TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
KHOA KHOA HỌC MÁY TÍNH

REPORT CASE STUDY 2

Bài toán: "News Headlines Dataset For Sarcasm Detection – High quality dataset for the task of Sarcasm Detection" (Phát hiện tin bài châm biếm – Bộ dữ liệu chất lượng cao từ các trang tin chuyên nghiệp)

Giáo viên hướng dẫn: Thầy Phạm Nguyễn Trường An

Nhóm: 17520943 – Trần Nguyễn Hồng Quân (người nộp)
17520964 – Nguyễn Đình Quyết
17521180 – Đặng Xuân Trường

Mục lục

CHƯƠNG 1: GIỚI THIỆU CHUNG

"Phát hiện tin bài châm biếm" phát sinh từ bài toán gốc "Phát hiện tin giả mạo" – thường dùng cho mạng xã hội (vốn tin bài trên mạng xã hội viết với văn phong không chính thống). Đây là bài toán cũng đang khá hot, được nhiều người quan tâm (đặc biệt là Facebook đang gặp khó khăn trong vấn đề lọc tin).

1.1. Input:

Thông tin về Dataset:

- Được crawl (thu thập) từ 2 trang web:
  - **TheOnion** (website: https://www.theonion.com/): toàn đăng những tin châm biếm.
  - **HuffPost** (website: https://www.huffpost.com/): bao gồm những tin chính thống.
- Chỉ crawl Headline (Tiêu đề) của các bài viết.
- Thống kê: dataset bao gồm:

Statistic/Dataset	Headlines
# Records	26,709
# Sarcastic records	11,725
# Non–sarcastic records	14,984

- Nguồn dữ liệu:

@dataset{dataset},

author = {Misra, Rishabh},

year = {2018},

month = {06},

pages = {},

title = {News Headlines Dataset For Sarcasm Detection},

doi = {10.13140/RG.2.2.16182.40004}

}

Mỗi record bao gồm 3 thông tin sau:

- `article_link`: đường dẫn liên kết tới bài viết. Dùng để crawl thêm dữ liệu nếu cần.
- `headline`: tiêu đề của bài viết.
- `is_sarcastic`: là label của record, bằng 1 nếu đó là tin châm biếm và bằng 0 cho ngược lại.

1.2. Output:

Đưa ra kết quả dự đoán của các bài viết trong testset và đánh giá mô hình học dựa trên các thông số Accuracy, Precision, Recall và F1–score.

CHƯƠNG 2: TIỀN XỬ LÝ DỮ LIỆU

2.1. Cách thức thực hiện:

Mỗi headline là một dòng dữ liệu kiểu text, để có thể đưa vào mô hình máy học, ta phải thực hiện chuyển dữ liệu text sang cấu trúc vector. Nhóm đã tham khảo mô hình **Vector Space Model** để trích xuất được vector đặc trưng (features vector) của các headlines.

Ngôn ngữ python hỗ trợ sẵn cho việc đọc file *.json. Sau khi đọc được dữ liệu trong file Sarcasm_Headlines_Dataset.json lưu vào thành một list, bắt đầu tách lấy riêng thông tin `headline` và `is_sarcastic` thành hai list để tiện cho việc xử lý.

Tiếp đó, bắt đầu việc tách mỗi headline thành các term (term được hiểu là một đơn vị có nghĩa). Nhóm sử dụng Natural Language Toolkit (NLTK) để hỗ trợ tối đa cho quá trình tách term.

Với mỗi headline, quá trình tách term được thực hiện như sau:

- **Lowercase:** Chuyển tất cả các từ viết hoa (nếu có) về viết thường. Việc này giúp giảm được số lượng từ có trong một headline (ví dụ: ‘Apps’ và ‘apps’ sẽ được xem là một).

Tuy nhiên khi xét như vậy nhóm đã bỏ qua những từ đặc trưng mà đáng lẽ không nên xét giống như từ viết thường (ví dụ: ‘Trump’ và ‘trump’ được xem như là một trong khi đáng lẽ ra ta phải xem nó như là 2 từ).

- **Expand Contraction:** Mở rộng các từ được rút ngắn (contractions). Việc mở rộng các từ được rút ngắn sẽ giúp ích cho việc tách từ sau này. Sử dụng sự hỗ trợ của module contractions.
- **Remove Punctuation:** Xóa các dấu đặc biệt “‘!@#$%^&...” do các dấu này không đem lại đặc trưng cho văn bản. Tương tự như lowercase, nhóm đã bỏ qua một vài trường hợp ta nên giữ lại dấu (ví dụ: “$7” nên được xét như là một term thay vì bị tách thành “$” và “7”) do chưa tìm được cách xóa hiệu quả hơn.

- **Tokenize:** Tách string thành một mảng các từ xuất hiện trong câu. Sử dụng hàm hỗ trợ của thư viện NLTK: word_tokenize(<string>). Hàm trả về một list các từ trong string được bằng việc tìm các từ và dấu câu, tách theo khoảng trắng hoặc xuống dòng. Tuy nhiên hàm yêu cầu cài đặt thêm Punkt sentence tokenization models để có thể thực hiện.

- **Replace Number:** Chuyển các số ở dạng biểu diễn số sang biểu diễn dạng chữ. Sử dụng sự hỗ trợ của module inflect (Đây là một thư viện đơn giản để hoàn thành các nhiệm vụ liên quan đến ngôn ngữ tự nhiên trong chuyển đổi qua lại từ dạng số ít và số nhiều, chuyển đổi các thì của động từ, thêm "a"/ "an" cho từ và chuyển đổi số sang chữ)

- **Remove Stopwords:** Sẽ luôn có những từ mà bất kỳ văn bản nào cũng sẽ dùng như "a"/ "an", "the", "he", "she", v.v… Những từ như vậy ta không nên xét nó như một thành phần tạo nên đặc trưng của văn bản. Việc loại bỏ thường giúp giảm bớt tổng số từ và đem lại hiệu quả tốt hơn. Khi sử dụng NLTK sẽ được hỗ trợ một danh sách các stopwords trong tiếng anh.

