quan đến nó thay đổi.

1.2. Mô hình phân lớp Naive Bayes

Naive Bayes là một thuật toán phân loại cho các vấn đề phân loại nhị phân (hai lớp) và đa lớp. Kỹ thuật này dễ hiểu nhất khi được mô tả bằng các giá trị đầu vào nhị phân hoặc phân loại. Nó là thuật toán được mô hình hoá dựa trên định lý Bayes trong xác suất thống kê.

Xét bài toán có $X = (x_1, x_2, \dots, x_d)$ là vector các đặc trưng và tập classes C (mỗi class ký hiệu là c). Khi đó đẳng thức Bayes trở thành:

$$p(c|X) = p(c|x_1, x_2, \dots, x_d) = \frac{p(x_1|c) p(x_2|c) \dots p(x_d|c)}{p(x_1)p(x_2) \dots p(x_d)} \cdot p(c)$$

(Tức là tính xác suất để đầu ra là class c , biết đầu vào là X)

Từ đó, có thể xác định class của điểm dữ liệu X bằng cách chọn ra class có xác suất cao nhất trong C :

$$c = \arg \max_{c \in C} p(c|X) \quad (c \in C)$$

Biểu thức này thường khó được tính trực tiếp. Thay vào đó, quy tắc Bayes thường được sử dụng như sau:

$$c = \arg \max p(c|X) = \arg \max \frac{p(X|c)}{p(X)} \cdot p(c) = \arg \max p(X|c)p(c)$$

(Do mẫu số $p(X)$ không phụ thuộc vào c)

Trong đó:

- $p(c)$ có thể được hiểu là xác suất để một điểm rơi vào class c
- $p(x|c)$ là phân phối của những điểm dữ liệu trong class c , nó khá khó tính toán bởi vì X là một biến ngẫu nhiên nhiều chiều và đòi hỏi sử dụng rất nhiều dữ liệu training mới có thể xây dựng được phân phối đó. Để giúp cho việc tính toán trở nên dễ dàng hơn người ta hay giả sử một cách đơn giản nhất là các thành phần của biến ngẫu nhiên X là **độc lập với nhau**, nếu biết c :

$$p(X|c) = p(x_1, x_2, \dots, x_d|c) = \prod_{i=1}^d p(x_i|c)$$

Trên thực tế, việc các thành phần độc lập với nhau ít khi xảy ra. Tuy nhiên, giả thiết “ngây thơ” này có thể mang lại kết quả tốt bất ngờ. Giả thiết về sự độc lập của các chiều dữ liệu này được gọi là **Naive Bayes**. Cách xác định class của dữ liệu dựa trên giả thiết này có tên là **Naive Bayes Classifier (NBC)**.

Bởi vì “ngây thơ” nên phương pháp này có tốc độ training và test đều rất nhanh, giúp cho các bài toán được giải quyết nhanh và dễ dàng hơn.

Ở bước **training**, các phân phối $p(c)$ và $p(x_i|c)$, $i=1, \dots, d$ sẽ được xác định dựa vào training data, bằng phương pháp Maximum Likelihood Estimation (MLE) hoặc Maximum A Posteriori (MAP), chủ yếu là dùng MLE.

Ở bước **test**, với một dữ liệu mới X nào đó, class của nó sẽ được xác định bởi:

$$c = \arg \max_{c \in C} p(c) \prod_{i=1}^d p(x_i|c) \quad (c \in C)$$

Khi d lớn và các xác suất nhỏ, biểu thức ở vế phải của biểu thức này sẽ là một số rất nhỏ, khi tính toán có thể gặp sai số. Để giải quyết việc này, nó thường được viết lại dưới dạng tương đương bằng cách lấy log của vế phải

$$c = \arg \max_{c \in C} p(c) = \log(p(c)) + \sum_{i=1}^d \log(p(x_i|c))$$

Do log là một hàm đồng biến trên tập các số dương nên việc này không ảnh hưởng tới kết quả.

Mặc dù giả thiết mà Naive Bayes Classifiers sử dụng là quá phi thực tế, chúng vẫn hoạt động khá hiệu quả trong nhiều bài toán thực tế, đặc biệt là trong các bài toán phân loại văn bản, ví dụ như lọc tin nhắn rác hay lọc email spam.

Thuật toán này là một thuật toán mạnh mẽ được ứng dụng trong các bài toán:

- Dự đoán với thời gian thực
- Phân loại Văn bản/ Lọc thư rác/ Phân tích tình cảm
- Hệ thống Gợi ý (Recommendation)

Naive Bayes chia thành một số kiểu mô hình cơ bản: Multinomial Naive Bayes, Bernoulli Naive Bayes, Gaussian Naive Bayes,... Trong đó, mô hình **Multinomial Naive Bayes** chủ yếu được sử dụng trong bài toán phân loại văn bản.

Ưu điểm

- Dễ dàng và nhanh chóng khi dự đoán lớp của tập dữ liệu thử nghiệm. Nó cũng hoạt động hiệu quả đối với dự đoán nhiều lớp
- Khi giả định giữ độc lập, bộ phân loại Naive Bayes hoạt động hiệu quả hơn so với những thuật toán khác vì cần ít dữ liệu đào tạo hơn.
- Nó hoạt động tốt khi có sự chênh lệch số lượng giữa các lớp phân loại.

Nhược điểm

- Nếu biến phân loại có một danh mục (trong tập dữ liệu thử nghiệm), không được quan sát trong tập dữ liệu huấn luyện, thì mô hình sẽ chỉ định xác suất 0 (không) và làm cho dự đoán sẽ không còn chính xác. Trường hợp này còn được gọi là **“Tần số không”**. Để giải quyết vấn đề này, chúng ta có thể sử dụng kỹ thuật Smooth. Một trong những kỹ thuật làm mịn đơn giản nhất được gọi là ước lượng Laplace (sẽ được phân tích kỹ hơn ở phần sau – 1.3).
- Mặt khác, Naive Bayes cũng được biết đến như một công cụ ước lượng tồi, vì vậy kết quả xác suất từ dự đoán *proba* không được coi trọng quá.
- Một hạn chế khác của Naive Bayes là giả định rằng các yếu tố dự đoán độc lập. Tuy nhiên trong thực tế hầu như không thể có được một tập hợp các yếu tố dự đoán hoàn toàn độc lập với nhau.

1.3. Mô hình Multinomial Naive Bayes

Mô hình Multinomial Naive Bayes chủ yếu được sử dụng trong phân loại văn bản, nghĩa là một tài liệu cụ thể thuộc về danh mục nào như Thể thao, Chính trị, giáo dục, v.v.

