TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

MÔN: MÁY HỌC

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

CASE STUDY #3

LOGISTIC REGRESSION

SARCASM DETECTION

Giảng viên: Huỳnh Thị Thanh Thương

Sinh viên thực hiện:

17520324 – Nguyễn Thành Danh

17521244 – Hồ Sỹ Tuyến

17520828 – Phan Nguyên

Nội dung Case Study #1

I. Define a learning problem (Phát biểu bài toán)

1. Task (Mục đích)

- Dựa trên bộ dữ liệu được cung cấp, bằng các phương pháp học Logistic Regression, khi nhận dữ liệu đầu vào là một tiêu đề của một tin mới sẽ đưa ra được câu trả lời tin đó là tin châm biếm hay không.

2. Input (Đầu vào)

- Tập dữ liệu là các tiêu đề của những bài viết được thu thập từ 2 trang web:
  - The Onion: đăng những bài châm biếm.
  - The Huff Post: đăng những bài chính thống.
- Tập dữ liệu được lưu dưới dạng file json với 26,709 hàng tương ứng 26,709 tiêu đề được thu thập. Trong đó:

Bài viết châm biếm	11,725
Bài viết không châm biếm	14,984

- Với mỗi hàng, định dạng dữ liệu như sau:

  {“article_link”: “Đường dẫn tới bài viết”, “headline”: “Nội dung headline”, “is_sarcastic”: x}

  Trong đó: x bằng 1 nếu là bài viết châm biếm, bằng 0 nếu là bài viết chính thống.
- Tập dữ liệu được chia thành 2 phần gồm trainingset và testset với tỷ lệ 20,000:6,709.

3. Output (Đầu ra)

- Dựa trên tiêu đề của một bài viết mới đưa ra được quyết định bài viết đó thuộc loại châm biếm hay không. (isSarcastic/not)

4. Evaluation Method (Cách đánh giá mô hình)

- Đánh giá bằng Precision (độ chính xác) và Recall (độ phủ).
- Từ đó tính ra được F1-Score là trung bình điều hòa của 2 tiêu chí trên.
- Cụ thể ở phần IV Performance Evaluation.

II. Build a model (Xây dựng mô hình từ tập huấn luyện)

1. Chọn phương pháp demo:

Sử dụng phương pháp học Logistic Regression để học 1 mô hình từ tập dữ liệu huấn luyện (gồm 20000 inputs).

2. Biểu diễn text thành vector:

Vì đầu vào các tiêu đề là một chuỗi ký tự mà máy tính không thể hiểu, ta cần chuyển chuỗi ký tự đó sang dạng vector để tạo thành tập features trước khi thực hiện thuật toán. Mô hình mà nhóm sử dụng để thực hiện là Vector Space Model.

3. Trình bày thuật toán Logistic Regression:

B1: Khởi tạo các lists chứa các bộ dữ liệu X, y cho trainingset và testset.

B2: Định nghĩa hàm kích hoạt (activate function) f(x) và learning rate γ.

B3: Khởi tạo bộ tham số w để tối ưu hóa mô hình.

B4: Định nghĩa hàm mất mát:

$$J(w) = \frac{-1}{m}\sum_{i=1}^{m}(y^i.\log(a^i) + (1-y^i).\log(1-a^i))$$

Trong đó:

m: số lượng mẫu dữ liệu.

y^i: nhãn của mẫu dữ liệu thứ i.

a^i: giá trị dự đoán phi tuyến tại mẫu dữ liệu thứ i (tính ở bước 5).

B5: Tính toán giá trị đầu ra dự trên bộ tham số w hiện tại:

- Mô hình tuyến tính:
  - Tính toán giá trị đầu ra tuyến tính z dựa vào bộ features X và bộ tham số w hiện tại.

$$z = w^T.X$$

- Mô hình phi tuyến:
  - Tính toán giá trị đầu ra phi tuyến a dựa vào giá trị tuyến tính z và hàm kích hoạt đã được định nghĩa.

$$a = f(z)$$

B6: Cập nhật trọng số w:

- Có nhiều phương pháp để cập nhật trọng số, trong phạm vi bài báo cáo, nhóm chúng em sử dụng phương pháp Gradient Descent.
- Tính đạo hàm của hàm mất mát J theo mỗi phần tử trong bộ tham số w

$$\frac{\partial J(w)}{\partial w_j} = \frac{-1}{m}\sum_{i=1}^{m} X_j^T(a_j - y_j)$$

- Tùy thuộc vào phương pháp Batch Gradient Descent, mini-Batch Gradient Descent hay Stochastic Gradient Descent ta lại cập nhật w theo số lượng mẫu dữ liệu khác nhau theo công thức:

$$w = w - \gamma.\frac{\partial J(w)}{\partial w_j}$$

- Thuật toán dừng khi hàm mất mát J không thay đổi quá nhiều sau lần lặp kế tiếp.

4. Minh họa thuật toán:

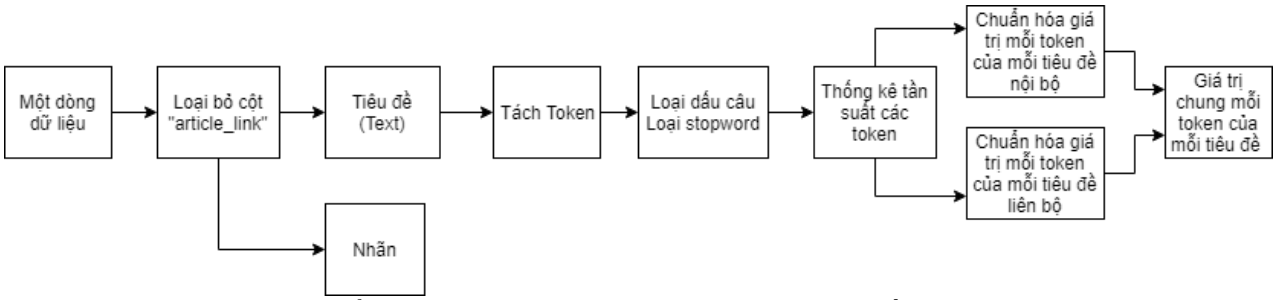
Vì dữ liệu đầu vào của bài toán khá lớn nên không khả thi khi chạy tay. Ở phần sau, nhóm chúng em sẽ giải thích rõ ràng các bước làm của thuật toán.

III. Performance Evaluation (Thực nghiệm đánh giá)

1. Tổ chức Dataset:

- Dataset đã qua tiền xử lý gồm 26,709 bộ dữ liệu (đặc trưng + nhãn) được chia làm 2 bộ Trainningset và Testset.
- Trainingset: 20,000 bộ dữ liệu đầu tiên của của tập dữ liệu.
- Testset: 6,709 bộ dữ liệu tiếp theo của của tập dữ liệu.
- Data được tiền xử lý trước khi được đưa vào thuật toán theo mô hình Vector Space Model bằng phương pháp TF-IDF.

- Quy trình xử lý tập dữ liệu:



- Công thức chuẩn hóa giá trị mỗi token của mỗi tiêu đề nội bộ:

$$tf(t,d) = \frac{n_t}{n_{MAX}}$$

- Công thức chuẩn hóa giá trị mỗi token của mỗi tiêu đề liên bộ:

$$idf(t,d) = \log\left(\frac{|D|}{1+\{d \in D \mid t \in d\}}\right)$$

- Trong đó:

  t: token đang xét tới.

  d: tiêu đề có chứa t.

  D: tập các tiêu đề.

  n_t: số lần xuất hiện của token.

  n_{MAX}: số lần xuất hiện của token có n_t cao nhất.

- Kết quả thu được: Từ tập đầu vào ta thu được tập dữ liệu mới gồm:
  - Tập features (26,709 x A). Với A là số lượng token có trong tập dữ liệu.
  - Tập labels (26,709 x 1).