- **Lemmatization:** là quá trình đưa một từ về dạng gốc (lemma) của chúng. Từ được trả về có ý nghĩa trong ngôn ngữ. Lemma là dạng nguyên mẫu của một từ (phần không có hậu tố (như –ed, –ize, –s, –de, v.v...)). Có thể dễ hình dung ứng dụng của lemmatization là đưa động từ bất quy tắc V2, V3 về dạng be_infinitive.

Như vậy 2 từ "run", "ran" sẽ được đưa về là "run"

- **Stemming:** cũng là quá trình biến một nhóm từ vựng về dạng gốc (stem) của chúng. Điểm khác biệt với lemma là stem có thể không phải là một từ hợp lệ trong ngôn ngữ và stemming không thể đưa động từ bất quy tắc về be_infinitive.

Như vậy 2 từ "visit", "visited" sẽ đưa về là "visit". Tuy nhiên 2 từ "run", "ran" được xem là 2 từ khác nhau.

Ngoài ra nhóm còn tiến hành bỏ các ký tự không thuộc bảng mã ASCII.

	String ví dụ	String kết quả	Ghi chú
Lowercase	facebook Reportedly Working on Healthcare features and Apps	facebook reportedly working on healthcare features and apps	
Expand contractions	just take it slow, and you'll be fine	just take it slow, and you will be fine	“you’ll” thành ‘you will’
Remove Punctuation	report: adjectives 'tony,' 'snarky' used only by media	report adjectives tony snarky used only by media	dấu ‘:’, ‘ ' ‘, ‘,’ được loại bỏ
Tokenizing	Good muffins cost $3.88\nin New York. Please buy me two of them.\n\nThanks.	'Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.', 'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.'	
Replace numbers	the 23 best songs of 2014	'twenty–three', 'best', 'songs', 'two thousand and fourteen'	‘23’ thành ‘twenty–three’
Remove stopwords	an interview with allen iverson, the	'interview', 'allen', 'iverson', ',', 'realest',	các từ ‘an’, ‘with’, ‘the’, ‘of’ được loại

	realest hall of famer	'hall', 'famer'	bỏ
Lemmatization	titles, ran	'title', 'run'	
Stemming	titles, ran	'titl', 'ran'	

Sau khi tách mỗi headline thành một list các từ mang đặc trưng, tiến hành vector hóa các tiêu đề bằng cách đánh trọng số TF–IDF cho mỗi từ. Độ dài của mỗi vector thu được sẽ bằng với số lượng từ tách được từ file Sarcasm_Headlines_Trainset.json.

TF (term frequency): ý tưởng của TF là đánh trọng số cao hơn cho những từ thường có tần số xuất hiện cao trong văn bản. Những từ như vậy thường mang ý nghĩa đại diện cho nội dung văn bản.

Công thức TF cơ bản:

$$TF(t,d) = f(t,d) \text{ (1.1)}$$

với $f(t,d)$ là số lần xuất hiện của $term\ t$ trong $doc\ d$

Công thức trên gặp hạn chế ở trường hợp nếu cùng một $term\ t$, từ đó có cùng số lần xuất hiện trong 2 $doc\ d1$và $doc\ d2$ thì với cách tính trên thì $TF(t,d1) = TF(t,d2)$. Tuy nhiên do 2 văn bản lại có tổng số lượng từ khác nhau nên ta sẽ muốn $TF(t,d1) \neq TF(t,d2)$. Vì vậy TF được tính lại:

$$TF(t,d) = \frac{number\ of\ times\ term\ t\ appears\ in\ a\ document}{total\ number\ of\ term\ in\ the\ document} = \frac{f(t,d)}{n(d)} \text{ (1.2)}$$

IDF (Inverse Document Frequency): ý tưởng của IDF là đánh trọng số cao hơn cho những từ xuất hiện trong ít văn bản. Những từ như vậy có thể xem như đại diện cho một nhóm văn bản mang nội dung gần gần giống nhau.

Công thức IDF:

$$IDF(t) = 1 + log(\frac{total\ number\ of\ documents}{number\ of\ documents\ with\ term\ t\ in\ it}) = 1 + log(\frac{N}{df(t)}) \text{ (1.3)}$$

Kết hợp TF và IDF để tính được trọng số của một từ trong headline:

$$w(t,d) = TF(t,d) * IDF(t) \text{ (1.4)}$$

2.2. Khó khăn và cách xử lý:

Như đã nói ở mục 2.1, nhóm đã thực hiện hầu hết các cách xử lý tách từ để có thể giảm nhiều nhất có thể tổng số lượng từ của toàn bộ trainset do số lượng từ này sẽ quyết định độ dài của một vector đại diện cho một headline. Vector này ứng với mỗi phần tử sẽ là trọng số w của từ đó nếu từ đó xuất hiện trong văn bản hoặc là 0 nếu từ đó không có trong văn bản.

Ban đầu nhóm đã đưa hết tất cả 26,709 records vào để lập IDF dẫn đến độ dài của IDF là 17,063 ứng với 17,063 term. Sau khi bắt đầu tính toán trọng số cho từng từ trong headline thu được một vector 1 × 17,063. Như vậy với 26,709 records ta sẽ có một ma trận TF–IDF 26,709 × 17,063 (ma trận thưa chứa rất nhiều số 0). Do một số giới hạn về phần cứng mà việc tính toán cũng như lưu trữ ma trận trên xảy ra lỗi do không đủ vùng nhớ. Sau một lúc nhóm cũng đã tìm ra được cách có thể lưu dữ liệu sau khi xử lý xuống file. Tuy nhiên lại nảy sinh vấn đề sau khi file dữ liệu đó đưa vào để tách thành trainset và testset để xây dựng mô hình học thì không đọc được do dữ liệu quá lớn gây lỗi.