Giả sử ta có văn bản X được biểu diễn thành vector và tập classes C (chứa các class c)

$$X = (x_1, x_2, \dots, x_d) \quad (x_i \geq 0)$$

Lúc này, mỗi văn bản được biểu diễn bởi một vector có độ dài d - số từ trong từ điển. Giá trị của thành phần thứ i trong mỗi vector chính là số lần từ thứ i xuất hiện trong văn bản đó.

Khi đó, $p(x_i|c)$ tỉ lệ với tần suất từ thứ i (hay feature thứ i cho trường hợp tổng quát) xuất hiện trong các văn bản của class c . Công thức tính giá trị này:

$$\lambda_{ci} = p(x_i|c) = \frac{N_{ci}}{N_c}$$

Trong đó:

- N_{ci} là tổng số lần từ x_i xuất hiện trong các văn bản của class c , nó được tính là tổng của tất cả các thành phần thứ i của các feature vectors ứng với class c .
- N_c là tổng số từ (kể cả lặp) xuất hiện trong class c (Tức là tổng độ dài của toàn bộ các văn bản thuộc vào class c). Có thể suy ra:

$$N_c = \sum_{i=1}^d N_{ci} \Rightarrow \sum_{i=1}^d \lambda_{ci} = 1$$

Hạn chế: Cách tính này có một hạn chế là nếu như có một từ mới x_k chưa bao giờ xuất hiện trong class c thì $N_{ck} = 0$, dẫn đến $p(x_k|C) = 0 \Rightarrow$ Kết quả sẽ không chính xác

Giải quyết: dùng *Laplace smoothing*, tức là cộng vào cả tử và mẫu giá trị luôn khác 0:

$$\lambda'_{ci} = p'(x_i|C) = \frac{N_{ci} + \alpha}{N_c + d\alpha}$$

Trong đó α là một số dương, thường bằng 1, để tránh tử số bị bằng 0.

Mẫu số được cộng với $d\alpha$ để đảm bảo tổng xác suất sau bằng 1

$$\sum_{i=1}^d \lambda'_{ci} = 1$$

Vấn đề: Cần tìm α để $\lambda'_{ci} - \lambda_{ci}$ min. Như vậy thì sai số mới tối thiểu và accuracy (độ chính xác) có thể lớn nhất.

Ta sẽ dùng vòng lặp for cho các giá trị α để tìm α tốt nhất


```

alphas = [1.0e-10, 0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 10.0]

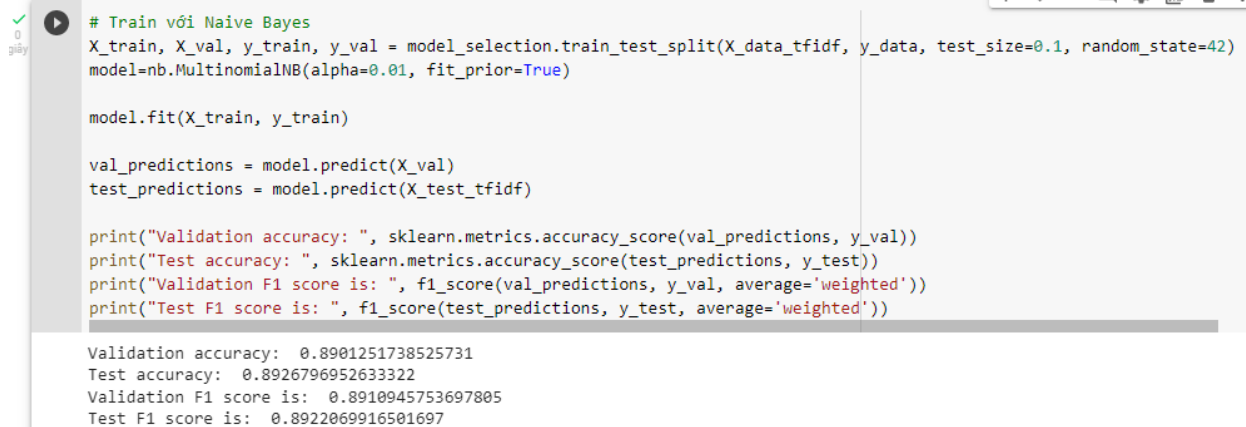
for i in range (len(alphas)):
    model=nb.MultinomialNB(alpha = alphas[i], fit_prior=False)
    model.fit(X_train, y_train)
    val_predictions = model.predict(X_val)
    test_predictions = model.predict(X_test_tfidf)

    print("alpha =", alphas[i], ":")
    print("\t Validation accuracy:", sklearn.metrics.accuracy_score(val_predictions, y_val))
    print("\t Test accuracy:", sklearn.metrics.accuracy_score(test_predictions, y_test))

```

Hình 11. Source code tìm α tốt nhất để accuracy lớn nhất

1.4. Triển khai mô hình Multinomial Naive Bayes



```

# Train với Naive Bayes
X_train, X_val, y_train, y_val = model_selection.train_test_split(X_data_tfidf, y_data, test_size=0.1, random_state=42)
model=nb.MultinomialNB(alpha=0.01, fit_prior=True)

model.fit(X_train, y_train)

val_predictions = model.predict(X_val)
test_predictions = model.predict(X_test_tfidf)

print("Validation accuracy: ", sklearn.metrics.accuracy_score(val_predictions, y_val))
print("Test accuracy: ", sklearn.metrics.accuracy_score(test_predictions, y_test))
print("Validation F1 score is: ", f1_score(val_predictions, y_val, average='weighted'))
print("Test F1 score is: ", f1_score(test_predictions, y_test, average='weighted'))

Validation accuracy: 0.8901251738525731
Test accuracy: 0.8926796952633322
Validation F1 score is: 0.8910945753697805
Test F1 score is: 0.8922069916501697