2. Cách đánh giá:

- Đánh giá bằng **Precision** (xem xét trong testset có bao nhiêu dữ liệu được mô hình dự đoán đúng) và **Recall** (xem xét trong số các dữ liệu được mô hình dự đoán đúng có bao nhiêu dữ liệu là đúng). Theo công thức[(1)]:

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{P}$$

Trong đó:

- Lớp positive là lớp có giá trị y = 1.
- TP (True Positive): tổng số các mẫu mà mô hình dự đoán là positive $\hat{y} = 1$ và thực sự có nhãn positive y = 1.
- FP (False Positive): là tổng số các mẫu mô hình dự đoán là positive $\hat{y} = 1$ nhưng thực sự có nhãn là negative y = 0.
- P là tổng số mẫu positive trong testset.

- Từ đó tính ra được **F1-Score** là trung bình điều hòa của 2 tiêu chí trên, nó có xu hướng lấy giá trị gần với giá trị nhỏ hơn giữa Precision và Recall đồng thời có giá trị lớn nếu cả Precision và Recall đều lớn.

$$F1\_Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

3. Kết quả đánh giá của từng phương pháp với tập dữ liệu được xử lý bằng thư viện có sẵn:

i. Sử dụng phương pháp K-nearest neighbor – code bằng thư viện có sẵn:

- Theo công thức, ta tính được Precision, Recall và F1-Score:

$$Precision = \frac{TP}{TP + FP} = \frac{1747}{1747 + 336} = 0.8268$$

$$Recall = \frac{TP}{P} = \frac{1747}{2930} = 0.5962$$

$$F1_{Score} = 2 \times \frac{Precision \times Recall}{Precision + Recall} = 2 \times \frac{0.8268x0.5862}{0.8268 + 0.5862} = 0.6928$$

$$Accurancy = 0.7691$$

Trong đó:

- Lớp positive là lớp có giá trị isSarcasm = yes.

- TP (True Positive): tổng số các mẫu mà mô hình dự đoán là positive $\widehat{\text{isSarcasm}} = yes$ và thực sự có nhãn positive isSarcasm = yes.
- FP (False Positive): là tổng số các mẫu mô hình dự đoán là positive $\widehat{\text{isSarcasm}} = yes$ nhưng thực sự có nhãn là negative isSarcasm = no.
- P là tổng số mẫu có nhãn là yes trong testset.

Áp dụng công thức như trên, ta tính cho các phương pháp khác, để ngắn gọn và dễ so sánh, dưới đây xin phép chỉ trình bày kết quả dưới dạng bảng.

Accurancy	TP	FP	P	Precision	Recall	F1-score
0.7691	1747	336	2930	0.8268	0.5962	0.6928

ii. Sử dụng phương pháp Decision Tree – code bằng thư viện có sẵn:

Accurancy	TP	FP	P	Precision	Recall	F1-score
0.7417	2072	875	2930	0.7031	0.7072	0.7051

iii. Sử dụng phương pháp Bernoulli Naïve Bayes – code bằng thư viện có sẵn:

Accurancy	TP	FP	P	Precision	Recall	F1-score
0.8405	2203	343	2930	0.8652	0.7519	0.8046

iv. Sử dụng phương pháp Logistic Regression – code bằng thư viện có sẵn:

Accurancy	TP	FP	P	Precision	Recall	F1-score
0.8424	2354	481	2930	0.8303	0.8034	0.8167

v. Sử dụng phương pháp Logistic Regression – code tay:

Nhóm chúng em đã sử dụng nhiều phương pháp vector hóa và xử lý chuỗi và thu được thực nghiệm kết quả như sau (các kết quả được lưu trong thư mục final):

	Accurancy	Precision	Recall	F1-score
TF + non-SW + chọn ngưỡng hợp lý	**0.8262**	0.7845	**0.8300**	**0.8066**
TF + non-SW	0.8258	0.8153	0.7771	0.7957
IDF + non-SW	0.8211	**0.8184**	0.7587	0.7875
TF-IDF + non-SW	0.8240	0.8172	0.7690	0.7923
Boolean + non-SW	0.8214	0.8168	0.7621	0.7885

TF + SW	0.7773	0.7660	0.7058	0.7346
IDF + SW	0.7776	0.7772	0.6881	0.7299
TF-IDF + SW	0.7760	0.7659	0.7014	0.7322
Boolean + SW	0.7791	0.7789	0.6901	0.7318

4. Nhận xét:

i. Nhận xét kết quả:

- Tổng kết lại kết quả F1-Score của các phương pháp: (chọn phương pháp code tay hiệu quả nhất)

	K-Nearest Neighbors	Decision Tree	Naive Bayes	Logistic Regression	Logistic Regression (code tay)
Accurancy	**0.7691**	**0.7414**	**0.8405**	**0.8424**	**0.8262**
Precision	**0.8268**	**0.7080**	**0.8652**	**0.8303**	**0.7845**
Recall	**0.5962**	**0.6942**	**0.7519**	**0.8034**	**0.8300**
F1-score	**0.6928**	**0.7010**	**0.8046**	**0.8167**	**0.8066**

Như vậy, theo cách đánh giá bằng F1-Score, ta suy ra được phương pháp Logistic Regression là mô hình học hiệu quả nhất trong 3 mô hình học được thử nghiệm.

ii. Nhận xét các mô hình học:

- Logistic Regression cho kết quả tốt nhất vì đây là mô hình học có tham số hóa so với các mô hình còn lại.
- Mô hình Naïve Bayes cũng cho kết quả khá tốt vì với phương pháp vector hóa dữ liệu ta đã dùng, mỗi từ đều độc lập với nhau. Tuy nhiên nhược điểm của Naïve Bayes cũng như các model K-nearest Neighbors và Decision Tree là không có sự tham số hóa model.
- K-nearest Neighbors cho ra kết quả kém hơn Naïve Bayes và Logistic Regression không những vì không có sự tham số hóa mà ta còn phải tự điều chỉnh ngưỡng k hợp lý. Mặt khác thuật toán này cũng tốn khá nhiều thời gian và tài nguyên để xử lý.
- Decision Tree cho ra kết quả kém hơn Naïve Bayes và Logistic Regression không những vì không có sự tham số hóa mà ta còn phải tự điều chỉnh số lượng phân chia tại

mỗi node, số node lá, chiều cao của cây,... Một điểm trừ nữa là thuật toán bị overfiting (chỉ đúng trên tập train) nên cho độ bao phủ không cao khi đưa vào tập dữ liệu test.

iii. Nhận xét phương pháp vector hóa dữ liệu và tối ưu hóa kết quả:

- Việc không loại bỏ stopword trong phần code tay mang lại hiệu quả cao hơn vì lý do trong bộ stopword mà nhóm sử dụng vẫn mang giá trị để xác định nhãn bài toán. Việc loại bỏ stopword sẽ ảnh hưởng đến hiệu quả phân loại.

5. Ưu điểm – Hạn chế của các mô hình học:

- K-Nearest Neighbors:
  - Ưu:
    - Độ phức tạp tính toán trong quá trình training thấp.
    - Việc dự đoán nhãn của dữ liệu mới đơn giản.
  - Nhược:
    - Khó khăn trong quá trình xác định neighbor gần nhất với điểm đang xét do có thể có nhiều điểm dữ liệu cùng cách đều điểm đang xét, dẫn đến khó khăn trong việc gắn nhãn điểm dữ liệu đó.
    - Tính toán khoảng cách nhiều lần, tốn bộ nhớ đối với các cơ sở dữ liệu nhiều chiều và nhiều điểm dữ liệu, ảnh hưởng hiệu năng của mô hình. Đặc biệt trong bài toán Xử lý ngôn ngữ tự nhiên vì số chiều dữ liệu rất lớn.
- Naive Bayes:
  - Ưu:
    - Giả định các đặc trưng độc lập nhau thì kết quả sẽ rất tốt do dựa trên xác suất các bộ dữ liệu học được từ trainingset.
  - Nhược:
    - Giả định tất cả các đặc trưng đều độc lập, không tác động lẫn nhau, điều này bất hợp lý trong thực tế.
- Decision Tree:
  - Ưu:
    - Khi tạo thành cây, ta thấy được rõ đặc trưng của data trong trainingset.