Nhóm đã thử dùng module sklearn.feature_extraction để tạo features vector bằng class TfidfVectorizer(). Class này giúp chuyển đổi một tập các văn bản thành một ma trận TF–IDF. Class TfidfVectorizer() cho phép tùy chỉnh các tham số tùy vào để có kết quả mong muốn. Tuy vậy, nhóm giữ tham số ở mặc định. Như vậy nếu một văn bản được fit vô class thì văn bản về sẽ được lowercase, không bỏ các ký tự không thuộc ASCII, tách thành nhiều từ, không loại bỏ stopword.

```
>>> trainXvectorized.shape
(20000, 22178)
>>> testXvectorized.shape
(6709, 22178)
>>>
```

Hình 2.1: Kết quả kích thước ma trận TF–IDF của trainset và testset dùng thư viện hỗ trợ

Nhóm nhận thấy code xây dựng vector TF và IDF được cài đặt gặp lỗi do đã xây dựng vector IDF dựa trên tất cả headlines, đồng nghĩa việc bao gồm cả những term chỉ có trong testset mà không có trong trainset. Trong khi IDF được xây dựng chỉ dựa trên trainset chứ không bao gồm các term trong testset và vector TF–IDF của mỗi mẫu testset sẽ có độ dài bằng độ dài IDF của trainset.

Sau khi load thành công dữ liệu, nhóm sẽ tách thành hai list con, một list chứa dữ liệu để train là 20,000 records đầu của dataset và list còn lại chứa dữ liệu test là 6,709 records sau của dataset.

```
>>> len(idf)
14810
>>> len(traindata)
20000
>>> len(traindata[0])
14811
>>> len(testdata)
6709
>>> len(testdata[0])
14811
```

Hình 2.2: Kết quả kích thước của ma trận TF–IDF của trainset và testset sau khi được xử lý lại. Trainset có kích thước 20,000 × 14,811, testset có kích thước 6,709 × 14,811 (trainset và testset đã bao gồm label).

Ma trận TF–IDF được lưu xuống thành 2 file ‘trainset.npy’ và ‘testset.npy’ để tăng tốc đọc dữ liệu trong file lên cấu trúc mảng của numpy.

CHƯƠNG 3: PHÂN LỚP DỮ LIỆU VỚI CÁC MÔ HÌNH

3.1. Mô hình Logistic Regression:

3.1.1. Giới thiệu thuật toán:

Logistic Regression là một thuật toán supervised learning trong machine learning thường được dùng cho bài toán Binary class Classification.

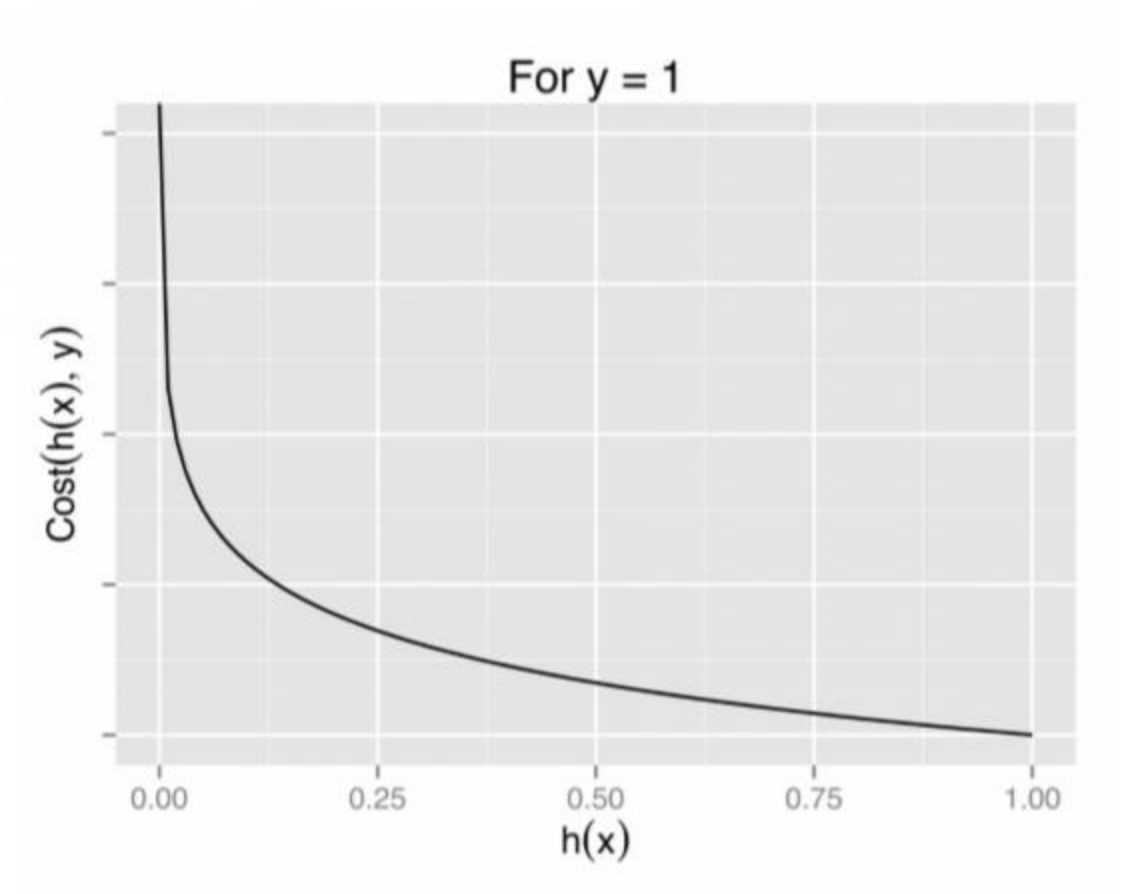
Thuật toán này phân tích dữ liệu đã đưa vào và dự đoán dựa trên xác suất bằng cách sử dụng sigmoid function. Hàm này sẽ ánh xạ một giá trị thực bất kì thành một giá trị khác trong khoảng từ 0 đến 1 theo công thức:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta x}} \quad (3.1)$$