```

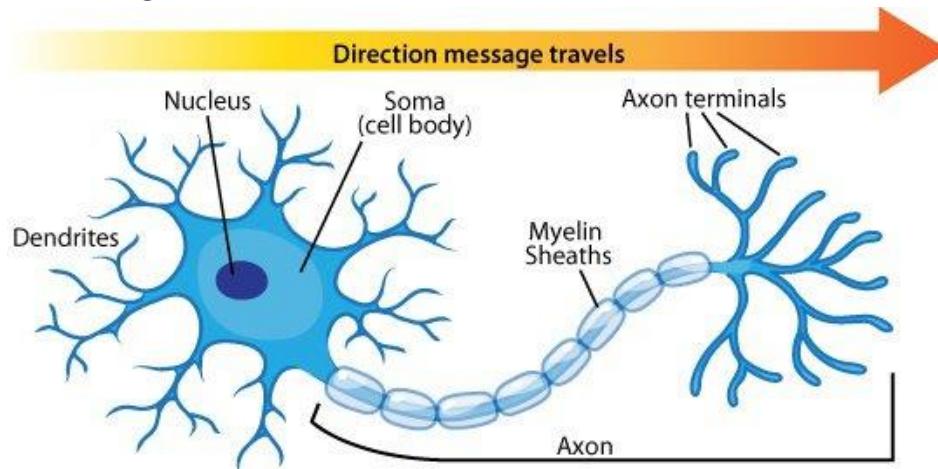
Hình 12. Source code triển khai mô hình Multinomial Naive Bayes

2. Mạng nơ-ron nhân tạo

2.1. Mạng nơ-ron nhân tạo (Neural Network) là gì?

Mạng nơ-ron nhân tạo là một chuỗi những thuật toán được đưa ra để tìm kiếm các mối quan hệ cơ bản trong tập hợp các dữ liệu. Thông qua việc bắt chước cách thức hoạt động từ não bộ con người. Nói cách khác, mạng nơ-ron nhân tạo được xem là hệ thống tính toán lấy cảm hứng từ sự hoạt động của các nơ-ron trong hệ thần kinh.

2.2. Hoạt động của các nơ-ron



Hình 13. Mô hình giải phẫu thần kinh nơ-ron

Nơ-ron là đơn vị cơ bản cấu tạo hệ thống thần kinh và là một phần quan trọng nhất của não. Não chúng ta gồm khoảng 10 triệu nơ-ron và mỗi nơ-ron liên kết với 10.000 nơ-ron khác.

Ở mỗi nơ-ron có phần thân (soma) chứa nhân, các tín hiệu đầu vào qua sợi nhánh (dendrites) và các tín hiệu đầu ra qua sợi trục (axon) kết nối với các nơ-ron khác. Hiểu đơn giản mỗi nơ-ron nhận dữ liệu đầu vào qua sợi nhánh và truyền dữ liệu đầu ra qua sợi trục, đến các sợi nhánh của các nơ-ron khác.

Mỗi nơ-ron nhận xung điện từ các nơ-ron khác qua sợi nhánh. Nếu các xung điện này đủ lớn để kích hoạt nơ-ron, thì tín hiệu này đi qua sợi trục đến các sợi nhánh của các nơ-ron khác.

Tuy nhiên Mạng nơ-ron nhân tạo chỉ là lấy cảm hứng từ não bộ và cách nó hoạt động, chứ không phải bắt chước toàn bộ các chức năng của nó. Việc chính của chúng ta là dùng mô hình này để giải quyết các bài toán chúng ta cần.

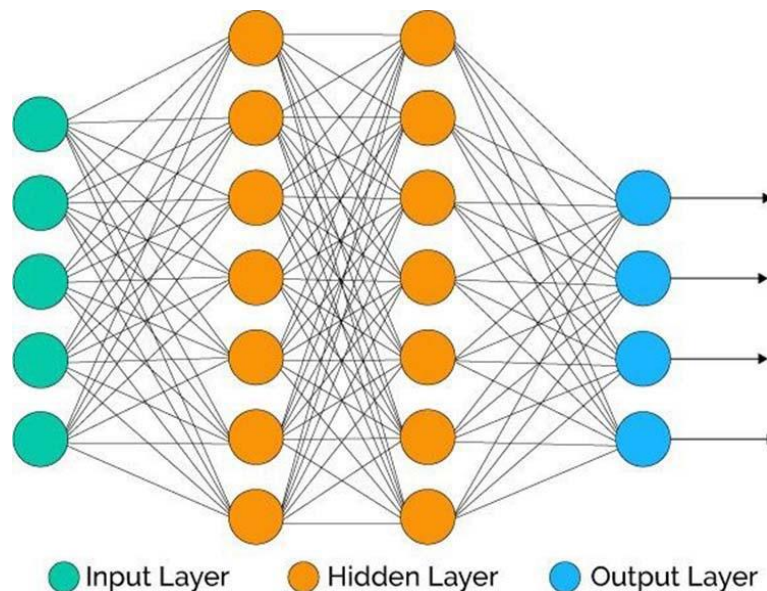
2.3. Mô hình Mạng nơ-ron nhân tạo tổng quát

Mạng Nơ-ron nhân tạo là sự kết hợp của những tầng perceptron hay còn gọi là perceptron đa tầng. Và mỗi một mạng nơ-ron nhân tạo thường bao gồm 3 kiểu tầng là:

- Tầng vào(input layer): Tầng này nằm bên trái cùng của mạng, thể hiện cho các đầu vào của mạng.

- Tầng ra (output layer): Là tầng bên phải cùng và nó thể hiện cho những đầu ra của mạng.
- Tầng ẩn (hidden layer): Tầng này nằm giữa tầng vào và tầng ra nó thể hiện cho quá trình suy luận logic của mạng.

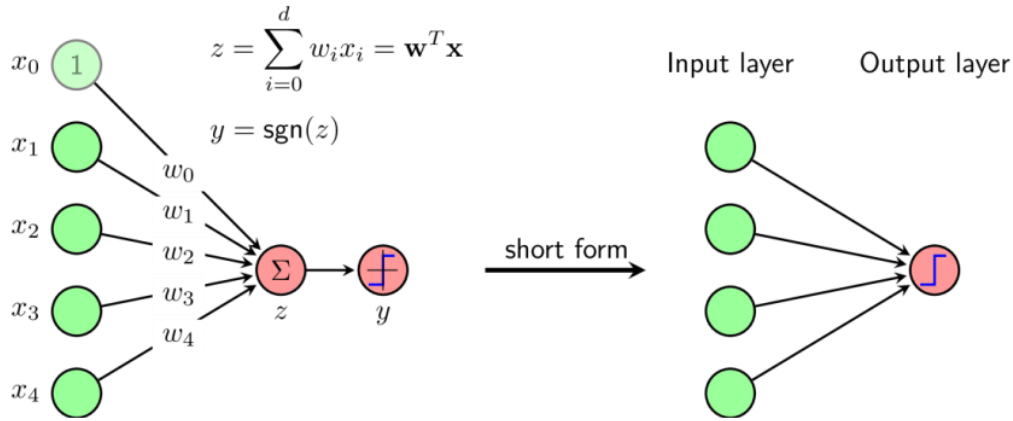
Mỗi mô hình luôn có một tầng vào, một tầng ra, có thể có hoặc không các tầng ẩn. Tổng số tầng trong mô hình được quy ước là số tầng - 1 (Không tính tầng vào).