        - Có độ ưu tiên giữa các đặc trưng, đặc trưng có tính quyết định cao sẽ là/gần node gốc, các đặc trưng ít quan trọng hơn sẽ ở xa hơn.
    - Nhược:
        - Bỏ qua một số đặc trưng khi đưa ra quyết định.
        - Độ chính xác cao tuy nhiên độ phủ lại kém (rút ra từ thực nghiệm trên dataset).
- Logistics Regression:
    - Ưu:
        - Sử dụng mô hình học có tham số hóa so với các thuật toán trước như Naive Bayes, KNN và Decision Tree.
        - Boundary có dạng phi tuyến.
    - Nhược:
        - Boundary tuy có dạng phi tuyến, nhưng thực chất chỉ có khả năng phân chia dữ liệu có dạng tách rời nhau một cách tuyến tính (linearly separable). Logistic không có khả năng phân chia dữ liệu phi tuyến (ví dụ một lớp điểm nằm trong một lớp vòng tròn khác).
        - Các điểm dữ liệu được tạo ra một cách độc lập với nhau nhưng thực tế chúng lại có thể ảnh hưởng qua lại với nhau.

IV. Implementation (Lập trình cài đặt)

1. Các thư viện đã sử dụng:

- ***Sklearn***: Scikit-learn là một thư viện máy học phần mềm miễn phí cho ngôn ngữ lập trình Python. Nó có các thuật toán phân loại, hồi quy và phân cụm khác nhau bao gồm các support vector machines, random forests, gradient boosting, k-means và DBSCAN, được thiết kế để tương tác với các thư viện khoa học và số của Python NumPy và SciPy.
- ***NumPy***: (viết tắt của Numerical Python) là một thư viện rất cần thiết khi chúng ta xây dựng các ứng dụng Máy học trên Python. Numpy cung cấp các đối tượng và phương thức để làm việc với mảng nhiều chiều và các phép toán đại số tuyến tính. [5]Nó là một mô-đun mở rộng mã nguồn mở cho Python, cung cấp các chức năng biên dịch nhanh

cho các thao tác toán học và số. Hơn nữa, NumPy làm phong phú ngôn ngữ lập trình Python với các cấu trúc dữ liệu mạnh mẽ để tính toán hiệu quả các mảng và ma trận đa chiều.

- ***Pandas***: là một thư viện mã nguồn mở, hỗ trợ đắc lực trong thao tác dữ liệu. Thư viện được sử dụng rộng rãi trong nghiên cứu lẫn phát triển các ứng dụng khoa học dữ liệu. Thư viện này sử dụng một cấu trúc dữ liệu riêng là Dataframe. Pandas cung cấp rất nhiều chức năng xử lý và làm việc trên cấu trúc dữ liệu này. Chính sự linh hoạt và hiệu quả đã khiến cho pandas được sử dụng rộng rãi.
- ***nltk:*** (viết tắt của Natural language ToolKit) là bộ công cụ ngôn ngữ tự nhiên, là một thư viện được viết bằng Python hỗ trợ xử lý ngôn ngữ tự nhiên. Bằng cách cung cấp các cơ chế và kỹ thuật xử lý ngôn ngữ phổ biến, nó giúp cho việc xử lý ngôn ngữ tự nhiên trở lên dễ dàng và nhanh chóng hơn. Được viết bởi Steven Bird và Edward Loper. Ngoài việc hỗ trợ xử lý ngôn ngữ, NLTK còn có các mô phỏng đồ hoạ và dữ liệu mẫu hữu ích. NLTK cung cấp các xử lý như classification, tokenization, stemming, tagging, parsing, và semantic reasoning... Những ứng dụng này chúng ta sẽ dần được tìm hiểu ở những bài viết sau. Ngoài việc phục vụ xử lý ngôn ngữ tự nhiên, NLTK còn được sử dụng trong Machine Learning với tác dụng làm sạch dữ liệu, xử lý dữ liệu đầu vào cho các thuật toán Machine Learning.
- ***SciPy***: (viết tắt của Science Python) là một thư viện mã nguồn mở các thuật toán và các công cụ toán học cho Python. Cung cấp khá nhiều module tính toán từ đại số tuyến tính, tích phân, vi phân, nội suy đến xử lý ảnh, fourier transform,… được sử dụng khá rộng rãi do có hệ thống thư viện tính toán mạnh, ngôn ngữ Python rõ ràng, dễ hiểu, gần gũi với ngôn ngữ tự nhiên, mã nguồn mở và cộng đồng đang dần lớn mạnh từng ngày.

2. Hướng dẫn cài đặt tools và giải thích các hàm đã sử dụng:

Với các thuật toán ở các Case Study trước, nhóm chúng em không tiến hành cài tay lại thuật toán mà chỉ dùng hàm có sẵn. Với thuật toán Logistics Regression chúng em thực hiện ở cả 2 phần: code tay và cài đặt bằng thư viện.

i. Thư viện Sklearn:

a. Train model:

- **Class KNeighborsClassifier(n_neighbors, weights):**

Chức năng: Khởi tạo phân loại bằng cách thực hiện KNN.

Tham số:

  - n_neighbors: kiểu int, số lượng láng giềng gần nhất để quyết định phân lớp.
  - weights: kiểu string, hàm trọng số để dự đoán. Ví dụ: ‘uniform’ là các trọng số được đối xử công bằng với nhau, ‘distance’ là các trọng số được đối xử dựa vào khoảng cách.

- **Class BernoulliNB():**

Chức năng: Phân loại bằng cách thực hiện Bernoulli Naïve Bayes.

- **Class DecisionTreeClassifier():**

Chức năng: Khởi tạo phân loại bằng Decision Tree.

- **Class LogisticRegression():**

Chức năng: Khởi tạo phân loại bằng Logistic Regression.

- **Hàm fit(X,y):**

Chức năng: fix model bằng cách sử dụng X làm dữ liệu huấn luyện, y làm giá trị đích (label).

- **Hàm predict(X):**

Chức năng: Hàm ước tính trả về xác xuất cho dữ liệu huấn luyện X.

```python
# vector hóa train set
def Train(train_X, train_y, test_X, type_train=None):
  # khởi tạo model cho từng loại phân loại
  if type_train == 'LO':
    logist = LogisticRegression()
    logist.fit(train_X, train_y)
    reg_predicted_logist = logist.predict(test_X)
    return reg_predicted_logist
  elif type_train == 'DC':
    DC = DecisionTreeClassifier()
    DC.fit(train_X, train_y)
    reg_predicted_DC = DC.predict(test_X)
    return reg_predicted_DC
  elif type_train == 'NB':
    NB = BernoulliNB()
    NB.fit(train_X.toarray(), train_y)
    reg_predicted_NB = NB.predict(test_X.toarray())
    return reg_predicted_NB
  elif type_train == 'KNN':
    KNN = KNeighborsClassifier(n_neighbors=10, weights='distance')
    KNN.fit(train_X, train_y)
    reg_predicted_KNN = KNN.predict(test_X.toarray())
    return reg_predicted_KNN
```

Hình IV-1

b. Xử lý dữ liệu:

- **Hàm TfidfVectorizer(stopwords, ngram_range, lowercase, max_features)**

Chức năng: Khởi tạo model chuyển một chuỗi ký tự X thành ma trận đặc trưng TF-IDF.

Tham số:

  - stopwords: kiểu list, nếu stopwords khác None, hàm sẽ sử dụng stopwords để loại bỏ các từ trong stopwords ra khỏi chuỗi trước khi vector hóa.
  - ngram_range: kiểu tuple dạng (min, max). Số lượng n token mà phương pháp TF-IDF lấy làm một đặc trưng với min $min \leq n \leq max$.
  - lowercase: kiểu boolean, nếu là Yes thì chuyển tất cả các ký tự thành ký tự thường trước khi token hóa.
  - max_features: Nếu max_features khác None, xây dựng một từ điển chỉ chứa các ký tự có tần suất xuất hiện cao nhất.

