TRƯỜNG ĐẠI HỌC THỦY LỢI

BÁO CÁO BÀI TẬP LỚN
HỌC PHẦN PHÂN TÍCH DỮ LIỆU LỚN

DỰ ĐOÁN BỆNH TRẦM CẢM CỦA SINH VIÊN

NGƯỜI HƯỚNG DẪN: Tạ Quang Chiểu

NHÓM 10

Họ và tên	MSV
Lương Tuấn Anh	2251262271
Phan Anh	2351260639
Trần Thị Vân Anh	2251262576
Nguyễn Gia Bách	2251262579

Hà Nội, Năm 2025

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT VÀ GIẢI THÍCH

1. **CGPA (Cumulative Grade Point Average):** là một hệ thống tính điểm trung bình tích lũy được sử dụng phổ biến trong giáo dục để đánh giá kết quả học tập của sinh viên trong một khoảng thời gian nhất định

- 1: 0 – 7 điểm
- 2: 7 – 8.5 điểm
- 3: 8.5 - 10

2. **Age: Tuổi**

- 1: 18 đến 25 tuổi
- 2: 26 đến 39 tuổi
- 3: 40 đến 59 tuổi

3. **Academic_Pressure: Áp lực học đường**

- 0: Không áp lực
- 1: Rất ít áp lực
- 2: Áp lực nhẹ
- 3: Áp lực trung bình
- 4: Áp lực cao
- 5: Áp lực cực cao

4. **Study_Satisfaction: Mức độ hài lòng trong học tập**

- 0: Hoàn toàn không hài lòng
- 1: Rất không hài lòng
- 2: Ít hài lòng
- 3: Trung bình
- 4: Hài lòng
- 5: Hoàn toàn hài lòng

5. **Financial_stress: Áp lực tài chính**

- 1: Không căng thẳng về tài chính
- 2: Căng thẳng rất thấp
- 3: Căng thẳng trung bình
- 4: Căng thẳng cao
- 5: Căng thẳng rất cao

1. CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN, GIỚI THIỆU TỔNG QUAN

1.1. Giới thiệu bài toán

- Trong bối cảnh xã hội hiện đại, vấn đề sức khỏe tâm lý, đặc biệt là trầm cảm đang ngày càng nhận được sự quan tâm từ các tổ chức y tế và cộng đồng. Trầm cảm không chỉ ảnh hưởng tiêu cực đến chất lượng cuộc sống mà còn dẫn đến các hệ lụy lâu dài nếu không được phát hiện và can thiệp kịp thời.
- Ngày nay, với sự phát triển mạnh mẽ của công nghệ, việc ứng dụng các phương pháp học máy để phân tích dữ liệu và dự đoán trầm cảm cho các sinh viên trở thành một giải pháp tiềm năng. Điều này không chỉ giúp nâng cao nhận thức về sức khỏe tâm lý mà còn hỗ trợ các biện pháp kịp thời để tránh khỏi việc các sinh viên bị trầm cảm.
- Với vấn đề này, nhóm chúng em sử dụng thuật toán Naïve Bayes và Logistic Regression để dự đoán khả năng mắc bệnh trầm cảm (Depression) dựa trên các thuộc tính liên quan trong các dữ liệu đã thu thập được

1.2. Tổng quan bài toán

- Dữ liệu trong bài toán bao gồm 12 thuộc tính quan trọng:
- Gender: Giới tính của sinh viên
- Age: Tuổi của sinh viên
- Academic_Pressure: Mức độ áp lực học tập
- CGPA: Điểm trung bình tích lũy
- Study_Sastifaction: Mức độ hài lòng với quá trình học tập
- Sleep_Duration: Thời gian ngủ mỗi ngày
- Dietary_Habits: Thói quen ăn uống
- Have_you_ever_had_suicidal_thoughts: Từng có suy nghĩ tự tử hay không
- Work/Study_Hours: Thời gian làm việc/ học tập mỗi ngày
- Financial_Stress: Áp lực tài chính
- Family_History_of_Mental_Illness: Tiền sử gia đình mắc bệnh tâm lý
- Depression: Trầm cảm
- Mục tiêu của bài toán là phân loại sinh viên vào 2 nhóm:
- Có nguy cơ mắc trầm cảm
- Không có nguy cơ mắc trầm cảm

2. CHƯƠNG 2: TRÌNH BÀY LÝ THUYẾT

2.1. Naïve Bayes

- Naïve Bayes là một thuật toán phân loại dựa trên định lý Bayes và giả định rằng các đặc trưng (features) trong dữ liệu là độc lập với nhau. Dù giả định này thường không hoàn toàn đúng trong thực tế, thuật toán Naïve Bayes vẫn cho hiệu suất tốt trong nhiều bài toán phân loại.
- Ý tưởng cơ bản của cách tiếp cận Naive Bayes là sử dụng xác suất có điều kiện giữa từ và chủ đề để dự đoán xác suất của một văn bản cần phân loại
- Được xem là thuật toán đơn giản nhất trong các phương pháp

2.1.1. Naïve Bayes Classifier

- Dữ liệu cần có:
- D: Tập dữ liệu huấn luyện, được vector hoá dưới dạng $x = (x_1, x_2,..., x_n)$
- C_i: tập các tài liệu của D thuộc lớp C_i với i = {1,2,3,...}
- Các thuộc tính $x_1, x_2,..., x_3$ độc lập xác suất đôi một với nhau
- Bước 1: Huấn luyện Naive Bayes (dựa vào tập dữ liệu)
  - Tính xác xuất $P(C_i)$
  - Tính xác xuất $P(x_k|C_i)$
- Bước 2: Phân lớp $Xnew$
  - Tính $P(Ci\ |xnew)$
  - X_{new} được gán vào lớp C_q sao cho: $P(C_q|x_{new}) = max(P(C_i\ |x_{new}))$
- Có 3 loại tính toán $p(x_i\ |c)$ là: Gaussian Naive Bayes, Multinomial Naive Bayes, và Bernoulli Naive .