Hình 14. Kiến trúc mạng nơ-ron nhân tạo

Mỗi node trong tầng ẩn và tầng ra:

- Liên kết với tất cả các node ở tầng trước đó với các hệ số trọng số(weight) riêng.
- Mỗi node có một hệ số sai khác(bias) riêng.
- Diễn ra hai bước: tính tổng tuyến tính(linear) và áp dụng hàm kích hoạt(activation function).



Hình 15. Hình ảnh minh họa cho hoạt động của các node trong mạng nơ-ron

Trong đó, hàm $\text{sgn}(z)$ được hiểu là hàm kích hoạt (activation function).

2.4. Hàm kích hoạt (activation function)

Hàm kích hoạt là những hàm phi tuyến được áp dụng vào đầu ra của các nơ-ron trong tầng ẩn của một mô hình mạng, và được sử dụng làm input data cho tầng tiếp theo.

Trước tiên, ta cần hiểu lý do chính làm các mô hình mạng neural nổi bật hơn so với các mô hình học máy là khả năng giải quyết được những vấn đề về tính chất phi tuyến của dữ liệu (non-linear separable data). Những tầng mạng nằm giữa tầng đầu vào và đầu ra của một mạng nơ-ron được gọi là tầng ẩn. Những tầng ẩn này giữ nhiệm vụ giải quyết những quan hệ phi tuyến phức tạp giữa các đặc điểm của dữ liệu và kết quả đầu ra của mô hình nhờ những hàm "phi tuyến hóa" thường được biến đến với tên hàm kích hoạt.

Nếu chỉ áp dụng một hàm tuyến tính vào đầu ra của mỗi nơ-ron, sẽ dẫn đến việc mô hình mạng nơ-ron không khác gì mô hình học máy thông thường (điển hình là mô hình hồi quy tuyến tính). Vì phép biến đổi không có tính chất phi tuyến, việc này không khác gì chúng ta thêm một tầng ẩn nữa vì phép biến đổi cũng chỉ đơn thuần là nhân đầu ra với các trọng số. Với chỉ những phép tính đơn thuần như vậy, trên thực tế mạng nơ-ron sẽ không thể phát hiện ra những quan hệ phức tạp của dữ liệu (ví dụ như: dự đoán chứng khoán, các bài toán xử lý ảnh hay các bài toán phát hiện ngữ nghĩa của các câu trong văn bản). Nói cách khác nếu không có các hàm kích hoạt, khả năng dự đoán của mạng nơ-ron sẽ bị giới hạn và giảm đi rất nhiều, sự kết hợp của các hàm kích hoạt giữa các tầng ẩn là để giúp mô hình học được các quan hệ phi tuyến phức tạp tiềm ẩn trong dữ liệu.

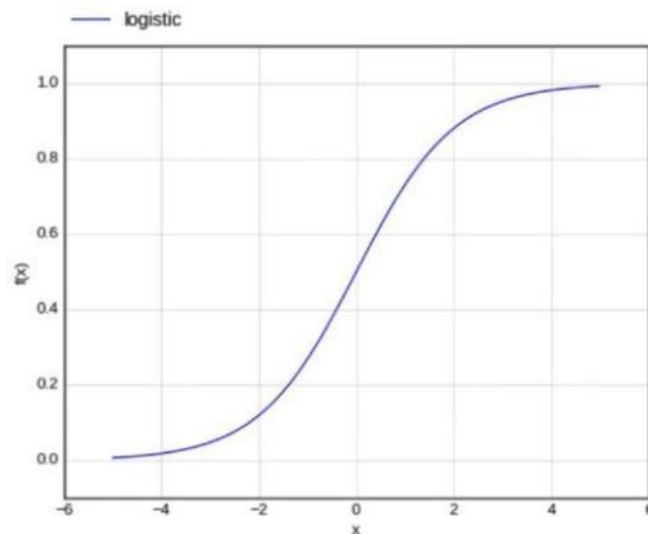
Các hàm kích hoạt thường gặp:

a. Hàm sigmoid

- Công thức:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Đồ thị hàm sigmoid:



Hình 16. Đồ thị hàm Sigmoid

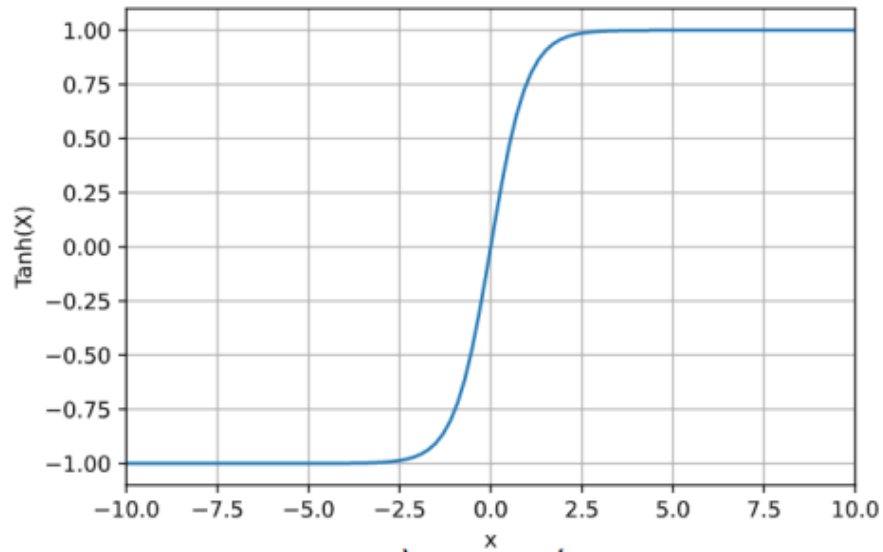
Hàm Sigmoid nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng (0;1) (xem đồ thị phía trên). Đầu vào là số thực âm rất nhỏ sẽ cho đầu ra tiệm cận với 0, ngược lại, nếu đầu vào là một số thực dương lớn sẽ cho đầu ra là một số tiệm cận với 1.