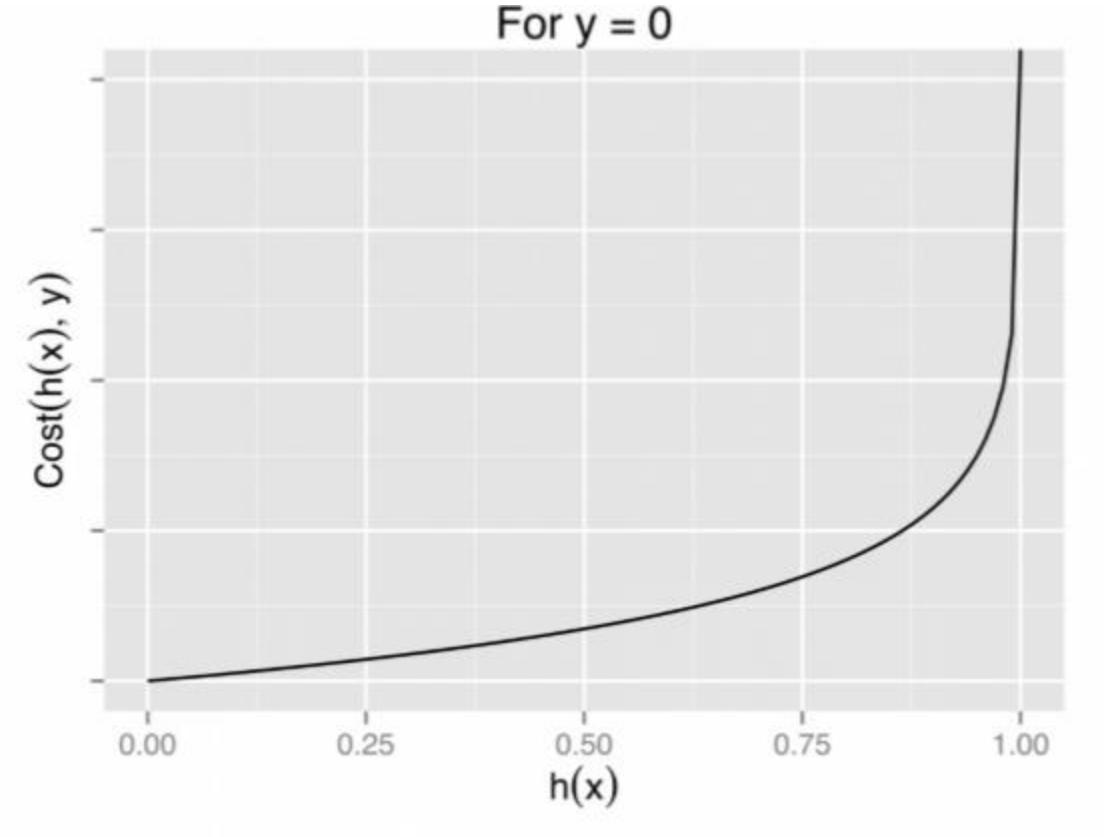
Sau đó mô hình sẽ đưa sigmoid function vào cost function. Đối với thuật toán này cần phải tạo ra một cost function và giảm thiểu nó để có thể xây dựng một mô hình với sai số nhỏ nhất.

Cost function cho logistic regression:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m} [y^{(i)} log(h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}) \quad (3.2)$$



[1]

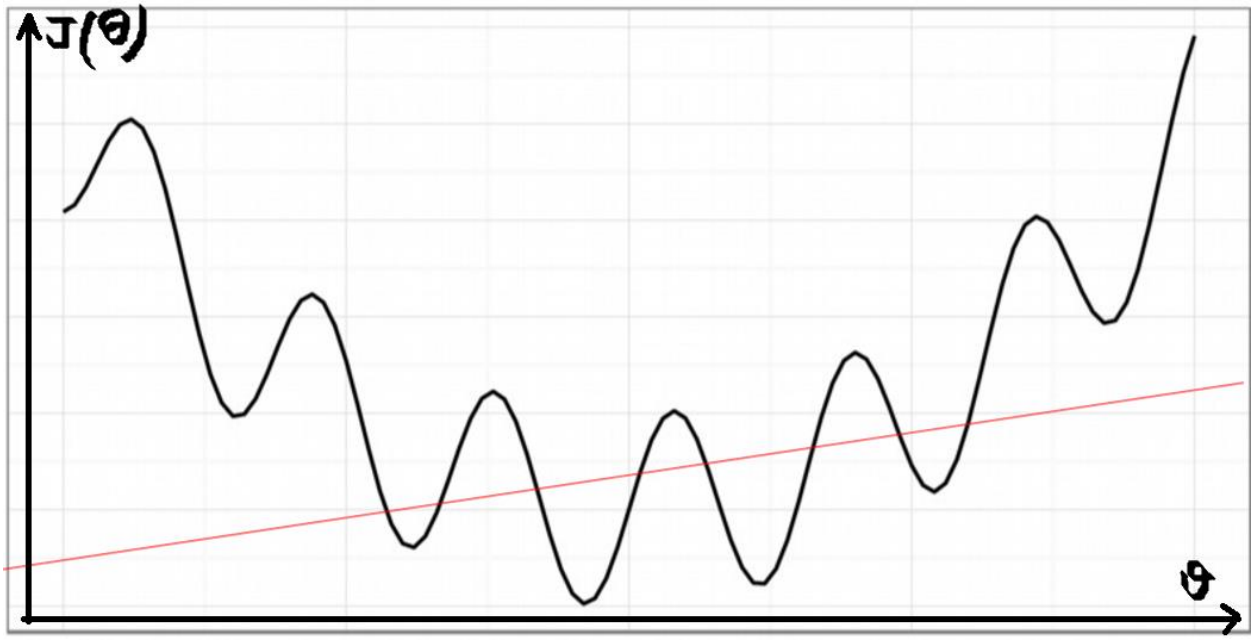


[1]