- **Hàm fit_transform(X)**

Chức năng: Thực thi model Tfidf với đầu vào là bộ chuỗi ký tự X kiểu string.

- **Hàm transform(X)**

Chức năng: Chuyển hóa bộ chuỗi ký tự X kiểu string sang không gian vector.

```
text_transformer = TfidfVectorizer()

train_X = text_transformer.fit_transform(title[:20000])
train_y = label[:20000]
test_X = text_transformer.transform(title[20000:])
test_y = label[20000:]
```

Hình IV-2

ii. Thư viện Pandas:

- **read_json(path, lines)**

Chức năng: đọc file định dạng .json

Tham số:

path: kiểu string, đường dẫn đến file.json.

lines: kiển Boolean, khi lines = True sẽ đọc từng dòng trên file json.

- **Dataframe.pop(item)**

Chức năng: Loại bỏ cột item ra khỏi Dataframe.

- **Dataframe.dropna()**

Chức năng: Loại bỏ những dòng dữ liệu bị thiếu thông tin.

- **Dataframe.apply(options)**

Chức năng: Xử lý dữ liệu trên hàng/cột dữ liệu theo options code mà người dùng định nghĩa.

Figure IV-1

```
# Đọc file
def ReadData(path):
  raw_df = pd.read_json(path, lines = True)
  return raw_df

def PreProcessing_Data(cleaned_df, stopwords):
  # loại bỏ cột article_link
  cleaned_df.pop('article_link')

  # loại bỏ những dữ liệu bị thiếu
  cleaned_df.dropna()

  # xử lý cột headline với stopwords
  cleaned_df['headline'] = cleaned_df['headline'].apply(
      lambda s : ' '.join([re.sub(r'\W+', '', word.lower())
                           for word in s.split(' ') if word not in stopwords]))

  return cleaned_df
```

Hình IV-3

iii. Thư viện SciPy:

- **save_npz(path, X)**

Chức năng: lưu ma trận X kiểu csr_matrix vào đường dẫn path.

- **load_npz(path, X)**

Chức năng: load trận X kiểu csr_matrix từ đường dẫn path.

```
sp.save_npz('D:/Document/Machine_Learning/CS3/boolean_processed_data_pt.npz', bl)
features = sp.load_npz('D:/Document/Machine_Learning/CS3/tf_idf_processed_data_sw.npz')
```

Hình IV-4

iv. Hàm tự định nghĩa:

a. Train model:

- **add_bias(features, labels)**

Chức năng: Thêm cột giá trị 1 (cột bias) vào ma trận features.

Tham số:

features: ma trận N*X với N là số datapoint, với X là số term trong dictionary.

labels: ma trận N*1 với N là số datapoint.

```python
def add_bias(features, labels):
    '''
    features là ma trận N*X với N là số datapoint, X là số term trong dictionary
    labels là ma trận N*1 với N là số datapoint

    Hàm trả về ma trận features(sau khi đã thêm 1 cột bias giá trị 1 cho tất cả các phần tử) và labels
    '''
    # Thêm cột giá trị bias cho dữ liệu
    labels_final = sp.csr_matrix(labels, shape = (1, len(labels)))
    matrix_ones = sp.csc_matrix((26709,1),dtype=np.float)
    matrix_ones[:]=1
    matrix_final_features= sp.csc_matrix(sp.hstack((features,matrix_ones)))
    return matrix_final_features, labels_final
```

Hình IV-5

- **sigmoid(z)**

Chức năng: Trả về giá trị sigmoid của giá trị số thực đầu vào z.

```python
def sigmoid(z):
    # hàm sigmoid
    '''
    z là model đầu ra Linear Regression
    np.exp là e mũ
    Hàm trả về sigmoid của z

    '''
    return 1/(1+np.exp(-z))
```

Hình IV-6

- **Cost_Function(w, y, X)**

Chức năng: Trả về giá trị hàm mất mát tương ứng bộ tham số w.

Tham số:

w: ma trận bộ tham số với kích thước bằng số cột của X.

y: labels của tập dữ liệu.

X: tập features của dữ liệu.

```
def Cost_Function(w, y, X):
    '''
    w là ma trận bộ tham số có shape bằng (1,X.shape[1])
    X là features
    y là y labels
    Hàm trả về cost tương ứng bộ tham số w

    '''
    # tính giá trị đầu ra của hàm Linear Regression
    s = X.dot(w).toarray()
    # scale s về khoảng [0,1] dùng hàm sigmoid
    # z là xác xuất điểm dữ liệu rơi vào lớp y
    z = sigmoid(s)
    y = y.toarray()

    cost = -np.sum((y*(np.log(z)) + (1-y)*np.log(1-z)))/ X.shape[0]

    return cost
```

Hình IV-7

- **Derivative(w, y, X)**

Chức năng: Trả về bộ giá trị đạo hàm của hàm mất mát theo từng giá trị trong ma trận bộ trong số w.

Tham số:

w: ma trận bộ tham số với kích thước bằng số cột của X.

y: labels của tập dữ liệu.

X: tập features của dữ liệu.

```
def Derivative(w, y, X):
    '''
    W là ma trận bộ trọng số w: (28901, 1)  <class 'scipy.sparse.csc.csc_matrix'>
    y là dữ liệu labels y: (1, 1) <class 'scipy.sparse.csc.csc_matrix'>
    X là dữ liệu Train X: (28901, 1) <class 'scipy.sparse.csr.csr_matrix'>

    Hàm trả về bộ giá trị đạo hàm của cost fuction theo từng giá trị trong ma trận w
    '''
    # tính giá trị đầu ra của hàm Linear Regression
    s = X.dot(w).toarray()
    # scale s về khoảng [0,1] dùng hàm sigmoid
    # z là xác xuất điểm dữ liệu rơi vào lớp y
    z = sigmoid(s)
    y = y.toarray()
    der = (z-y).T

    X = X.T
    der = sp.csr_matrix(der, shape = (1,der.shape[1]))
    der = der.T

    return X.dot(der)
```

Hình IV-8

- **Prediction(w, y, X,thred)**

Chức năng: Hàm trả về tập dự đoán dựa trên bộ trọng số w, tập thuộc tính và ngưỡng.

Tham số:

w: ma trận bộ tham số với kích thước bằng số cột của X.

y: labels của tập dữ liệu.

X: tập features của dữ liệu.

thred: ngưỡng xét. Nếu lớn hơn ngưỡng sẽ có label là 1 và ngược lại.

```
def Prediction(w, y, X, thred):
    '''
    W là ma trận bộ trọng số w: (28901, 1)  <class 'scipy.sparse.csc.csc_matrix'>
    y là dữ liệu labels y: (1, 1) <class 'scipy.sparse.csc.csc_matrix'>
    X là dữ liệu Train X: (28901, 1) <class 'scipy.sparse.csr.csr_matrix'>

    Hàm trả về ma trận N*1 với N là số datapoint
    '''
     # tính giá trị đầu ra của hàm Linear Regression
    s = X.dot(w).toarray()
    # scale s về khoảng [0,1] dùng hàm sigmoid
    # z là xác xuất điểm dữ liệu rơi vào lớp y
    z = sigmoid(s)
    for i in range(z.shape[0]):
        if(z[i]>thred):
            z[i] = 1
        else: z[i] = 0
    return z
```

Hình IV-9

- **choose_Thred(w, y, X)**

Chức năng: Hàm trả giá trị ngưỡng giúp tối ưu hóa F1 score.

Tham số:

w: ma trận bộ tham số với kích thước bằng số cột của X.

y: labels của tập dữ liệu.