b. Hàm tanh

- Công thức:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Đồ thị hàm tanh



Hình 17. Đồ thị hàm Tanh

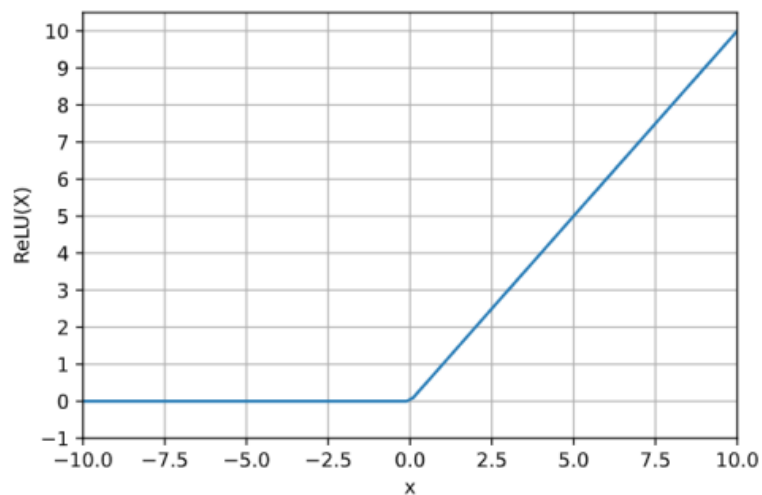
Hàm tanh nhận đầu vào là một số thực và chuyển thành một giá trị trong khoảng $(-1; 1)$. Cũng như sigmoid, hàm tanh bị bão hoà ở 2 đầu (gradient thay đổi rất ít ở 2 đầu). Tuy nhiên hàm Tanh lại đối xứng qua 0 nên khắc phục được một nhược điểm của Sigmoid.

c. Hàm ReLU

- Công thức

$$f(x) = \max(0, x)$$

- Đồ thị hàm ReLU



Hình 18. Đồ thị hàm ReLU

Hàm ReLU đang được sử dụng khá nhiều trong những năm gần đây khi huấn luyện các mạng neuron. ReLU đơn giản lọc các giá trị < 0 . Nhìn vào công thức chúng ta dễ dàng hiểu được cách hoạt động của nó. Một số ưu điểm khá vượt trội của nó so với Sigmoid và Tanh.

3. K-Nearest-Neighbor

3.1. *K-Nearest-Neighbor là gì?*

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này *không học* một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại [lazy learning](#)), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-nearest neighbor có thể áp dụng được vào cả hai loại của bài toán Supervised learning là [Classification](#) và [Regression](#). KNN còn được gọi là một thuật toán [Instance-based](#) hay [Memory-based learning](#).

Với kNN, trong bài toán Classification, label của một điểm dữ liệu mới (hay kết quả của câu hỏi trong bài thi) được suy ra trực tiếp từ K điểm dữ liệu gần nhất trong training set. Label của một test data có thể được quyết định bằng major voting (bầu chọn theo số phiếu) giữa các điểm gần nhất, hoặc nó có thể được suy ra bằng cách đánh trọng số khác nhau cho mỗi trong các điểm gần nhất đó rồi suy ra label.

Trong bài toán Regression, đầu ra của một điểm dữ liệu sẽ bằng chính đầu ra của điểm dữ liệu đã biết gần nhất (trong trường hợp $K=1$), hoặc là trung bình có trọng số của đầu ra của những điểm gần nhất, hoặc bằng một mối quan hệ dựa trên khoảng cách tới các điểm gần nhất đó.

Một cách ngắn gọn, KNN là thuật toán đi tìm đầu ra của một điểm dữ liệu mới bằng cách *chỉ* dựa trên thông tin của K điểm dữ liệu trong training set gần nó nhất (K-lân cận), *không quan tâm đến việc có một vài điểm dữ liệu trong những điểm gần nhất này là nhiều*.

3.2. *Thí nghiệm để tìm tham số k tốt nhất*

Chọn k trong tập sau: {3,5,7,9,11,15,19,25,31}

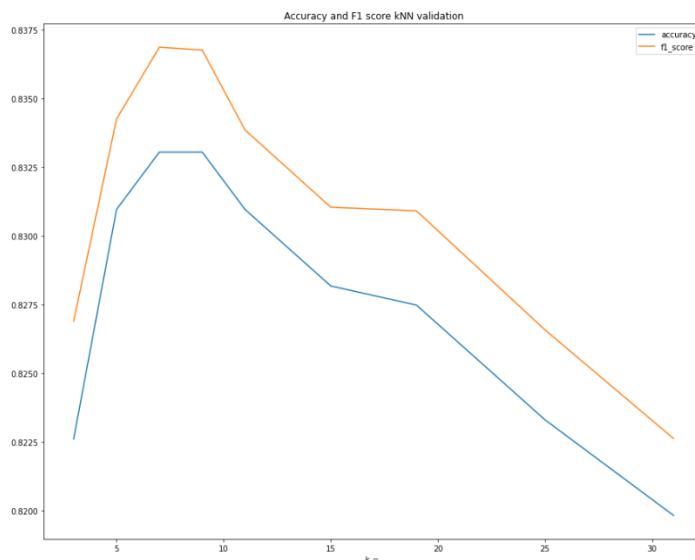
Xây dựng mô hình kNN đối với từng k trong tập tương ứng

```

from sklearn.neighbors import KNeighborsClassifier
arr = [3,5,7,9,11,15,19,25,31]
for k in arr:
    knn = KNeighborsClassifier(n_neighbors=k)
    train_model(classifier=knn, X_data=X_data_tfidf_svd, y_data=np.array(y_data_n),
                  X_test=X_test_tfidf_svd, y_test=np.array(y_test_n), is_neuralnet=False)

```

Kết quả thử nghiệm như sau:



Hình 19. Kết quả thử nghiệm mô hình kNN

Như vậy, từ kết quả thử nghiệm, tham số $k=7$ được cho là tốt nhất cho mô hình.