2.1.2. Gaussian Naïve Bayes

- Mô hình này được sử dụng chủ yếu trong loại dữ liệu mà các thành phần là các biến liên tục.
- Với mỗi chiều dữ liệu ii và một class c, x_i tuân theo một phân phối chuẩn có kỳ vọng μ_{ci} và phương sai σ_{ci}^2

$$p(x_i|c) = p(x_i|\mu_{ci}, \sigma_{ci}^2) = \frac{1}{\sqrt{2\pi\sigma_{ci}^2}} \exp\left(-\frac{(x_i - \mu_{ci})^2}{2\sigma_{ci}^2}\right)$$

2.1.3. Multinomial Naïve Bayes

- Mô hình này chủ yếu được sử dụng trong phân loại văn bản mà feature vectors được tính bằng Bags of Words. Lúc này, mỗi văn bản được biểu diễn bởi một vector có độ dài d chính là 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. Giá trị này có thể được tính bằng cách:

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

Trong đó:

- N_{ci} là tổng số lần từ thứ ii 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. Nói cách khác, nó bằng tổng độ dài của toàn bộ các văn bản thuộc vào class c. Có thể suy ra rằng

$$N_c = \sum_{i=1}^{d} N_{ci}, \text{ từ đó } \sum_{i=1}^{d} \lambda_{ci} = 1.$$

Cách tính này có một hạn chế là nếu có một từ mới chưa bao giờ xuất hiện trong class cc thì biểu thức sẽ bằng 0, điều này dẫn đến vế phải bằng 0 bất kể các giá trị còn lại có lớn thế nào. Việc này sẽ dẫn đến kết quả không chính xác.
Để giải quyết việc này, một kỹ thuật được gọi là *Laplace smoothing* được áp dụng:

$$\hat{\lambda}_{ci} = \frac{N_{ci} + \alpha}{N_c + d\alpha}$$

2.1.4. Bernoulli Naïve Bayes

- Mô hình này được áp dụng cho các loại dữ liệu mà mỗi thành phần là một giá trị binary bằng 0 hoặc 1. Ví dụ: cũng với loại văn bản nhưng thay vì đếm tổng số lần xuất hiện của 1 từ trong văn bản, ta chỉ cần quan tâm từ đó có xuất hiện hay không.
- Khi đó, p(xi|c) được tính bằng:

$$p(x_i|c) = p(i|c)^{x_i}(1 - p(i|c)^{1-x_i}$$

với p(i|c) có thể được hiểu là xác suất từ thứ i xuất hiện trong các văn bản của class c.

2.1.5. Ưu và nhược điểm của Naïve Bayes

2.1.5.1. Ưu điểm

- Nhanh và hiệu quả
- Đơn giản và dễ triển khai
- Hoạt động tốt với dữ liệu phân loại

2.1.5.2. Nhược điểm

- Naive Bayes cũng có vấn đề khi xử lý lượng dữ liệu lớn.
- Trong quá trình học (training), nếu số lượng dữ liệu quá lớn, dẫn đến các vấn đề thiếu bộ nhớ, tốc độ xử lý
- Với lượng dữ liệu lớn (khoảng vài triệu bản ghi) thì hầu hết thời gian của Naive Bayes là đếm số lần xuất hiện của các biến => tính các xác xuất cần thiết để xây dựng mô hình

2.2. Logistic Regression

- Logistic Regression là một thuật toán học có giám sát (supervised learning) được sử dụng phổ biến cho các bài toán phân loại nhị phân (binary classification). Mục tiêu của Logistic Regression là dự đoán xác suất một mẫu thuộc vào một trong hai lớp (ví dụ: "1" hoặc "0").
- Sử dụng hàm sigmoid để đưa ra dự đoán xác suất nằm trong khoảng [0,1], giúp dễ dàng phân loại sinh viên thuộc hai nhóm:
    - 1: Có dấu hiệu trầm cảm.
    - 0: Không có dấu hiệu trầm cảm.

2.2.1. Hàm hồi quy tuyến tính

- Logistic Regression dựa trên ý tưởng từ Linear Regression:
  $z = w0 + w1*x1 + w2*x2 + \cdots + wn*xn = wT.xz$
- Trong đó:

    - z: Điểm tổng hợp (logit).
    - w=[w0,w1,...,wn] : Trọng số.
    - x=[1,x1,...,xn]: Vector đặc trưng (bao gồm hằng số 1 cho bias).