Thuật toán logistic regression cũng tương tự như linear regression nhưng không thể sử dụng cost function của linear cho logistic do cost function trong linear có công thức:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \quad (3.3)$$

Nếu cố gắng sử dụng hàm này cho logistic thì hàm xây dựng được sẽ trở thành một hàm không lồi với nhiều mức tối thiểu cục bộ. Do đó rất khó để giảm thiểu giá trị cost và tìm mức tối thiểu toàn cục [1].



[1]

Để giảm giá trị cost, thuật toán đã sử dụng Batch Gradient Descent:

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (3.4)$$

với αlà learning rate dùng để kiểm soát độ dốc của gradient descent.

Ta cần phải thực hiện hàm này với mỗi tham số x được đưa vào mô hình. Sau đó đưa các giá trị θđã tính được vào cost function để tìm ra giá trị mới của cost. Bước này sẽ được lặp lại cho đến khi ta tìm được giá trị cost nào xây dựng mô hình với sai số tối thiểu.

Khi đã xây dựng xong mô hình việc cần làm bây giờ là đưa dữ liệu test vào mô hình để dự đoán dữ liệu đó thuộc label nào (0 hoặc 1). Tính xác suất dữ liệu test thuộc class 0 hay 1 với giá trị θđã tính được.

$$P = \frac{1}{1+e^{-\theta x}} \quad (3.5)$$

Nếu P >= 0.5 thì dữ liệu test có label là 1

Nếu P < 0.5 thì dữ liệu test có label là 0

3.1.2. Pseudocode:

1. Khởi tạo vector θ chứa các giá trị là 0 và xây dựng cost function.
2. Tính lại θ mới bằng gradient descent với giá trị α và đưa vào cost function để tính lại giá trị cost mới.
3. Lặp lại bước 2 cho đến khi đạt được giá trị cost nhỏ nhất. Trong quá trình này có thể thay đổi giá trị của αvà tăng số lần lặp để mô hình được xây dựng có sai số nhỏ nhất.
4. Đưa dữ liệu test vào sigmoid function cùng vớiθđã tính được để gán label cho dữ liệu test.

3.2. Mô hình Support Vector Machine:

Support Vector Machine (SVM) là một thuật toán supervised learning trong machine learning thường được dùng cho bài toán classification. Trong bài báo cáo này nhóm sẽ cài đặt thuật toán SVM linear kernel.

Ý tưởng: Cho trước một tập dữ liệu được biểu diễn trong không gian vector, trong đó mỗi dữ liệu là một điểm. Phương pháp này sẽ tìm ra một siêu phẳng (hyperplane) có thể chia các điểm trong không gian thành hai lớp riêng biệt. Trong không gian hai chiều siêu phẳng này là một đường phân chia (đường thẳng, đường tròn, đường cong, ...) mặt phẳng thành hai phần trong đó mỗi lớp nằm ở một bên. Siêu phẳng phân tách dữ liệu càng chính xác khi khoảng cách biên (khoảng cách từ điểm dữ liệu gần nhất của mỗi lớp đến nó) càng lớn.

Trong không gian hai chiều ta giả sử đường thẳng phân chia cần tìm có phương trình là [2]:

$$\theta^T x = \theta_1 x_1 + \theta_2 x_2 + \theta_0 = 0 \quad (3.6)$$

Với cặp dữ liệu (x_n, y_n)bất kỳ khoảng cách từ điểm đó tới mặt phân chia là:

$$\frac{y_n(\theta^T x)}{||\theta||_2} \quad (3.7)$$

Lúc này margin (biên) sẽ được tính là khoảng cách từ điểm dữ liệu gần nhất đến đường thẳng đó:

$$margin = min\left(\frac{y_n(\theta^T x)}{||\theta||_2}\right) \text{ (3.8)}$$

Để xây dựng được mô hình SVM có hyperplane phân tách chính xác nhất ta cần phải tìm θsao cho margin đạt giá trị lớn nhất:

$$(\theta) = argmax_\theta\{min_n\left(\frac{y_n(\theta^T x)}{||\theta||_2}\right)\} \text{ (3.9)}$$

Tuy nhiên để tìm được θtheo công thức trên rất khó nên ta có thể giả sử với mọi n, ta có:

$$y_n(\theta^T x) \geq 1, \forall n = 1,2,\dots N \text{ (3.10)}$$

Như vậy ta chỉ cần tìm giá trị θ thỏa điều kiện:

$$(\theta) = argmax_\theta \frac{1}{||\theta||_2} \text{ (3.11)}$$

Nghĩa là ta có thể đưa về bài toán:

$$(\theta) = argmin\frac{1}{2}||x||_2 \text{ (3.12)}$$

với điều kiện: $1 - y_n(\theta^T x) \leq 0\ \forall n = 1,2,\dots N$

Sau khi giải bài toán trên và tìm được hyperplane có thể phân chia dữ liệu chính xác nhất ta chỉ cần đưa vector của dữ liệu test vào và xét label của dữ liệu đó.