4. Tiêu chí đánh giá

Vì dữ liệu trong mô hình phân bố không đồng đều, nên việc sử dụng accuracy để đánh giá có thể đưa ra những kết quả không chính xác. Ví dụ với bài toán phân loại 2 nhãn (binary classification) nhưng nhãn thứ nhất có kích thước áp đảo là 99% bộ dữ liệu trong khi nhãn thứ hai chỉ chiếm 1% tập dữ liệu. Nếu mô hình dự đoán toàn bộ các điểm dữ liệu là thuộc nhãn số một thì nó đã đạt được 99% độ chính xác trong khi thực chất nó không hề học được gì và dự đoán sai hết những phần tử thuộc nhãn thứ hai. Vì vậy, việc sử dụng accuracy trong trường hợp này có thể không hiệu quả và không công bằng giữa các nhãn.

$$\frac{2}{F1} = \frac{1}{precision} + \frac{1}{recall}$$

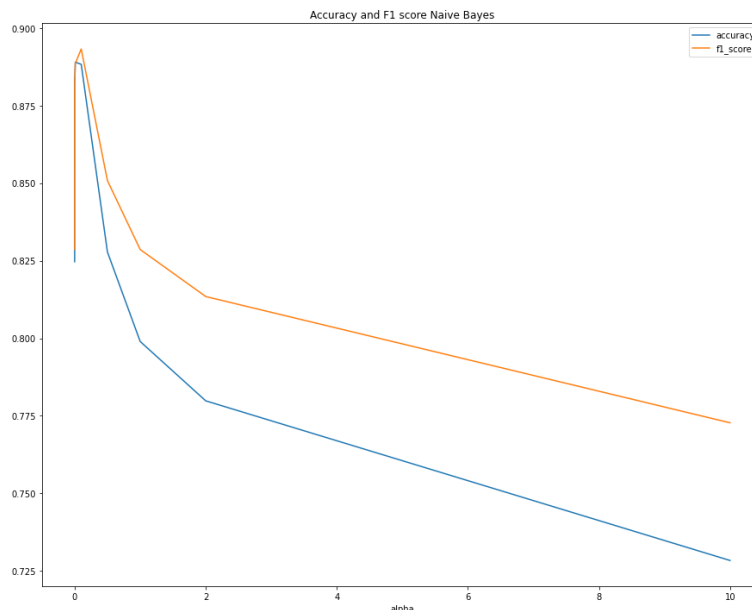
Với độ đo f1 thì lại khác, f1 là trung bình điều hòa giữa precision và recall, tức là nó sẽ gần hơn với giá trị nhỏ hơn trong 2 giá trị trên. Với đặc điểm này, việc đánh giá sử dụng độ đo f1 sẽ cho kết quả chính xác hơn. Giả sử trong tình huống trên, nếu sử dụng f1 score, nó sẽ cho điểm rất thấp. Điều này sẽ buộc mô hình phải cố gắng phân loại cả hai nhãn thay vì thiên vị hoàn toàn cho nhãn có số lượng áp đảo hơn. Nói tóm lại, nhóm em sẽ sử dụng độ đo chính là f1 score, và bên cạnh đó, nhóm cũng đo accuracy score, train time, valid time để có thể có được một góc nhìn toàn vẹn hơn trong quá trình đánh giá.

D. Kết quả thí nghiệm

I. Thử nghiệm mô hình Multinomial Naive Bayes

Xét tập $\alpha = \{ 1.0e-10, 0.0001, 0.001, 0.01, 0.1, 0.5, 1.0, 2.0, 10.0 \}$

Thử nghiệm mô hình với từng alpha, ta thu được kết quả sau đây



Hình 20. Kết quả thu được với mô hình Naive Bayes trên tập validation

Như vậy, chúng em nhận thấy tham số $\alpha = 0.01$ là tốt nhất đối với bộ dữ liệu

Tiến hành chạy thử mô hình Multinomial Naive Bayes cho ra kết quả sau đây:

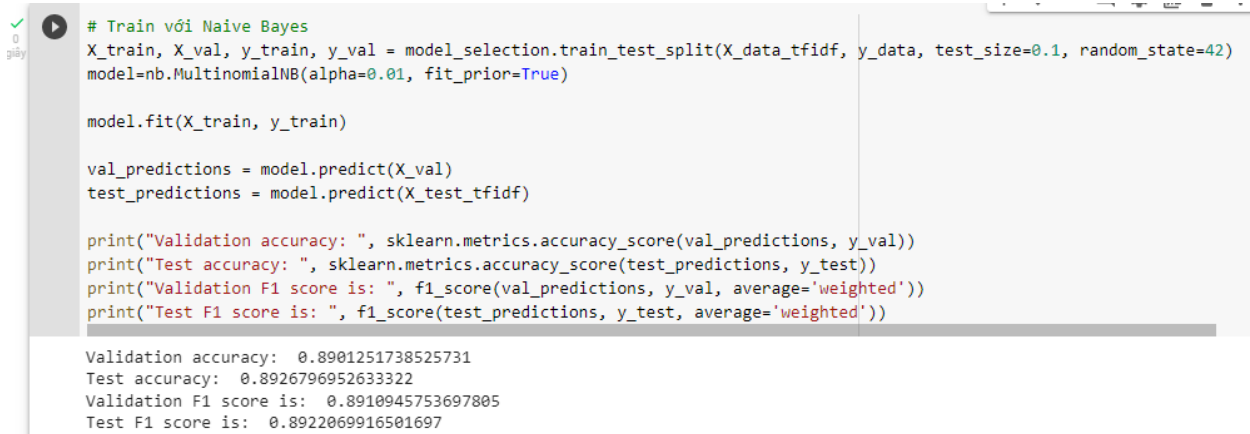
- Với $\alpha = 0.01$ (alpha = 0.01):

Validation accuracy: 0.8901251738525731

Test accuracy: 0.8926796952633322

Validation F1 score is: 0.8910945753697805

Test F1 score is: 0.8922069916501697



```
# Train với Naive Bayes
X_train, X_val, y_train, y_val = model_selection.train_test_split(X_data_tfidf, y_data, test_size=0.1, random_state=42)
model=nb.MultinomialNB(alpha=0.01, fit_prior=True)

model.fit(X_train, y_train)

val_predictions = model.predict(X_val)
test_predictions = model.predict(X_test_tfidf)

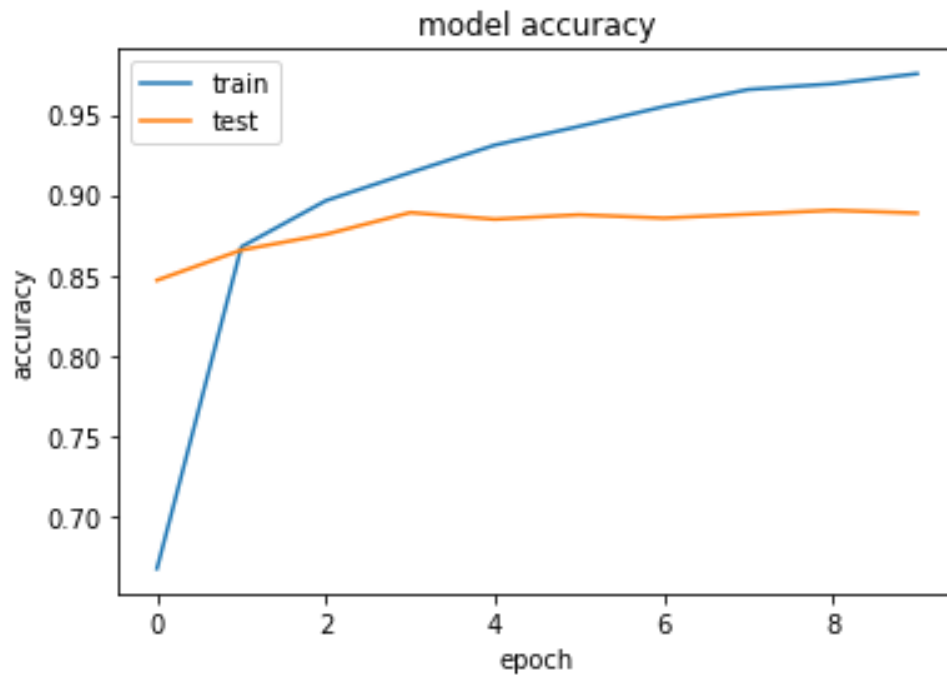
print("Validation accuracy: ", sklearn.metrics.accuracy_score(val_predictions, y_val))
print("Test accuracy: ", sklearn.metrics.accuracy_score(test_predictions, y_test))
print("Validation F1 score is: ", f1_score(val_predictions, y_val, average='weighted'))
print("Test F1 score is: ", f1_score(test_predictions, y_test, average='weighted'))

Validation accuracy: 0.8901251738525731
Test accuracy: 0.8926796952633322
Validation F1 score is: 0.8910945753697805
Test F1 score is: 0.8922069916501697
```