2.2.2. Hàm sigmoid

$$f(s) = \frac{1}{1+e^{-s}} \triangleq \sigma(s)$$

- **Mục đích:**
  - Hàm sigmoid chuyển đổi giá trị đầu ra z (từ tổng có trọng số của các đặc trưng) thành xác suất nằm trong khoảng [0,1].
  - Công thức: sigmoid(z) = 1 / (1 + np.exp(-z))
- **Ý nghĩa:**
  - Dùng để dự đoán xác suất
  - Hàm sigmoid được sử dụng nhiều nhất, vì nó bị chặn trong khoảng (0, 1).
  - Thêm nữa, lim s→−∞ = 0; lim s→+∞= 1

2.2.3. Hàm mất mát và phương án tối ưu

- Ta có hàm số:

$$J(\mathbf{w}) = -\frac{1}{N}\log p(\mathbf{y}|\mathbf{X};\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log z_i + (1-y_i)\log(1-z_i))$$

- Trong đó:
  - zi: Xác suất dự đoán.
  - yi: Nhãn thực tế (0 hoặc 1).
  - N: Số lượng mẫu trong tập huấn luyện.
- Hàm số này chính là hàm mất mát của Logistic Regression. Ta cần đi tìm w để J(w) đạt giá trị nhỏ nhất

2.2.4. Tối ưu hàm mất mát

- Mục tiêu: Tìm các trọng số w tối ưu để tối thiểu hóa hàm mất mát L(w).
- Phương pháp Gradient Descent: Gradient Descent được sử dụng để cập nhật w theo hướng giảm dần giá trị của L(w)

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(z_i - y_i)\mathbf{x}_i = \mathbf{w} - \eta(\sigma(\mathbf{w}^T\mathbf{x}_i) - y_i)\mathbf{x}_i$$

2.2.5. Quy trình huấn luyện Logistic Regression

Bước 1: Chuẩn hóa và tiền xử lý dữ liệu

- Chuẩn hóa dữ liệu để các đặc trưng có cùng thang đo, giúp thuật toán hội tụ nhanh hơn.

Bước 2: Khởi tạo trọng số

- Khởi tạo trọng số w = [w0, w1, ..., wn] với giá trị ban đầu (thường là 0).

Bước 3: Tối ưu hóa bằng Gradient Descent

- Tính giá trị dự đoán h(w) cho từng mẫu.
- Tính gradient của hàm mất mát.
- Cập nhật trọng số.

Bước 4: Dừng lặp

- Thuật toán dừng khi:
  - Số lần lặp đạt ngưỡng tối đa.
  - Gradient trở nên rất nhỏ (hội tụ).

Bước 5: Dự đoán và đánh giá

- Sử dụng mô hình để dự đoán trên tập kiểm tra (test set).
- Đánh giá hiệu suất bằng các chỉ số như Accuracy, Precision, Recall, F1-Score.

2.2.6. Ưu điểm Logistic Regression

- Hiệu quả và dễ triển khai:
  - Logistic Regression đơn giản, dễ huấn luyện và không yêu cầu tài nguyên tính toán cao.
- Giải thích được kết quả:
  - Trọng số của các đặc trưng (features) trong Logistic Regression dễ hiểu, cho phép phân tích tác động của từng yếu tố đến kết quả dự đoán.
- Khả năng mở rộng:
  - Logistic Regression hoạt động tốt với dữ liệu vừa và nhỏ, đồng thời dễ tích hợp với các kỹ thuật tiền xử lý như mã hóa dữ liệu và chuẩn hóa.

2.2.7. Phương pháp dùng để chuyển đổi value phục vụ cho thuật toán

2.2.7.1. Bag of Words (BoW)

- Bag of Words (BoW) là một phương pháp đơn giản và phổ biến trong xử lý ngôn ngữ tự nhiên (NLP) để chuyển đổi văn bản thành dạng số liệu mà máy tính có thể hiểu và xử lý.
- Khái niệm cơ bản:

  Trong mô hình Bag of Words, một văn bản (hoặc câu, đoạn văn) sẽ được chuyển thành một tập hợp các từ (các từ riêng biệt trong văn bản) mà không quan tâm đến ngữ pháp hoặc thứ tự của các từ. Mỗi từ trong văn bản sẽ được đại diện bằng một đặc trưng số, thể hiện tần suất xuất hiện của nó trong văn bản.

- Đặc điểm chính của BoW:
    - Không lưu trữ ngữ pháp: Mô hình BoW không quan tâm đến trật tự của các từ trong văn bản, chỉ ghi nhận tần suất xuất hiện của từ.
    - Đơn giản và dễ hiểu: Đây là một trong những mô hình cơ bản và dễ sử dụng trong NLP.
    - Không lưu trữ ngữ nghĩa: Mô hình này chỉ tập trung vào tần suất các từ mà không quan tâm đến ngữ nghĩa của chúng.
    - Vấn đề với từ đồng nghĩa: Từ đồng nghĩa như "happy" và "joyful" sẽ được coi là hai từ khác nhau, mặc dù chúng có thể mang cùng một ý nghĩa.
- Ưu điểm:
    - Đơn giản và dễ thực hiện.
    - Khả năng biểu diễn văn bản dưới dạng số: Thích hợp để sử dụng trong các thuật toán máy học.
- Nhược điểm:
    - Không lưu trữ ngữ nghĩa: BoW không lưu lại thông tin ngữ nghĩa của các từ.
    - Lượng tính toán lớn với văn bản dài: Nếu từ điển quá lớn (do có quá nhiều từ trong văn bản), bộ dữ liệu sẽ trở nên rất thưa thớt và dễ dẫn đến việc phải xử lý một lượng tính toán khổng lồ.
    - Không phân biệt từ điển: Các từ giống nhau nhưng có cách viết khác nhau (như "running" và "run") sẽ được coi là khác nhau.