Đối với thuật toán SVM linear kernel ta cần sử dụng thêm công thức:

$$f_i = exp\left(-\frac{||x - l^{[i]}||^2}{2\sigma^2}\right) \text{ với i = 1, 2, …, N (3.13)}$$

Với l là tọa độ của từng điểm giá trị x trong tập dữ liệu. Có bao nhiêu tập dữ liệu là có bấy nhiêu giá trị f ứng với mỗi dữ liệu x. Hàm này được sử dụng để xây dựng những hyperplane đặc biệt trong trường hợp không thể chỉ sử dụng một đường thẳng, đường cong, đường tròn để phân tách các lớp dữ liệu của bài toán. Điểm dữ liệu x sẽ được phân vào class 1 nếu:

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \ldots > 0$$

3.3. Mô hình Naive Bayes:

Naive Bayes là một thuật toán machine learning dựa trên định lý Bayes, được sử dụng nhiều trong các bài toán classification. Trong bài báo cáo này, nhóm xây dựng thuật toán Naive Bayes với bài toán binary classification.

Đối với bài toán này ta có c classes 1, 2, ..., c. Giả sử có một điểm dữ liệu x, ta cần tính xác suất để điểm dữ liệu này có label là class c: $p(y = c|x)$. Từ đó có thể xác định label của điểm dữ liệu này bằng cách chọn ra class có xác suất cao nhất [2]:

$$c = arg\ max_{c \in \{1,\ldots,C\}} p(c|x) \quad (3.14)$$

Biểu thức này rất khó để tính trực tiếp nên cần sử dụng quy tắc Bayes và do mẫu số của biểu thức không phụ thuộc vào c nên ta có thể bỏ qua mẫu số trong quá trình tính toán:

$$c = argmax_c p(c|x) = argmax_c \frac{p(x|c)p(c)}{p(x)} = argmax_c p(x|c)p(c) \quad (3.15)$$

Tuy nhiên $p(x|c)$khá khó để tính toán do x là một biến ngẫu nhiên nhiều chiều nên phải cần rất nhiều dữ liệu train để có thể xây dựng được phân phối của các điểm dữ liệu trong class c. Vì vậy để giúp cho việc tính toán đơn giản hơn người ta thường giả sử rằng nếu dữ liệu x có các thành phần $x_1, x_2, \ldots, x_d$ thì các thành phần này là độc lập với nhau.

$$p(x|c) = p(x_1, x_2, \ldots, x_d|c) = \prod_{i=1}^{d} p(x_i|c) \quad (3.16)$$

Vậy đối với một điểm dữ liệu x, class của nó sẽ được xác định bởi:

$$c = argmax_{c \in \{1,\ldots,C\}} p(c) \prod_{i=1}^{d} p(x_i|c) \quad (3.17)$$

Mặc dù khá là ít dữ liệu mà các thành phần hoàn toàn độc lập với nhau nhưng bằng việc sử dụng giả thiết này ta có thể xây dựng những mô hình phân loại khá tốt. Cũng nhờ vào giả thiết này mà tốc độ train, test của Naive Bayes rất nhanh và có thể mang lại hiệu quả cao trong các bài toán phân loại.

CHƯƠNG 4: CÀI ĐẶT

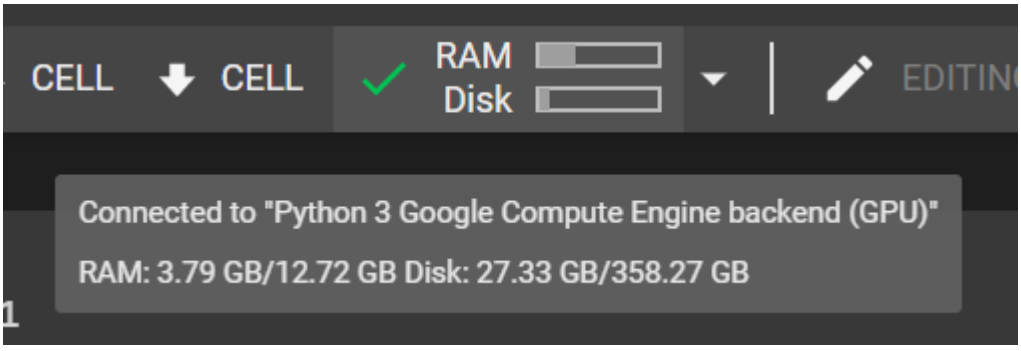
4.1. Thư viện và công cụ hỗ trợ:

4.1.1. Thư viện:

- NLTK là một nền tảng hàng đầu để xây dựng các chương trình Python hoạt động với dữ liệu ngôn ngữ của con người.
- Numpy: là một module mở rộng cho Python, cung cấp chức năng biên dịch nhanh cho các thao tác toán học và số. Hơn nữa nó còn có cấu trúc dữ liệu mạnh mẽ để tính toán hiệu quả các mảng và ma trận đa chiều cùng với các chức năng tính toán cao cấp trên ma trận và mảng. So với "core Python" đơn thuần, numpy có tốc độ xử lý nhanh hơn nhiều lần, đặt biệt là với các ma trận và mảng lớn.
- Scikit-learn: Scikit-learn là thư viện mã nguồn mở rất phổ biến về machine learning của Python, đơn giản và hiệu quả cho việc khai thác dữ liệu và phân tích dữ liệu, đặc biệt thư viện cung cấp rất nhiều công cụ cho máy học. Nó được xây dựng dựa trên numpy, scipy và matplotlib nên rất tương thích với các thư viện này.

4.1.2. Công cụ:

Thay vì phải code và train model với máy tính cá nhân, nhóm đã sử dụng Google Colab với free GPU Tesla K80. Hệ thống có 12.72GB RAM và 358.27GB disk.