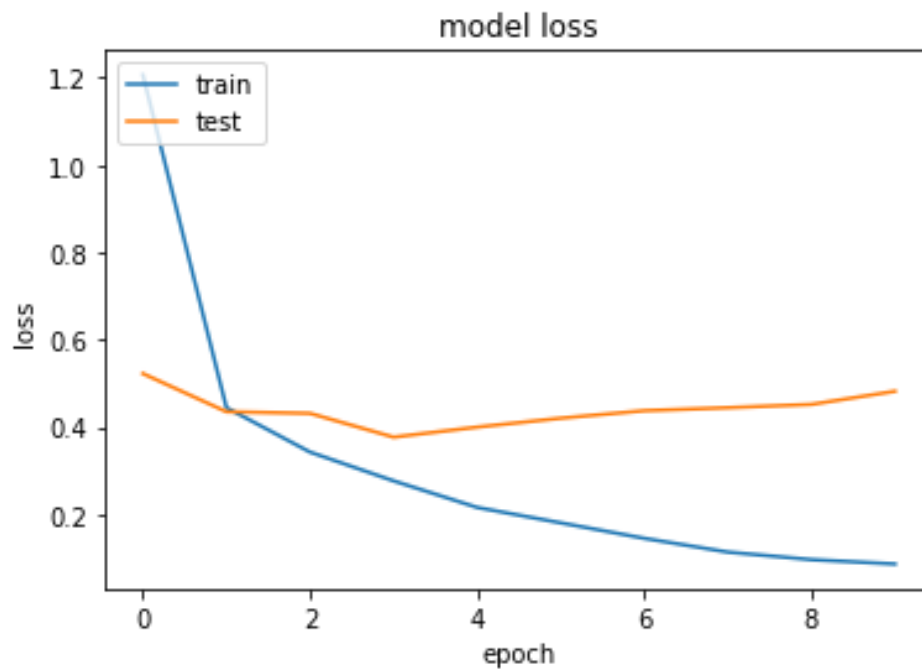
Hình 21. Kết quả thử nghiệm mô hình Multinomial Naive Bayes

II. Thử nghiệm mô hình Deep Neural Network

Tiến hành chạy thử mô hình Deep Neural Network cho ra kết quả sau đây:



Hình 22. Đồ thị accuracy theo từng epoch



Hình 23. Đồ thị loss theo từng epoch

Các chỉ số đánh giá cho mô hình mạng nơ ron:

```
Validation accuracy: 0.888695652173913
Test accuracy: 0.8801755548194766
Validation F1 score: 0.8899426595180661
Test F1 score: 0.8796588779275226
```

III. Thử nghiệm với mô hình K-Nearest-Neighbor

Tiến hành thử nghiệm với mô hình kNN, tham số $k=7$

```
def train_model(classifier, X_data, y_data, X_test, y_test, is_neuralnet=False, n_epochs=10):
    X_train, X_val, y_train, y_val = train_test_split(X_data, y_data, test_size=0.2, random_state=42)
    classifier.fit(X_train, y_train)
    train_predictions = classifier.predict(X_train)
    val_predictions = classifier.predict(X_val)
    test_predictions = classifier.predict(X_test)
    print(test_predictions)
    print("Validation accuracy: ", accuracy_score(val_predictions, y_val))
    print("Test accuracy: ", accuracy_score(test_predictions, y_test))
    print("Validation F1 score: ", f1_score(val_predictions, y_val, average='weighted'))
    print("Test F1 score: ", f1_score(test_predictions, y_test, average='weighted'))
    return classifier
```