2.2.7.2. Binary Encoding

- Binary Encoding là một phương pháp mã hóa dữ liệu phân loại (categorical data) thành dạng số học, thường được sử dụng trong học máy (machine learning) để biến các giá trị phân loại thành các đại diện số, giúp thuật toán có thể xử lý dữ liệu dễ dàng hơn.
- Khái niệm cơ bản của Binary Encoding:
  Binary Encoding là một phương pháp mã hóa các giá trị phân loại thành các dãy nhị phân (binary strings). Các giá trị trong cột phân loại sẽ được chuyển thành số nguyên, sau đó mỗi số nguyên sẽ được chuyển thành dãy nhị phân tương ứng. Mỗi bit của dãy nhị phân sẽ trở thành một cột riêng biệt.
- Ưu điểm của Binary Encoding:
    - Hiệu quả hơn One-Hot Encoding:
    - One-Hot Encoding có thể tạo ra quá nhiều cột nếu có quá nhiều giá trị phân loại. Trong khi đó, Binary Encoding chỉ tạo ra số lượng cột ít hơn, vì mỗi giá trị được mã hóa thành một dãy nhị phân có độ dài nhỏ hơn.
    - Tối ưu bộ nhớ: Vì sử dụng ít cột hơn One-Hot Encoding, Binary Encoding giúp tiết kiệm bộ nhớ và có thể hoạt động hiệu quả hơn trên tập dữ liệu lớn.

- Giảm thiểu tính thưa thớt: Một trong những nhược điểm của One-Hot Encoding là nó tạo ra ma trận thưa thớt (sparse matrix), nơi đa số các giá trị là 0. Binary Encoding có thể giảm bớt sự thưa thớt này.

- Nhược điểm của Binary Encoding:
  - Phức tạp hơn One-Hot Encoding: Mặc dù Binary Encoding ít cột hơn, nhưng nó lại phức tạp hơn One-Hot Encoding và có thể khó hiểu hơn trong việc giải thích mô hình.
  - Khả năng gây mất thông tin: Các giá trị nhị phân có thể không giữ được mối quan hệ giữa các giá trị phân loại, đặc biệt là khi có sự khác biệt lớn trong ý nghĩa của các giá trị. Điều này có thể gây mất mát thông tin trong một số trường hợp.

2.2.8. Overfitting

Overfitting (quá khớp) là một hành vi học máy không mong muốn xảy ra khi mô hình máy học đưa ra dự đoán chính xác cho dữ liệu đào tạo nhưng không cho dữ liệu mới.

Hiện tượng quá khớp xảy ra do một số nguyên nhân, chẳng hạn như:

- Kích thước dữ liệu đào tạo quá nhỏ và không chứa đủ mẫu dữ liệu để thể hiện chính xác tất cả các giá trị dữ liệu đầu vào khả thi.
- Dữ liệu đào tạo chứa một lượng lớn thông tin không liên quan, được gọi là dữ liệu nhiễu.
- Mô hình đào tạo quá lâu trên một tập dữ liệu mẫu duy nhất.
- Do có độ phức tạp cao, mô hình học cả phần nhiễu trong dữ liệu đào tạo.

Giải pháp dùng để khắc phục overfitting :

L2 regularization và cross-validation là hai kỹ thuật quan trọng trong machine learning, giúp mô hình tổng quát hóa tốt hơn và giảm nguy cơ overfitting.

2.2.8.2. L2 Regularization (Ridge Regularization):

· L2 regularization thêm một hình phạt vào hàm mất mát, nhằm giảm độ lớn của các tham số (weights) trong mô hình.

· Hàm mất mát khi sử dụng L2 regularization có dạng:

$$J(w) = \frac{1}{n}\sum_{i=1}^{n}\text{Loss}(y_i, \hat{y}_i) + \lambda\sum_{j=1}^{p}w_j^2$$

- λ: Hệ số điều chỉnh mức độ regularization. Giá trị lớn hơn dẫn đến hình phạt lớn hơn, giúp giảm độ phức tạp của mô hình.
- w_j: Các tham số của mô hình.

Mục tiêu: Tránh overfitting bằng cách giữ cho các trọng số www không quá lớn, đồng thời vẫn giữ được khả năng dự đoán.

2.2.8.2. Cross-Validation (CV)

- Cross-validation là phương pháp đánh giá hiệu suất mô hình trên tập dữ liệu không được sử dụng để huấn luyện.
- Kỹ thuật phổ biến: k-fold cross-validation:
    - Dữ liệu được chia thành kkk phần.
    - Huấn luyện trên k−1k-1k−1 phần và kiểm tra trên phần còn lại, lặp lại kkk lần.
    - Tính trung bình lỗi để có đánh giá ổn định.
- Quy trình kết hợp:
    1. Chia tập dữ liệu thành tập huấn luyện và tập kiểm tra (hoặc sử dụng cross-validation toàn phần trên toàn bộ dữ liệu).
    2. Với mỗi giá trị λ:
        - Huấn luyện mô hình với L2 regularization trên tập huấn luyện.
        - Sử dụng cross-validation để đo hiệu suất.
    3. Chọn λ tối ưu từ kết quả cross-validation.
    4. Huấn luyện lại mô hình với λ tối ưu trên toàn bộ tập huấn luyện.
    5. Đánh giá trên tập kiểm tra.