X: tập features của dữ liệu.

```
def choosen_Thred(w, y_test, X_test):
    f1 = []
    for i in np.arange(0.0,1.0,0.001):
        y_pred = Prediction(w, y_test, X_test, i)
        TP = find_TP(y_pred, y_test.toarray())
        FP = find_FP(y_pred, y_test.toarray())
        P = find_P(y_test.toarray())
        Re = Recall(TP, P)
        Pr = Precision(TP, FP)
        f1.append(F1_Score(Pr, Re))
    axis_x = list(range(1000))
    axis_y = f1
    plt.scatter(axis_x, axis_y)
    plt.show()

    thred = f1.index(max(f1))
    thred = thred/1000
    print(max(f1))
    return thred

thred = choosen_Thred(w, y_test, X_test)
```

Hình IV-10

b. Xử lý dữ liệu:

- **eliminate_stopwords(Data)**

Chức năng: Data kiểu Dataframe (của pandas). Trả về Dataframe khi đã loại bỏ stopwords và các ký tự đặc biệt.

```
def eliminate_stopwords(Data):
  '''
  Data là dữ liệu ban đầu

  Hàm trả về dữ liệu sau khi đã loại bỏ stopwords và các ký tự đặc biệt
  '''
  # Làm sách dữ liệu
  cleaned_df = Data
  # Loại bỏ cột article_link
  cleaned_df.pop('article_link')
  # Loại bỏ những dữ liệu bị thiếu
  cleaned_df.dropna()
  # Load bộ từ điển stopwords trong tiếng anh
  stop = set(stopwords.words('english'))

  # Xử lý cột headline
  cleaned_df['headline'] = Data['headline'].apply(
      lambda s : ' '.join([re.sub(r'\W+', '', word.lower()) for word in
                           s.split(' ') if word not in stop]))

  # Loại bỏ ký tự đặc biệt
  processed_data = cleaned_df['headline'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',' ')
  processed_data = processed_data.str.replace(r'[^\w\d\s]', ' ')
  processed_data = processed_data.str.replace(r'\s+', ' ')
  processed_data = processed_data.str.replace(r'^\s+|\s+?$', ' ')
  processed_data = processed_data.str.replace(r'\d+',' ')
  processed_data = processed_data.str.lower()
  return processed_data
```

Hình IV-11

- **create_list_diction(processed_data)**

Chức năng: processed_data kiểu Dataframe (của pandas). Trả về Dictionary chứa các token đơn trong processed_data.

```
def create_list_diction(processed_data):
  '''
  processed_data là dữ liệu đã qua xử lý loại bỏ các stop words

  Hàm trả về 1 dictionary chứa các từ có trong processed_data

  '''
  # danh sách tất cả các token
  diction = []
  # lặp cho toàn bộ dữ liệu
  for headline in processed_data:
    # tách token
    token = word_tokenize(headline)

    for w  in token:
      diction.append(w)

  # khởi tạo diction từ danh sách token
  diction = nltk.FreqDist(diction)
  words = list(diction.keys())

  # chuyển token sang từ điển
  return words
```

Hình IV-12

- **TF(data, diction, dtype = int)**

Chức năng: Hàm trả về ma trận TF kiểu matrix csr.

Tham số:

data: Kiểu Dataframe, dữ liệu đầu vào.

diction: kiển Dictionary, khi bộ từ điển các token của daa.

dtype:

```
def TF(data, diction, dtype = int):
  '''
  data là dữ liệu đã qua xử lý
  diction là bộ từ điển

  Hàm trả về ma trận thưa csr
  '''
  row = []
  col = []
  val = []
  for i in range(len(data)):
    # tách token mỗi từ trong mỗi hàng
    line = data[i].split()
    # các từ đã thêm
    proceeded_word = set()
    for word in line:
      if word not in proceeded_word:
        try:
          dict_index = diction.index(word)
        except ValueError:
          continue
        row.append(i)
        col.append(dict_index)
        val.append(line.count(word))
        proceeded_word.update(word)

  # khởi tạo và trả về ma trận csr
  row = np.array(row)
  col = np.array(col)
  val = np.array(val)
  tf = csr_matrix((val, (row, col)), shape = (len(data), len(diction)),
                  dtype = dtype)
  return tf
```

Hình IV-13

- **Compute_TF(data, diction, dtype = float)**

Chức năng:

Tham số:

data: Kiểu Dataframe, dữ liệu đầu vào.

diction: kiển Dictionary, khi bộ từ điển các token của daa.

```
def Compute_TF(data, diction, dtype = float):
  '''
  data là dữ liệu đã qua xử lý
  diction là bộ từ điển

  Hàm trả về ma trận thưa sau ma trận thưa tf
  '''
  # khởi tạo ma trận tf
  tf = TF(data, diction, dtype)
  # tính giá trị tf
  for i in range(tf.shape[0]):
    try:
      tf.data[tf.indptr[i]:tf.indptr[i+1]] = tf.data[tf.indptr[i]:tf.indptr[i+1]]/np.max(tf.data[tf.indptr[i]:tf.indptr[i+1]])
    except ValueError:
      pass

  return tf
```

Hình IV-14

- **Compute_TF_IDF(data, diction)**

Chức năng: Hàm trả về ma trận thưa sau khi tf và idf.

Tham số:

data: Kiểu Dataframe, dữ liệu đầu vào.

diction: kiển Dictionary, khi bộ từ điển các token của daa.

```
def Compute_TF_IDF(data, diction):
  '''
  data là dữ liệu đã qua xử lý
  diction là bộ từ điển

  Hàm trả về ma trận thưa sau khi nhân tf và idf
  '''
  # số từ trong từ điển
  features = len(diction)
  # số điểm dữ liệu
  number_of_datapoints = len(data)
  # tính giá trị tf
  tf = Compute_TF(data, diction)
  # chuyển sang ma trận thưa csc
  matrix_csc = tf.tocsc()
  # tính giá trị IDF
  for i in range(features):
    matrix_csc.data[matrix_csc.indptr[i]:matrix_csc.indptr[i+1]] = np.log(number_of_datapoints/(1+ np.count_nonzero(matrix_csc.data[matrix_csc.indptr[i]:matrix_csc.in
  # nhân 2 ma trận và trả về kết quả
  return tf.multiply(matrix_csc.tocsr())
```

Hình IV-15

c. *Đánh giá kết quả:*

- **find_TP(predict, test)**

Chức năng: Tính giá trị TP (True Positive) từ tập dự đoán predict và tập test.

```
# Tìm giá trị TP
def find_TP(predict, test_y):
  _and = np.multiply(test_y, predict)
  return np.count_nonzero(_and)
```

Hình IV-16

- **find_FP(predict, test)**

Chức năng: Tính giá trị FP (False Positive) từ tập dự đoán predict và tập test.

```
# Tìm giá trị FP
def find_FP(predict, test):
  test_y = np.array(test)
  count = 0
  for i in range(predict.shape[0]):
    if predict[i] == 1:
      if test_y[i] == 0:
        count += 1
  return count
```

Hình IV-17

- **find_P(test)**

Chức năng: Tính giá trị P (Positive) từ tập test.

```
# Tính số nhãn bằng 1 trong tập
def find_P(test):
  return np.count_nonzero(test)
```

Hình IV-18

- **F1_score(Precision, Recall)**

Chức năng: Tính giá trị F1_score từ Precision và Recall.

```
# Tính F1_score
def F1_score(Precision, Recall):
  return (Precision*Recall)/(Precision+Recall)*2
```

Hình IV-19

- **Accurancy(y_pred, y_test)**

Chức năng: Tính giá trị Accurancy từ tập dự đoán predict và tập test.

```
# Tính Accurancy
def Accuracy(y_pred, y_test):
  count = 0
  for i in range(y_pred.shape[0]):
    if y_pred[i] == y_test[i]:
      count += 1
  return count/y_pred.shape[0]
```