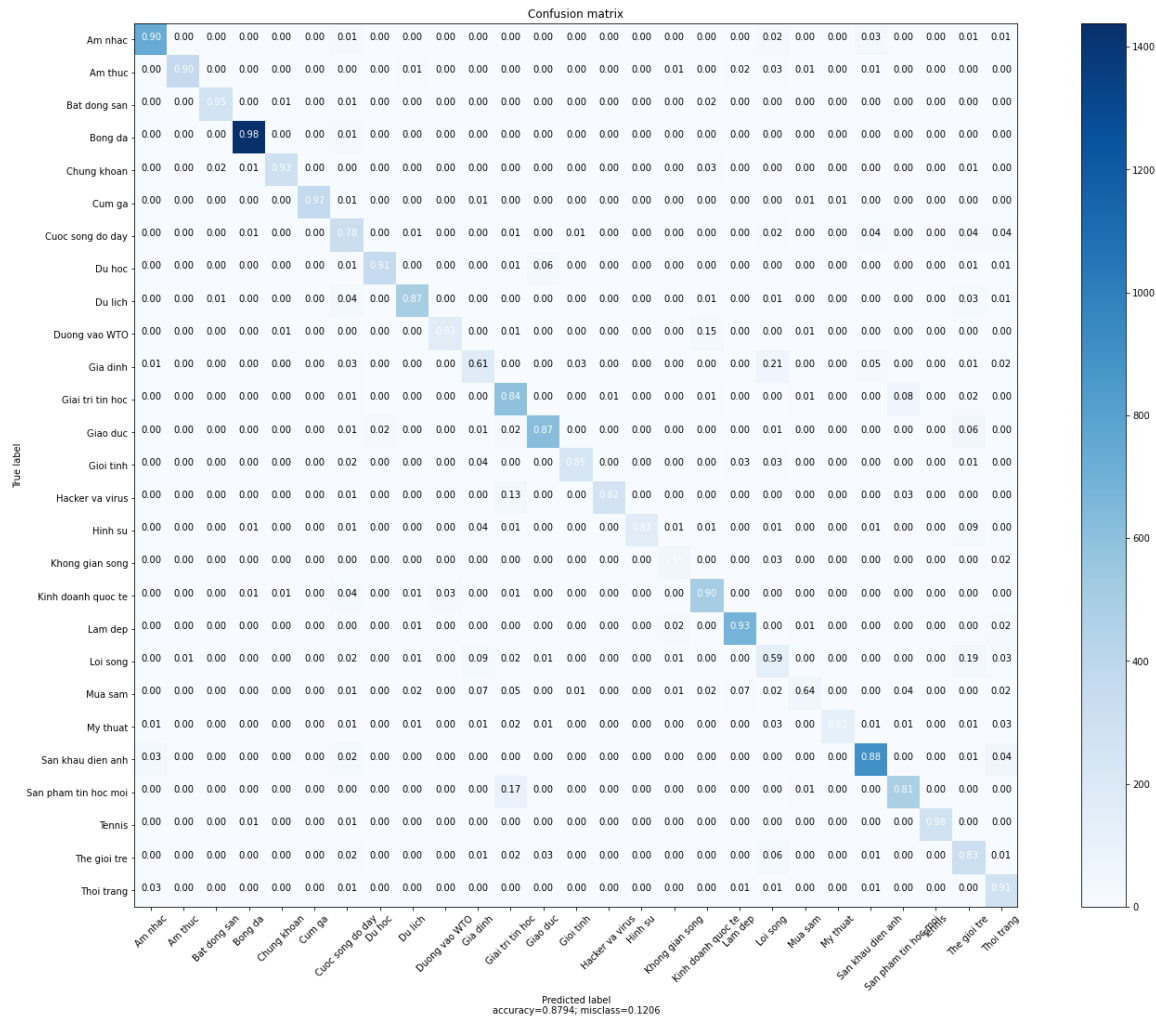
Kết quả thu được đối với mô hình kNN

```
Validation accuracy: 0.8330434782608696
Test accuracy: 0.8176548526001988
Validation F1 score: 0.8368607017801911
Test F1 score: 0.8162262508075233
```

IV. Confusion matrix

Như vậy, sau khi tiến hành thử nghiệm, mô hình tốt nhất mà nhóm đề xuất là mô hình Naïve Bayes với tham số $\alpha=0.01$

Ta thu được ma trận nhầm lẫn sau:



Hình 24. Ma trận nhầm lẫn của mô hình Naive Bayes

E. Kết luận và cải tiến

Qua quá trình thử nghiệm và đánh giá, nhóm đã đạt được một số kết quả nhất định.

Theo đó, mô hình Naive Bayes là mô hình cho được kết quả tốt nhất. Tuy nhiên, theo như confusion matrix (ma trận nhầm lẫn), một số chủ đề giống nhau vẫn đang bị phân loại sai. Điển hình như nhãn “Sản phẩm tin học” với “Giải trí tin học”. Nhìn chung, các chủ đề này cũng khá khó để phân biệt. Ngoài ra, các nhãn như “Lối sống”, “Gia đình”, “Mua sắm” đang có độ chính xác thấp. Theo đó, trong quá trình cải tiến sau này, nhóm dự định sẽ cải tiến mô hình để dự đoán các nhãn này chính xác hơn. Mô hình cần được tối ưu hơn phân tiền xử lý văn bản và trích chọn đặc trưng vì có vẻ như các chủ đề trên không có các từ khóa chủ đạo (dễ bị nhầm lẫn sang chủ đề khác), nên phương pháp tf-idf không thực sự hữu dụng, hoặc mô hình cần được cải tiến bằng các thuật toán khác của Học sâu.

Danh mục tài liệu tham khảo

1. Bài viết *Singular Value Decomposition* trên blog <https://machinelearningcoban.com/>.
2. Bài viết *Naive Bayes Classifier* trên blog <https://machinelearningcoban.com/>.
3. Bài viết *Phân loại văn bản tiếng Việt sử dụng mBaiachine learning* trên blog <https://blog.luyencode.net>.
4. Bài viết *Phân loại văn bản tự động bằng machine learning như thế nào*.
Phát hành ngày 11/10/2018, Trên website <https://viblo.asia/>.
5. Đồ án tốt nghiệp *Phân loại từ tiếng việt sử dụng mô hình CRFs* của tác giả Nguyễn Trung Kiên – trường đại học Công Nghệ ĐHQGHN.
6. Bài viết *Tokenization là gì? Các kỹ thuật tách từ trong xử lý ngôn ngữ tự nhiên*. Phát hành ngày 18/08/2021, đăng trên website <https://vinbigdata.com/>
7. Bài viết *Tổng quan về thuật toán phân lớp Naive Bayes Classification (NBC)*. Phát hành ngày 21/07/2018, từ <http://hoctructuyen123.net/tong-quan-ve-thuat-toan-phan-lop-naive-bayes-classification-nbc/>
8. Bài viết *Phân loại văn bản tự động bằng Machine Learning như thế nào?* của tác giả Nguyễn Thành Hậu. Phát hành ngày 11/10/2018, từ <https://viblo.asia/p/phan-loai-van-ban-tu-dong-bang-machine-learning-nhu-the-nao-4P856Pa1ZY3>
9. Bài viết *Phân loại văn bản tự động bằng Machine Learning như thế nào? (Phần 2)* của tác giả Nguyễn Thành Hậu. Phát hành ngày 09/11/2018, từ <https://viblo.asia/p/phan-loai-van-ban-tu-dong-bang-machine-learning-nhu-the-nao-phan-2-4P856PqBZY3>
10. Bài giảng Convolutional Neural Network. (2019). Phát hành tháng 07/2020, từ <https://nttuan8.com/bai-6-convolutional-neural-network/>
11. Charu C. Aggarwal. (2018). Neural Networks and Deep Learning: A Textbook, first Edition
12. Đồ thị các hàm kích hoạt phổ biến. (2019). Phát hành tháng 09/2019, từ <https://vietanh.dev/blog/2019-09-23-cac-ham-kich-hoat-activation-function-trong-neural-networks>