2.3. Các kỹ thuật đã sử dụng

- Kỹ thuật xử lý dữ liệu (Data Preprocessing)
- Kỹ thuật chia dữ liệu (Data Splitting)
- Kỹ thuật xây dựng mô hình (Model Building)
- Kỹ thuật đánh giá mô hình (Model Evaluation)
- Kỹ thuật trực quan hóa (Visualization)
- Kỹ thuật xử lý số học
- Kỹ thuật lưu trữ kết quả

3. CHƯƠNG 3: KẾT QUẢ VÀ ĐÁNH GIÁ MÔ HÌNH

3.1. Naïve Bayes

3.1.1. Thu thập dữ liệu

- Thu thập dữ liệu:

```
df = pd.read_csv("DATA.csv")
```

- Dữ liệu được đọc từ file DATA.csv

Xử lý dữ liệu thiếu và chuyển đổi dữ liệu

```
mode_value = df['Financial_Stress'].mode()[0]
df['Financial_Stress'] = df['Financial_Stress'].replace('?', mode_value)
def preprocess_data(df):
    # Xử lý giá trị thiếu cho tất cả các cột
    for column in df.columns:
        if df[column].dtype == 'object':
            mode_value = df[column].mode()[0]
            df[column] = df[column].replace('?', mode_value)

    # Tiếp tục xử lý như cũ
    for column in df.columns:
        if df[column].dtype == 'object':
            unique_values = df[column].unique()
            mapping = {value: idx for idx, value in enumerate(unique_values)}
            df[column] = df[column].map(mapping)
        elif df[column].dtype != 'object':
            df[column] = df[column].astype(float)
```

- Thay thế giá trị ”?” bằng mode (giá trị xuất hiện nhiều nhất) của cột ương ứng
- Chuyển đổi dữ liệu categorical sang numerical bằng label encoding
- Chuyển đổi các cột số sang kiểu float

- Chuẩn bị dữ liệu cho mô hình

```
X = df.drop('Depression', axis='columns').values
y = df['Depression'].values
```

- Tách features và target

3.1.2. Chia tập Train/Test

```
def train_test_split_custom(X, y, test_size=0.2, random_state=None):
    np.random.seed(random_state)
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    split = int(len(X) * (1 - test_size))
    train_indices, test_indices = indices[:split], indices[split:]
    return X[train_indices], X[test_indices], y[train_indices], y[test_indices]
```

- Tỷ lệ chia 80% Train và 20% Test
- Sử dụng random_state để đảm bảo tính tái lập

3.1.3. Xây dựng mô hình

- Huấn luyện mô hình

```
class NaiveBayes:

    def __init__(self):
        self.class_probabilities = {}
        self.feature_statistics = {}

    def fit(self, X, y):
        # Số lượng mẫu và số lượng đặc trưng
        n_samples, n_features = X.shape

        # Các lớp trong dữ liệu
        classes = np.unique(y)

        # Tính xác suất a priori cho mỗi lớp
        for c in classes:
            self.class_probabilities[c] = np.sum(y == c) / n_samples

        # Tính toán trung bình và phương sai cho mỗi lớp và đặc trưng
        for c in classes:
            X_c = X[y == c]
            feature_stats = {}
            for feature_idx in range(n_features):
                feature_values = X_c[:, feature_idx]
                mean = np.mean(feature_values)
                var = np.var(feature_values)
                feature_stats[feature_idx] = {'mean': mean, 'var': var}
            self.feature_statistics[c] = feature_stats
```

- Sử dụng công thức Gaussian Naive Bayes

```
    def gaussian_probability(self, x, mean, var):
        # Thêm một giá trị nhỏ để tránh chia cho 0
        epsilon = 1e-10
        var = var + epsilon
        exponent = math.exp(-0.5 * ((x - mean) ** 2) / var)
        return (1 / math.sqrt(2 * math.pi * var)) * exponent
```

- Dự đoán

```
    def predict(self, X):
        predictions = []
        for x in X:
            class_scores = {}
            for c, class_prob in self.class_probabilities.items():
                # Tính xác suất log của lớp
                log_prob = math.log(class_prob)
                # Tính toán xác suất điều kiện cho từng thuộc tính
                for feature_idx, value in enumerate(x):
                    mean = self.feature_statistics[c][feature_idx]['mean']
                    var = self.feature_statistics[c][feature_idx]['var']
                    # Tính xác suất của giá trị với phân phối chuẩn
                    prob = self.gaussian_probability(value, mean, var)
                    log_prob += math.log(prob)
                class_scores[c] = log_prob
            # Chọn lớp có xác suất cao nhất
            predictions.append(max(class_scores, key=class_scores.get))
        return np.array(predictions)
```

3.1.4. Đánh giá mô hình

- Metrics đánh giá và trực quan hóa kết quả

```
accuracy = sum([1 if pred == true else 0 for pred, true in zip(y_pred, y_test)]) / len(y_test)
print(f"Accuracy: {accuracy * 100:.2f}%")
plt.figure(figsize=(8, 6))
sleep_duration=X_test[:,5]
Work_Study=X_test[:,8]
sns.scatterplot(x=sleep_duration,y=Work_Study,hue=y_pred, palette='viridis', s=100, edgecolor='k')
plt.title("Sleep_Duration vs Work/Study Hours with Predicted Classes")
plt.xlabel("Sleep_Duration")
plt.ylabel("Work/Study Hours")
plt.legend(title="Predicted Class")

# Hiển thị biểu đồ
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

3.1.5. Lưu kết quả