Hình IV-20

3. Source Code:

i. TF_IDF.py: *(Tiền xử lý dataset code tay)*

```
import pandas as pd
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import string
import nltk
import re
import numpy as np
from scipy.sparse import csr_matrix
from scipy.sparse import spmatrix
from scipy.sparse import coo_matrix
import scipy.sparse as sp

def eliminate_stopwords(Data):
    '''
    Data là dữ liệu ban đầu

    Hàm trả về dữ liệu sau khi đã loại bỏ stopwords và các ký tự đặc biệt
    '''
    # Làm sách dữ liệu
    cleaned_df = Data
    # Loại bỏ cột article_link
    cleaned_df.pop('article_link')
    # Loại bỏ những dữ liệu bị thiếu
    cleaned_df.dropna()
    # Load bộ từ điển stopwords trong tiếng anh
    stop = set(stopwords.words('english'))
```

```
    # Xử lý cột headline
    cleaned_df['headline'] = Data['headline'].apply(lambda s : ' '.join([re.sub(r'\W+', '',
word.lower()) for word in s.split(' ') if word not in stop]))

    # Loại bỏ ký tự đặc biệtact
    processed_data = cleaned_df['headline'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',' ')
    processed_data = processed_data.str.replace(r'[^\w\d\s]', ' ')
    processed_data = processed_data.str.replace(r'\s+', ' ')
    processed_data = processed_data.str.replace(r'^\s+|\s+?$', ' ')
    processed_data = processed_data.str.replace(r'\d+',' ')
    processed_data = processed_data.str.lower()
    return processed_data

def eliminate_punctuation(Data):
    '''
    Data là dữ liệu ban đầu

    Hàm trả về dữ liệu sau khi đã loại bỏ các ký tự đặc biệt
    '''
    # Làm sách dữ liệu
    cleaned_df = Data
    # Loại bỏ cột article_link
    cleaned_df.pop('article_link')
    # Loại bỏ những dữ liệu bị thiếu
    cleaned_df.dropna()

    # # Xử lý cột headline
```

```
    cleaned_df['headline'] = Data['headline'].apply(lambda s : ' '.join([re.sub(r'\W+', '',
word.lower()) for word in s.split(' ')]))

    # Loại bỏ ký tự đặc biệtact
    processed_data = cleaned_df['headline'].str.replace(r'^.+@[^\.].*\.[a-z]{2,}$',' ')
    processed_data = processed_data.str.replace(r'[^\w\d\s]', ' ')
    processed_data = processed_data.str.replace(r'\s+', ' ')
    processed_data = processed_data.str.replace(r'^\s+|\s+?$', ' ')
    processed_data = processed_data.str.replace(r'\d+',' ')
    processed_data = processed_data.str.lower()
    return processed_data

def create_list_diction(processed_data):
    '''
    processed_data là dữ liệu đã qua xử lý loại bỏ các stop words

    Hàm trả về 1 dictionary chứa các từ có trong processed_data

    '''
    # danh sách tất cả các token
    diction = []
    # lặp cho toàn bộ dữ liệu
    for headline in processed_data:
            # tách token
      token = word_tokenize(headline)

      for w  in token:
        diction.append(w)
```

```
    # khởi tạo diction từ danh sách token
    diction = nltk.FreqDist(diction)
    words = list(diction.keys())

    # chuyển token sang từ điển
    return words

def TF(data, diction, dtype = int):
    '''
    data là dữ liệu đã qua xử lý
    diction là bộ từ điển

    Hàm trả về ma trận thưa csr
    '''
    row = []
    col = []
    val = []
    for i in range(len(data)):
            # tách token mỗi từ trong mỗi hàng
            line = data[i].split()
            # các từ đã thêm
            proceeded_word = set()
            for word in line:
                    if word not in proceeded_word:
                            try:
                                    dict_index = diction.index(word)
                            except ValueError:
                                    continue
                            row.append(i)
```

```
                        col.append(dict_index)
                        val.append(line.count(word))
                        proceeded_word.update(word)

    # khởi tạo và trả về ma trận csr
    row = np.array(row)
    col = np.array(col)
    val = np.array(val)
    tf = csr_matrix((val, (row, col)), shape = (len(data), len(diction)), dtype = dtype)
    return tf

def boolean(data, diction, dtype = int):
    '''
    data là dữ liệu đã qua xử lý
    diction là bộ từ điển

    Hàm trả về ma trận thưa csr
    '''
    row = []
    col = []
    val = []
    for i in range(len(data)):
            # tách token mỗi từ trong mỗi hàng
            line = data[i].split()
            # các từ đã thêm
            proceeded_word = set()
            for word in line:
                    if word not in proceeded_word:
                            try:
```

```
                    dict_index = diction.index(word)
                except ValueError:
                    continue
                row.append(i)
                col.append(dict_index)
                val.append(1)
                proceeded_word.update(word)

    # khởi tạo và trả về ma trận csr
    row = np.array(row)
    col = np.array(col)
    val = np.array(val)
    boolean = csr_matrix((val, (row, col)), shape = (len(data), len(diction)), dtype =
dtype)
    return boolean

def Compute_TF(data, diction, dtype = float):
    '''
    data là dữ liệu đã qua xử lý
    diction là bộ từ điển

    Hàm trả về ma trận thưa tf
    '''
    # khởi tạo ma trận tf
    tf = TF(data, diction, dtype)
    # tính giá trị tf
    for i in range(tf.shape[0]):
        try:
```

```
                    tf.data[tf.indptr[i]:tf.indptr[i+1]] =
tf.data[tf.indptr[i]:tf.indptr[i+1]]/np.max(tf.data[tf.indptr[i]:tf.indptr[i+1]])
            except ValueError:
                    pass

    return tf

def Compute_TF_IDF(data, diction):
    '''
    data là dữ liệu đã qua xử lý
    diction là bộ từ điển

    Hàm trả về ma trận thưa sau khi nhân tf và idf
    '''
    # số từ trong từ điển
    features = len(diction)
    # số điểm dữ liệu
    number_of_datapoints = len(data)
    # tính giá trị tf
    tf = Compute_TF(data, diction)
    # chuyển sang ma trận thưa csc
    matrix_csc = tf.tocsc()
    # tính giá trị IDF
    for i in range(features):
            matrix_csc.data[matrix_csc.indptr[i]:matrix_csc.indptr[i+1]] =
np.log(number_of_datapoints/(1+
np.count_nonzero(matrix_csc.data[matrix_csc.indptr[i]:matrix_csc.indptr[i+1]])))
    # nhân 2 ma trận và trả về kết quả
    return tf.multiply(matrix_csc.tocsr())
```

```
def Compute_IDF(data, diction):
    '''
    data là dữ liệu đã qua xử lý
    diction là bộ từ điển

    Hàm trả về ma trận thưa idf
    '''
    # số từ trong từ điển
    features = len(diction)
    # số điểm dữ liệu
    number_of_datapoints = len(data)
    # tính giá trị tf
    tf = Compute_TF(data, diction)
    # chuyển sang ma trận thưa csc
    matrix_csc = tf.tocsc()
    # tính giá trị IDF
    for i in range(features):
            matrix_csc.data[matrix_csc.indptr[i]:matrix_csc.indptr[i+1]] =
np.log(number_of_datapoints/(1+
np.count_nonzero(matrix_csc.data[matrix_csc.indptr[i]:matrix_csc.indptr[i+1]])))
    # nhân 2 ma trận và trả về kết quả
    return (matrix_csc.tocsr())

def main():
    # đọc file
    path = 'D:/Document/Machine_Learning/CS3/Sarcasm_Headlines_Dataset.json'
    Data = pd.read_json(path, lines = True)
```

```
    ## Loại bỏ stopwords và punctuation


    processed_data_stopwords = eliminate_stopwords(Data)
    # khởi từ điển
    diction = create_list_diction(processed_data_stopwords[:20000])
    # tính boolean
    bl = boolean(processed_data_stopwords, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/boolean_processed_data_sw.
npz', bl)
    # tính tf
    tf = Compute_TF(processed_data_stopwords, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/tf_processed_data_sw.npz',
tf)
    # tính idf
    idf = Compute_IDF(processed_data_stopwords, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/idf_processed_data_sw.npz',
idf)
    # tính tfidf
    tf_idf = Compute_TF_IDF(processed_data_stopwords, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/tf_idf_processed_data_sw.np
z', tf_idf)




    # loại bỏ punctuation


    processed_data_punctuation = eliminate_punctuation(Data)
    # khởi từ điển
    diction = create_list_diction(processed_data_punctuation[:20000])
```

```
    # tính boolean
    bl = boolean(processed_data_punctuation, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/boolean_processed_data_pt.
npz', bl)
    # tính tf
    tf = Compute_TF(processed_data_punctuation, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/tf_processed_data_pt.npz',
tf)
    # tính idf
    idf = Compute_IDF(processed_data_punctuation, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/idf_processed_data_pt.npz',
idf)
    # tính tfidf
    tf_idf = Compute_TF_IDF(processed_data_punctuation, diction)
    sp.save_npz('D:/Document/Machine_Learning/CS3/tf_idf_processed_data_pt.npz
', tf_idf)


if __name__ == '__main__':
    main()
```