4.2. Chuẩn bị dữ liệu:

Load và chia dữ liệu bằng thư viện `numpy`.

```
import numpy as np

traindata = np.load("traindata.npy")

testdata = np.load("testdata.npy")
```

Tách cột cuối trong traindata thành nhãn y, phần còn lại là tập train X. Làm tương tự cho testdata.

```
X = traindata[0: , :-1]     # ma trận 20000 × 14810

y = traindata[0: , -1]      # vector 20000 × 1


Xtest = testdata[0: , :-1]  # ma trận 6709 × 14810

ytest = testdata[0: , -1]   # vector 6709 × 1
```

4.3. Cài đặt thuật toán:

4.3.1. Logistic Regression:

4.3.1.1.Handwrite:

Hàm tính sigmoid: hàm nhận vào một vector, tính toán theo công thức sigmoid (3.1) để tính giá trị cho từng phần tử, hàm trả về một vector.

#np.exp là hàm tính e mũ

```
def sigmoid(z):
    #g = np.zeros((len(z)),1)
    g = 1.0/(1 + np.exp(-z))
```

```
    return g
```

Hàm tính cost: đầu vào của hàm là vector θ , ma trận X và vector y. Kết quả hàm trả về giá trị cost tính được dựa vào công thức (3.2). Có thể thấy trong công thức có xuất hiện việc duyệt từng mẫu dữ liệu, nếu tính toán như vậy với 20000 mẫu sẽ tốn thời gian nên nhóm đã vectorized công thức để có thể tính toán nhanh chóng hơn.

X.dot(theta) thực hiện phép nhân giữa 2 ma trận

transpose() giúp chuyển ma trận thành ma trận chuyển vị.

y.transpose() là vector 1×6709

```
def costFunction(theta, X, y):
    """ number of training examples"""
    m = len(y)
    """ initial return cost values"""
    J = 0
    """ compute cost"""
    h = sigmoid(X.dot(theta))
    J = 1.0/m * ((-y).transpose().dot(np.log(h)) -
                 (1 - y).transpose().dot(np.log(1 - h)))
    return J
```

Hàm Batch Gradient Descent: hàm tính batch gradient (công thức (3.4)) để học tham số θ sau mỗi lần lặp với learning rate α. Hàm trả về bộ tham số θ học được và list chứa sự thay đổi của cost sau mỗi lần lặp. Cũng tương tự `costFunction`, nhóm đã vectorized công thức để có thể tính toán nhanh chóng hơn.

```
def BGD(X, y, theta, alpha, num_iters=1000):
    """initial some values"""
    m = len(y)
    J_history = np.zeros((num_iters, 1))
    for iter in range(0, num_iters):
        """ perform a single gradient step on the parameter vector theta."""
        h = sigmoid(X.dot(theta))
        theta = theta - (alpha / m) * X.transpose().dot((h - y))
        """ save the cost J in every iteration"""
        J_history[iter] = costFunction(theta, X, y)
        #print(iter)
    return theta, J_history
```

Hàm predict: đưa ra dự đoán ứng với Xtest hoặc dữ liệu mới (nếu có). Hàm sẽ quyết định label là 0 hoặc 1 bằng việc sử dụng tham số θ đã học. Hàm sẽ đưa ra dự đoán với ngưỡng

threshold mặc định bằng 0.5. Nếu kết quả dự đoán mới >= 0.5 thì dự đoán nhãn là 1, ngược lại dự đoán là 0.

```
def predict_handwrite(theta, X, threshold=0.5):
    """ number of training examples"""
    m = X.shape[0]
    """ return values"""
    p = np.zeros((m, 1))
    for i in range(0, m):
        print(sigmoid(X[i, :].dot(theta)))
        if sigmoid(X[i, :].dot(theta)) >= threshold:
            p[i] = 1
        else:
            p[i] = 0
    return p
```

Hàm chính để xây dựng mô hình logistic regression cho dữ liệu.

```
def LogisticRegression_handwrite(X_, Xtest_, y_, alpha,
num_iters=1000, threshold=0.5):
    X = copy.deepcopy(X_)
    y = copy.deepcopy(y_)
```

```
    Xtest = copy.deepcopy(Xtest_)
    y = y.reshape((len(y), 1))
    m, n = X.shape
    mtest = Xtest.shape[0]

    """ add intercept term x0 to X"""
    X = np.hstack((np.ones((m, 1)), X))
    Xtest = np.hstack((np.ones((mtest, 1)), Xtest))
    """ initialize fitting parameters"""
    initial_theta = np.zeros((n + 1, 1))
    """ run BGD to obtain the optimal theta"""
    theta, cost = BGD(X, y, initial_theta, alpha=alpha,
num_iters=num_iters)
    """ get prediction on testset"""
    p = predict_handwrite(theta, Xtest, threshold=threshold)
    return p, cost
```

4.3.1.2.Dùng model của thư viện:

Nhóm sử dụng module LogisticRegression trong thư viện sklearn để việc xây dựng mô hình.

```
from sklearn.linear_model import LogisticRegression
```

Tham số `solver = 'saga'`, tối ưu độ lỗi tính toán của mô hình và nhanh hơn khi dùng cho các tập dữ liệu lớn.

```
model = LogisticRegression(solver='saga')
```

Sử dụng dữ liệu đã xử lý ở mục 4.2.1, Dùng hàm fit để train mô hình trên và dùng hàm predict để dự đoán kết quả của Xtest

```
model.fit(X, y)
```

```
p = model.predict(Xtest)
```

4.3.2. KNN:

Cách xây dựng mô hình KNN bằng thư viện trong báo cáo này tương tự như trong báo cáo ở case study 1 – mục 2.3 – trang 6 [].