```
results_df = pd.DataFrame({
    'True Labels': y_test,
    'Predicted Labels': y_pred
})
results_df.to_csv('predictions.csv', index=False)
```

- Lưu kết quả dự đoán vào file CSV để phân tích sau này

3.2. Logistic Regression - BagOfWords

3.2.1. Thu thập dữ liệu

- Đọc Data:

```
## read data
df = pd.read_csv('DATA_TRAIN_LOGISTIC_BagOfWords.csv')
pd.DataFrame(df)
```

- Xử lý dữ liệu xử dụng phương pháp Bag Of Words:

```
non_numeric_columns = list(df.select_dtypes(exclude=["number"]).columns)
print(non_numeric_columns)


result_df = BagOfWord(df, non_numeric_columns)
```

```
##bagOfWord
def BagOfWord(result_df,String_Atoutlooktributes):
    vectorizer = CountVectorizer(tokenizer=lambda x: [x],token_pattern=None, lowercase=False)
    for i in String_Atoutlooktributes:
        BoW = vectorizer.fit_transform(result_df[i])
        BoW_array = BoW.toarray()
        BoW_df = pd.DataFrame(BoW_array, columns=vectorizer.get_feature_names_out())
        result_df = pd.concat([result_df, BoW_df], axis=1)
        result_df = result_df.drop(columns=[i])
    return result_df
```

3.2.2. Chia tập Train/Test

- X chứa dữ liệu tập train, y chứa nhãn tập train:

```
X = result_df.iloc[ : , :-1].to_numpy()
y = df.iloc[ : , -1].to_numpy()
```

3.2.3. Xây dựng mô hình

-- Chèn thêm một np.ones((X.shape[0],1)) lên hàng đầu để phục vụ bias:

```
X = np.hstack((X, np.ones((X.shape[0], 1))))
X = X.astype(float)
```

- Khởi tạo các giá trị eta (learning rate), d (chứa số lượng thuộc tính), w (weights khởi tạo ban đầu random):

```
eta = .05
d = X.shape[1]
w_init = np.random.randn(1, d)
w = logistic_sigmoid_regression(X, y, w_init, eta)
```

- Function thực hiện logistic Regression:

```
def sigmoid(w,x):
    return 1/(1 + np.exp(-np.dot(w.T,x)))

##
def logistic_sigmoid_regression(X, y, w_init, eta, tol = 1e-4, max_count = 10000):
    w = [w_init]
    it = 0
    count = 0
    d = X.shape[1]
    N = X.shape[0]
    check_after = 20
    while count < max_count:
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X[i]
            yi = y[i]
            zi = sigmoid(w[-1].T,xi)
            w_new = w[-1] + eta*(yi-zi)*xi
            count+=1
            #stopping
            if count % check_after == 0:
                if np.linalg.norm(w_new - w[-check_after]) < tol:
                    return w
            w.append(w_new)
    return w
```

3.2.4. Đánh giá mô hình

-- So sánh độ chính xác giữa tập train và tập test:

```
##test voiw train

custom_round_vectorized = np.vectorize(custom_round)
y_pred_train = custom_round_vectorized(sigmoid(w[-1].T,X.T))
print(y_pred_train)

percentage = calculate_match_percentage(y_pred_train[0], y)
print("percentage match with train = ", percentage, "%")

##TEST VOI TAP TEST
dfTest = pd.read_csv("DATA_TEST_Logistic_BagOfWords.csv")
pd.DataFrame(dfTest)

result_dfTEST = BagOfWord(dfTest, non_numeric_columns)

XTEST = result_dfTEST.iloc[ : , :-1].to_numpy()
yTEST = result_dfTEST.iloc[ : , -1].to_numpy()

XTEST = np.hstack((XTEST, np.ones((XTEST.shape[0], 1))))
XTEST = XTEST.astype(float)
y_pred_test = custom_round_vectorized(sigmoid(w[-1].T,XTEST.T))
print(y_pred_test)

percentageTEST = calculate_match_percentage(y_pred_test[0], yTEST)
print("percentage match with test = ", percentageTEST, "%")
```

-Kết quả so sánh:

```
[[0 1 0 ... 0 1 1]]
percentage match with train =  100.0 %
[[1 0 0 ... 1 1 1]]
percentage match with test =  56.43966547192354 %
```

- Train accuraccy: 100% , Test accuraccy : 56% => trường hợp overfitting

=> Sử dụng kỹ thuật cross-validation + l2 regularization để giảm overfitting

- Sử dụng cross-validation trên tập train để tìm giá trị tối ưu của **lambda** ,sau đó sử trụng giá trị tối ưu vào hàm mất mát của l2 regularization để cập nhập w(weights) mới

```
best_lambda, best_score = cross_Validation(X,y)

w_new,bias = L2_regularization_Gradient_descent(w,XTEST,yTEST,y_pred_test,best_lambda)

w_new = [np.array(w_new)]
```

```python
def cross_Validation(X,y):
    k = 5  #
    lambdas = np.logspace(-6, 2, 9)
    fold_size = len(X) // k

    # Shuffle data
    indices = np.arange(len(X))
    np.random.shuffle(indices)
    X, y = X[indices], y[indices]

    # Initialize variables to track the best lambda
    best_lambda = None
    best_score = float('inf')  # For minimizing mean squared error