ii. Evaluate.py: (*Hàm đánh giá code tay*)

```
import numpy as np

# Tìm giá trị TP
def find_TP(y_pred, y_test):
   count = 0
   Positive = np.multiply(y_pred, y_test)
   return np.count_nonzero(Positive)

def Recall(TP, P):
   return 1.0*TP/P
```

```
def Precision(TP, FP):
    return 1.0*TP/(TP+FP)

# tính tổng số điểm dữ liệu dự đoán nhãn là 1 nhưng sai
def find_FP(y_pred, y_test):
    count = 0
    for i in range(y_pred.shape[0]):
        if y_pred[i] == 1:
            if y_test[i] == 0:
                count += 1
    return count

# tính tổng số điểm dữ liệu có nhãn là 1 trong tập test
def find_P(y_test):
    return np.count_nonzero(y_test)

# Tính F1_score
def F1_score(Precision, Recall):
    return (Precision*Recall)/(Precision+Recall)*2

# Tính Accurancy
def Accuracy(y_pred, y_test):
    count = 0
    for i in range(y_pred.shape[0]):
        if y_pred[i] == y_test[i]:
            count += 1
    return count/y_pred.shape[0]
```

iii. Predicting_By_Function.py: *(Naïve Bayes + Decision Tree + Logistic Regression + K-nearest Neighbors dùng thư viện Sklearn)*

```
import numpy as np
import pandas as pd


from Evaluate import *


from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import BernoulliNB
```

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.feature_extraction.text import TfidfVectorizer

import scipy.sparse



# vector hóa train set
def Train(train_X, train_y, test_X, type_train=None):
    # khởi tạo model cho từng loại phân loại
    if type_train == 'LO':
            logist = LogisticRegression()
            logist.fit(train_X, train_y)
            reg_predicted_logist = logist.predict(test_X)
            return reg_predicted_logist
    elif type_train == 'DC':
            DC = DecisionTreeClassifier()
            DC.fit(train_X, train_y)
            reg_predicted_DC = DC.predict(test_X)
            return reg_predicted_DC
    elif type_train == 'NB':
            NB = BernoulliNB()
            NB.fit(train_X.toarray(), train_y)
            reg_predicted_NB = NB.predict(test_X.toarray())
            return reg_predicted_NB
    elif type_train == 'KNN':
            KNN = KNeighborsClassifier(n_neighbors=10, weights='distance')
            KNN.fit(train_X, train_y)
            reg_predicted_KNN = KNN.predict(test_X.toarray())
```

```
            return reg_predicted_KNN

def main():

    label = np.load('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/label.npy')

    data = pd.read_json("C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/Sarcasm_Headlines_Dataset.json", lines = True)
    title = data['headline']

    text_transformer = TfidfVectorizer()

    train_X = text_transformer.fit_transform(title[:20000])
    train_y = label[:20000]
    test_X = text_transformer.transform(title[20000:])
    test_y = label[20000:]

    reg_predicted = Train(train_X, train_y, test_X, 'KNN')

    TP = find_TP(reg_predicted, test_y)
    FP = find_FP(reg_predicted, test_y)
    P = find_P(test_y)
    Ac = Accuracy(reg_predicted, test_y)

    print("TP = ", TP)
    print("FP = ", FP)
    print("P  = ", P)
```

```
    Precision = TP/(TP+FP)
    Recall = TP/P

    print("Accuracy = ", Ac)

    print("Precision = ", Precision)
    print("Recall = ", Recall)

    print("F1_score:", F1_score(Precision, Recall))

if __name__ == '__main__':
    main()
```

*iv. *.ipynb trong thư mục final: (Logistic Regression code tay)*

a. Xử lý và phân chia dữ liệu:

```
# import thư viện numpy
import numpy as np
# import thư viện matplotlib
import matplotlib.pyplot as plt

import scipy.sparse as sp

import math

from numpy import linalg as LA

# Tải dữ liệu data X và y đã được xử lý thành ma trận
```

```
labels = np.load('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-master/label.npy')
# dữ liệu bình thường dùng hàm tách
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/tfidf.npz')

# dữ liệu tf_idf loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/tf_idf_processed_data_sw.npz')
# dữ liệu tf_idf không loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/tf_idf_processed_data_pt.npz')

# dữ liệu boolean loại bỏ stop words
features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/boolean_processed_data_sw.npz')
# dữ liệu boolean không loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/boolean_processed_data_pt.npz')

# dữ liệu tf loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/tf_processed_data_sw.npz')
# dữ liệu tf không loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/tf_processed_data_pt.npz')

# dữ liệu idf loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-
master/PreProcessed_Data/idf_processed_data_sw.npz')
```

```
# dữ liệu idf không loại bỏ stop words
# features = sp.load_npz('C:/Users/Admin/Downloads/NLTK_is_Sarcastic-master/PreProcessed_Data/idf_processed_data_pt.npz')

def add_bias(features, labels):
    '''
    features là ma trận N*X với N là số datapoint, X là số term trong dictionary
    labels là ma trận N*1 với N là số datapoint

    Hàm trả về ma trận features(sau khi đã thêm 1 cột bias giá trị 1 cho tất cả các phần tử) và labels
    '''
    # Thêm cột giá trị bias cho dữ liệu
    labels_final = sp.csr_matrix(labels, shape = (1, len(labels)))
    matrix_ones = sp.csc_matrix((26709,1),dtype=np.float)
    matrix_ones[:]=1
    matrix_final_features= sp.csc_matrix(sp.hstack((features,matrix_ones)))
    return matrix_final_features, labels_final

# chia Train và test
matrix_final_features, labels_final = add_bias(features, labels)
X_Train = matrix_final_features[:20000]
X_test = matrix_final_features[20000:]
y_Train = labels_final.T[:20000]
y_test = labels_final.T[20000:]
```

b. *Định nghĩa hàm sigmoid:*

```
def sigmoid(z):
    # hàm sigmoid
```

```
    '''
    z là model đầu ra Linear Regression
    np.exp là e mũ
    Hàm trả về sigmoid của z

    '''
    return 1/(1+np.exp(-z))
```

c. *Định nghĩa hàm cost:*

```
def Cost_Function(w, y, X):
    '''
    w là ma trận bộ tham số có shape bằng (1,X.shape[1])
    X là features
    y là y labels
    Hàm trả về cost tương ứng bộ tham số w

    '''
    # tính giá trị đầu ra của hàm Linear Regression
    s = X.dot(w).toarray()
    # scale s về khoảng [0,1] dùng hàm sigmoid
    # z là xác xuất điểm dữ liệu rơi vào lớp y
    z = sigmoid(s)
    y = y.toarray()

    cost = -np.sum((y*(np.log(z)) + (1-y)*np.log(1-z)))/ X.shape[0]

    return cost
```

d. *Tính đạo hàm:*

```
def Derivative(w, y, X):
```

```
    '''
    W là ma trận bộ trọng số w: (28901, 1)  <class 'scipy.sparse.csc.csc_matrix'>
    y là dữ liệu labels y: (1, 1) <class 'scipy.sparse.csc.csc_matrix'>
    X là dữ liệu Train X: (28901, 1) <class 'scipy.sparse.csr.csr_matrix'>

    Hàm trả về bộ giá trị đạo hàm của cost fuction theo từng giá trị trong ma trận w
    '''
    # tính giá trị đầu ra của hàm Linear Regression
    s = X.dot(w).toarray()
    # scale s về khoảng [0,1] dùng hàm sigmoid
    # z là xác xuất điểm dữ liệu rơi vào lớp y
    z = sigmoid(s)
    y = y.toarray()
    der = (z-y).T