4.3.3. Decision Tree:

Nhóm đã sử dụng module tree trong thư viện scikit-learn hỗ trợ cho python.

Mô hình này đã được cài đặt sẵn thuật toán Decision Tree. Thuật toán này xây dựng cây quyết định theo phương pháp CART (Classification and Regression Trees) được phát triển năm 1984. Các phương pháp được dùng để xây dựng cây quyết định đều dựa trên phương pháp ID3 đã được nhóm báo cáo ở thực hành 2 – mục 3.1 []. C4.5 là một phương pháp được cải tiến từ ID3, so với ID3 thì nó không cần biến số phân loại đặc trưng, output theo dạng if-then không hiển thị những nhánh cây không cần thiết. C5.0 là bản cải tiến của C4.5 giúp cải thiện vấn đề hiệu năng và sử dụng ít bộ nhớ hơn. CART khá giống với C4.5 nhưng phương pháp này tạo cây quyết định dựa trên biến phân loại, giải thích, mục đính và hồi quy [3].

Để xây dựng mô hình ta cần phải fit dữ liệu đã tách ở mục 4.2.1 vào mô hình:

```
model = DecisionTreeClassifier().fit(vector_train, y)
```

4.3.4. Support Vector Machine:

Để việc xây dựng mô hình SVM đơn giản nhóm đã sử dụng module svm trong thư viện scikit-learn hỗ trợ cho python.

```
from sklearn import svm
```

Module sklearn.svm đã được cài đặt sẵn thuật toán SVM linear kernel trong hàm svc. Hàm này được truyền vào nhiều tham số nhưng để xây dựng mô hình SVM linear kernel ta cần truyền vào tham số kernel = 'linear'. Để xây dựng mô hình ta cần tạo biến:

```
model = svm.SVC(kernel='linear')
```

Sau khi khởi tạo mô hình ta cần đưa tập dữ liệu train đã tách ở mục 4.2.1 vào mô hình:

```
model.fit(vector_train,y)
```

4.3.5. Naive Bayes:

Nhóm đã sử dụng module BernoulliNB trong thư viện scikit-learn hỗ trợ cho python.

```
from sklearn.naive_bayes  import BernoulliNB
```

Module sklearn.naive_bayes đã được cài đặt sẵn thuật toán Naive Bayes trong hàm BernoulliNB(). Thuật toán được cài đặt sẵn trong hàm này xây dựng mô hình theo thuật toán Naive Bayes mà nhóm đã giới thiệu ở 3.5. Để xây dựng được mô hình ta cần phải fit dữ liệu đã tách ở mục 4.2.1 vào mô hình:

```
model = BernoulliNB().fit(vector_train, y)
```

**Sau khi xây dựng mô hình với các thuật toán trên nhóm đã sử dụng hàm predict(X) (với X là test set), giúp đưa ra dự đoán dựa trên mô hình vừa có và sử dụng các hàm trong module metrics của thư viện scikit-learn để tính Accuracy, Precision, Recall, F1-score.*

CHƯƠNG 5: KẾT QUẢ VÀ ĐÁNH GIÁ MÔ HÌNH

5.1. Kết quả:

Để đánh giá và so sánh giữa các mô hình phân lớp, nhóm dựa vào 4 giá trị:

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total\ Examples} \quad (5.1)$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (5.2)$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (5.3)$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (5.4)$$

Mô hình	Tham số	Accuracy	Precision	Recall	F1-score	Tổng thời gian "học" và đưa dự đoán (s)
Logistic Regression handwrite	alpha = 5, threshold = 0.5 num_iters = 1000	0.777612162766433 2	0.765902366863905 3	0.706825938566552 8	0.735179268725594 5	878.8698782920837
Logistic Regression sklearn		0.775525413623490 8	0.760614934114202	0.709215017064846 5	0.734016248675379 8	418.9540874958038 3

KNN						
Decision Tree		0.700998658518408 1	0.659640635798203 2	0.651535836177443	0.655563186813186 8	1466.523723840713 5
Naive Bayes		0.779847965419585 7	0.796409628722970 2	0.666211604095563 1	0.725515703400854 9	6.227381467819214 s
SVM						

5.2. Đánh giá:

Từ bảng trên ta có thể thấy các mô hình đưa ra kết quả dự đoán có độ chính xác khá cao. Hầu hết các giá trị accuracy, precision, recall và F1-score đều có giá trị lớn hơn 0,7. Nghĩa là các mô hình được xây dựng "học" khá tốt với dữ liệu đã được xây dựng theo mô hình Vector Space model. Tuy nhiên thời gian "học" của mỗi mô hình lại có sự khác biệt khá lớn.

Đặc biệt là đối với mô hình Decision Tree do dữ liệu có quá nhiều thuộc tính nên thuật toán này sẽ mất rất nhiều thời gian để xây dựng cây. Giá trị F1-score của mô hình này cũng thấp hơn so với các mô hình khác do mô hình này dễ xảy ra lỗi khi xây dựng cây quyết định cao với quá nhiều thuộc tính cần tính toán. Do vậy mô hình này không phù hợp để sử dụng cho bài toán này.

Mô hình có thời gian chạy ngắn nhất là Naive Bayes do mô hình này đã giả sử các thành phần của mô hình là hoàn toàn độc lập nhau.

REFERENCES

[1] A. Pant, "Introduction to Logistic Regression – Towards Data Science," [Online]. Available: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148. [Accessed 17 May 2018].

[2] N. V. Tiệp, Machine Learning cơ bản, 2018.

[3] "Phân tích cây quyết định với scikit-learn | Code từ đầu - machine learning," 03 Sep 2017. [Online]. Available: https://codetudau.com/phan-tich-cay-quyet-dinh-voi-scikit-learn/index.html. [Accessed 17 May 2019].