    # Perform manual cross-validation
    for lam in lambdas:
        mse_scores = []

        for fold in range(k):
            # Split the data
            start, end = fold * fold_size, (fold + 1) * fold_size
            X_val, y_val = X[start:end], y[start:end]
            X_train = np.concatenate((X[:start], X[end:]), axis=0)
            y_train = np.concatenate((y[:start], y[end:]), axis=0)

            # Train the model
            model = Ridge(alpha=lam)  # Ridge regression, alpha = lambda
            model.fit(X_train, y_train)

            # Validate the model
            y_pred = model.predict(X_val)
            mse = np.mean((y_val - y_pred) ** 2)
            mse_scores.append(mse)

        # Calculate the mean MSE across folds
        mean_mse = np.mean(mse_scores)

        # Track the best lambda
        if mean_mse < best_score:
            best_score = mean_mse
            best_lambda = lam

    print(f"Best lambda: {best_lambda}")
    print(f"Mean squared error: {best_score}")

    return best_lambda, best_score
```

```python
def L2_regularization_Gradient_descent(w,XTEST,yTEST,y_pred_test,lambda_reg ):
    n_samples, n_features = XTEST.shape
    learning_rate = 0.01
    num_iterations = 1000
    weights = list(np.zeros(len(w[-1])))
    bias = 0
    for _ in range(num_iterations):
        y_pred_train = sigmoid(weights[-1].T,XTEST.T)
        error = y_pred_test - yTEST
        grad_weights  = (1 / n_samples) * np.dot(XTEST.T, error.T) + lambda_reg * w[-1]
        grad_bias = (1 / n_samples) * np.sum(error)
        weights -= learning_rate * grad_weights
        bias -= learning_rate * grad_bias

    return weights, bias
```

- Giá trị lamda và sai số bình phương trung bình:

```
Best lambda: 1e-06
Mean squared error: 3.475768309289335e-20
```

- Độ chính xác mới sau khi sử dụng cross-validation và l2 regularization

```
##test with train after l2 regulazation

custom_round_vectorized = np.vectorize(custom_round)
y_pred_train = custom_round_vectorized(sigmoid(w_new[-1].T,X.T))

y_pred_train = np.array(y_pred_train)

percentage = calculate_match_percentage(y_pred_train[0], y)
print("percentage match with train = ", percentage, "%")

##TEST with test after l2 regulazation

y_pred_test = custom_round_vectorized(sigmoid(w_new[-1].T,XTEST.T))

percentageTEST = calculate_match_percentage(y_pred_test[0], yTEST)
print("percentage match with test = ", percentageTEST, "%")
```

```
percentage match with train =  41.42133819302647 %
percentage match with test =  62.688172043010745 %
```

=> Sau khi áp dụng Bag of Words để xử lý dữ liệu và áp dụng thuật toán Logistic Regression xảy ra trường hợp overfitting, sau khi sử dụng cross-validation và l2 regularization để giảm thiểu overfitting dẫn đến trường hợp undefitting => cách tiếp cận không phù hợp.

3.3. Logistics Regression

3.3.1. Thu thập dữ liệu

1.1 Tổng quan về dữ liệu và mã hóa dữ liệu

```
# Load the data from the uploaded CSV file
file_path = "D:/PHAN ANH/Machine Learning/cleaned_data_final.csv"
data = pd.read_csv(file_path)

# Display the first few rows to understand its structure
data.shape
```

[55]

```
(27901, 12)
```

```
data.info()     # thông tin về kiểu dữ liệu
```

[6]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27901 entries, 0 to 27900
Data columns (total 12 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   Gender                                 27901 non-null  object
 1   Age                                    27901 non-null  int64
 2   Academic_Pressure                      27901 non-null  int64
 3   CGPA                                   27901 non-null  int64
 4   Study_Satisfaction                     27901 non-null  int64
 5   Sleep_Duration                         27901 non-null  object
 6   Dietary_Habits                         27901 non-null  object
 7   Have_you_ever_had_suicidal_thoughts    27901 non-null  object
 8   Work/Study_Hours                       27901 non-null  int64
 9   Financial_Stress                       27901 non-null  object
 10  Family_History_of_Mental_Illness       27901 non-null  object
 11  Depression                             27901 non-null  int64
dtypes: int64(6), object(6)
memory usage: 2.6+ MB
```

- Lọc ra các giá trị duy nhất của các thuộc tính

```
# In ra các giá trị unique của mỗi cột
for column in data.columns:
    unique_value = data[column].unique()
    print(column + ": ", end="")
    print(', '.join(map(str, unique_value)))
```

[40]

```
Gender: Male, Female
Age: 2, 1, 3
Academic_Pressure: 5, 2, 3, 4, 1, 0
CGPA: 3, 1, 2
Study_Satisfaction: 2, 5, 3, 4, 1, 0
Sleep_Duration: 5-6 hours, Less than 5 hours, 7-8 hours, More than 8 hours, Others
Dietary_Habits: Healthy, Moderate, Unhealthy, Others
Have_you_ever_had_suicidal_thoughts: Yes, No
Work/Study_Hours: 3, 9, 4, 1, 0, 12, 2, 11, 10, 6, 8, 5, 7
Financial_Stress: 1, 2, 5, 3, 4, ?
Family_History_of_Mental_Illness: No, Yes
Depression: 1, 0
```