    X = X.T
    der = sp.csr_matrix(der, shape = (1,der.shape[1]))
    der = der.T

    return X.dot(der)
```

e. Hàm dự đoán:

```
def Prediction(w, y, X, thred):
    '''
    W là ma trận bộ trọng số w: (28901, 1)  <class 'scipy.sparse.csc.csc_matrix'>
    y là dữ liệu labels y: (1, 1) <class 'scipy.sparse.csc.csc_matrix'>
    X là dữ liệu Train X: (28901, 1) <class 'scipy.sparse.csr.csr_matrix'>

    Hàm trả về ma trận N*1 với N là số datapoint
    '''
     # tính giá trị đầu ra của hàm Linear Regression
    s = X.dot(w).toarray()
    # scale s về khoảng [0,1] dùng hàm sigmoid
```

```
    # z là xác xuất điểm dữ liệu rơi vào lớp y
    z = sigmoid(s)
    for i in range(z.shape[0]):
        if(z[i]>thred):
            z[i] = 1
        else: z[i] = 0
    return z
```

f. Dùng BGD để train model:

```
# Khởi tạo learning rate
alpha = 0.0005
# Khởi tạo ma trận biến với giá trị khởi tạo tại mỗi phần tử là 0
w = np.zeros((X_Train.shape[1], 1))
w = sp.csc_matrix(w, shape = (len(w),1))
# list chứa tất cả mse
all_Cost = []

# Khởi tạo số vòng lặp
i = 0
# Huấn luyện
while True:
    # Tính cost với w hiện thời
    err = Cost_Function(w, y_Train, X_Train)
    all_Cost.append(err)
    # Cập nhật w theo công thức gradient descent
    der = Derivative(w, y_Train, X_Train)
    w = w - alpha*der

    if i % 500 == 0:
        # In ra cost tại vòng lặp thứ i
        print("epoch = ", i, "cost = ",Cost_Function(w, y_Train, X_Train))
```

```
        i += 1
        if math.isnan(float(err)) or (LA.norm(der.toarray())/ der.shape[0]) == 0:
            break


    # kiểm tra trên tập test
    print("Cost trên tập test_data:", Cost_Function(w, y_test, X_test))
    # Vẽ đồ thị biểu diễn cost qua từng vòng lặp
    axis_x = list(range(i))
    axis_y = all_Cost
    plt.scatter(axis_x, axis_y)
    plt.show()
```

4. Demo kết quả chụp màn hình:

i. Predict_By_Function.py: *(Các model được xây dựng và train bằng hàm có sẵn)*

- **K-nearest-neighbor: với K = 10 và weight = 'distance'**

Precision = 0.8262

Recall = 0.5962

F1_Score = 0.6928

Accurancy = 0.7612

```
(NEW) C:\Users\Admin>python C:\Users\Admin\Downloads\NLTK_is_Sarcastic-master\Predicting_By_Function.py
Accuracy =  0.7691161126844537
Precision =  0.8267865593942262
Recall =  0.596245733788396
F1_score: 0.692841562561967
```

Hình IV-21

- **Decision Tree**

Precision = 0.7031

Recall = 0.7072

F1_Score = 0.7051

Accurancy = 0.7417

```
Accuracy =  0.7416902668057833
Precision =  0.7030878859857482
Recall =  0.7071672354948806
F1_score: 0.7051216607112473
```

Hình IV-22

- **Naïve Bayes: sử dụng phương pháp Bernoulli Naïve Bayes**

Precision = 0.8652

Recall = 0.7519

F1_Score = 0.8046

Accurancy = 0.8405

```
(NEW) C:\Users\Admin>python C:\Users\Admin\Downloads\NLTK_is_Sarcastic-master\Predicting_By_Function.py
Accuracy =  0.840512744075123
Precision =  0.8652788688138257
Recall =  0.7518771331058021
F1_score: 0.8046018991964939
```

Hình IV-23

- **Logistic Regression**

Precision = 0.8303

Recall = 0.8034

F1_Score = 0.8167

Accurancy = 0.8424

```
Accuracy =  0.8424504397078552
Precision =  0.8303350970017637
Recall =  0.8034129692832764
F1_score: 0.816652211621856
```

Hình IV-24

ii. Logistic_Regression.ipynb: *(model logistic regression được xây dựng và train bằng code tay)*

- **Logistic Regression (Boolean + NonStopWord)**

Evaluate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.8214338947682218
Precision score = 0.8167520117044623
Recall score: 0.7621160409556313
F1_score: 0.7884887005649718
```

Hình IV-25

- **Logistic Regression (Boolean + StopWord)**

Evaluate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.7791026978685348
Precision score = 0.7788906009244992
Recall score: 0.6901023890784983
F1_score: 0.731813246471227
```

Hình IV-26

- **Logistic Regression (IDF + NonStopWord)**

Valuate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.8211357877478015
Precision score = 0.8184830633284241
Recall score: 0.758703071672355
F1_score: 0.7874601487778958
```

Hình IV-27

- **Logistic Regression (IDF + StopWord)**

Valuate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.7776121627664332
Precision score = 0.7771781033153431
Recall score: 0.6880546075085324
F1_score: 0.729905865314989
```

Hình IV-28

- **Logistic Regression (TF + NonStopWord)**

Evaluate

```
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_Score(Pr, Re))
```

```
Accuracy =  0.8257564465643166
Precision score = 0.8152524167561761
Recall score: 0.7771331058020478
F1_score: 0.795736501834702
```

Hình IV-29

- **Logistic Regression (TF + StopWord)**

Evaluate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.7773140557460129
Precision score = 0.7659259259259259
Recall score: 0.70580204778157
F1_score: 0.7346358792184724
```

Hình IV-30

- **Logistic Regression (TF-IDF + NonStopWord)**

Evaluate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.8239678044417946
Precision score = 0.8171926006528836
Recall score: 0.7689419795221843
F1_score: 0.7923333919465447
```

Hình IV-31

- **Logistic Regression (TF-IDF + StopWord)**

Valuate

```
from Evaluate import *
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_score(Pr, Re))
```

```
Accuracy =  0.7759725741541214
Precision score = 0.7659336563548267
Recall score: 0.7013651877133106
F1_score: 0.7322287546766436
```

Hình IV-32

- **Logistic Regression (TF + StopWord + Chọn ngưỡng hợp lý)**

```
y_pred = Prediction(w, y_test, X_test, thred)
TP = find_TP(y_pred, y_test.toarray())
FP = find_FP(y_pred, y_test.toarray())
P = find_P(y_test.toarray())
Re = Recall(TP, P)
Pr = Precision(TP, FP)
print('Accuracy = ',Accuracy(y_pred, y_test.toarray()))
print("Precision score =", Pr)
print("Recall score:", Re)
print("F1_score:",F1_Score(Pr, Re))
```

```
Accuracy =  0.8262036070949471
Precision score = 0.7845161290322581
Recall score: 0.8300341296928327
F1_score: 0.8066334991708125
```

Hình IV-33

V. References (Tài liệu tham khảo)

(1) Precision và Recall

(2) K-nearesr Neighbors(KNN)

(3) Decision Tree(DC)

(4) Naive Bayes(NB)

(5) Giới thiệu thư viện Numpy

(6) KNN bằng sk-learn

(7) NB Bernoulli bằng sk-learn

(8) DC bằng sk-learn

(9) Giới thiệu Linear Regression(LR)

(10) LR bằng sk-learn

(11) TF-IDF model

(12) Read .json

(13) Pop dataframe .json

(14) Dropna datafame .json

(15) Apply dataframe .json

(16) Ma trận thưa sparse

--//--

CASE STUDY #3

LOGISTIC REGRESSION – SARCASM DETECTION

Giảng viên: Huỳnh Thị Thanh Thương

Sinh viên thực hiện: 17520324 – Nguyễn Thành Danh

17521244 – Hồ Sỹ Tuyến

17520828 – Phan Nguyên