- Xử lý ngoại lai

1.2 Xử lý ngoại lại và và mã hóa dữ liệu

+ Code + Markdown

```
#Xử lý ngoại lai "?"
count_question_mark = (data['Financial_Stress'] == '?').sum()
print(f"Số lượng bản ghi có giá trị '?' trong cột '{'Financial_Stress'}': {count_question_mark}")
```

[41]

```
Số lượng bản ghi có giá trị '?' trong cột 'Financial_Stress': 3
```

```
data = data[data['Financial_Stress'] != '?']  # Loại bỏ các hàng có giá trị là "?"
data['Financial_Stress'] = data['Financial_Stress'].astype(int)  # Chuyển đổi sang số
print('Chuyển đổi thành công cột "Financial_Stress" về dạng int64')
print(data['Financial_Stress'].unique())
```

[42]

```
Chuyển đổi thành công cột "Financial_Stress" về dạng int64
[1 2 5 3 4]
```

- Encode binary

```
# Danh sách các cột cần encode binary
columns_to_encode = ['Gender', 'Sleep_Duration', 'Dietary_Habits',
                     'Have_you_ever_had_suicidal_thoughts',
                     'Family_History_of_Mental_Illness']

# Apply Binary Encoding to the specified columns
encoder = BinaryEncoder(cols=columns_to_encode)
data = encoder.fit_transform(data)

# Display the first few rows of the encoded data
data.head()
```

[43] Python

```
C:\Users\phana\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\base.py:411: FutureWarning: The `_get_tags` method is deprecated in 1.6 and will be removed in 1.7. Plea
  warnings.warn(
```

	Gender_0	Gender_1	Age	Academic_Pressure	CGPA	Study_Satisfaction	Sleep_Duration_0	Sleep_Duration_1	Sleep_Duration_2	Dietary_Habits_0	Dietary_Habits_1	Dietary_Habits_2	Have_you_eve
0	0	1	2	5	3	2	0	0	1	0	0	1	
1	1	0	1	2	1	5	0	0	1	0	1	0	
2	0	1	2	3	2	5	0	1	0	0	0	1	
3	1	0	2	3	1	2	0	1	1	0	1	0	
4	1	0	1	4	2	3	0	0	1	0	1	0	

3.3.2. Chia tập Train/Test

```
# Tách dữ liệu thành features (X) và target (y)
X = data.drop(columns=['Depression'])  # Thay 'target' bằng tên cột nhãn
y = data['Depression']                 # Thay 'target' bằng tên cột nhãn

# Chia dữ liệu train-test với 80% train và 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Kiểm tra kích thước tập train và test
print("Train size:", X_train.shape[0])
print("Test size:", X_test.shape[0])
```

[44]

```
Train size: 22318
Test size: 5580
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

print(X_train[:10])  # In ra 10 hàng đầu tiên
```

[45]

- Chia Train/Test

3.3.3. Xây dựng mô hình

```
#Code Logistic Regression
import numpy as np

# Hàm sigmoid
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Hàm khởi tạo tham số
def initialize_weights(n_features):
    weights = np.zeros(n_features)
    return weights

# Huấn luyện Logistic Regression
def train_logistic_regression(X_train, y_train, learning_rate=0.01, num_iterations=1000):
    tol = 1e-4

    #Khởi tạo mảng weights
    nb_features = X_train.shape[1]
    weights = initialize_weights(nb_features)
    w = [weights]

    it = 0
    count = 0
    d = X_train.shape[1]
    N = X_train.shape[0]
    check_after = 20

    while count < num_iterations:
        mix_id = np.random.permutation(N)
        for i in mix_id:
            xi = X_train[i]
            zi = sigmoid(np.dot(w[-1].T,xi))

            yi = y_train[i]

            w_new = w[-1] + learning_rate*(yi-zi)*xi
            count+=1
```

```
            #stopping
            if count % check_after == 0:
                if np.linalg.norm(w_new - w[-check_after]) < tol:
                    return w
            w.append(w_new)
    return w

# Hàm dự đoán dùng train
def predict(X, weights):
    y_pred = sigmoid(np.dot(X, weights))
    return (y_pred >= 0.5).astype(int)

# Hàm dự đoán cho người dùng
def predict_with_probabilities(X, weights):
    # Tính xác suất thuộc lớp 1
    prob_class_1 = sigmoid(np.dot(X, weights))
    prob_class_0 = 1 - prob_class_1  # Xác suất lớp 0

    # Dự đoán class (theo ngưỡng 0.5)
    predicted_class = (prob_class_1 >= 0.5).astype(int)

    # Trả về xác suất và class dự đoán
    return prob_class_0, prob_class_1, predicted_class
```

3.3.4. Đánh giá mô hình

```
# Đánh giá mô hình
def evaluate(y_true, y_pred):
    accuracy = np.mean(y_true == y_pred)
    print(f"Accuracy: {accuracy * 100:.2f}%")
    return accuracy

# Huấn luyện và kiểm tra mô hình
if __name__ == "__main__":
    # Chuyển đổi dữ liệu sang numpy array
    X_train = np.array(X_train)
    X_test = np.array(X_test)
    y_train = np.array(y_train)
    y_test = np.array(y_test)

    # Huấn luyện mô hình
    learning_rate = 0.01
    num_iterations = 1000
    weights = train_logistic_regression(X_train, y_train, learning_rate, num_iterations)
    weights = weights[-1]

    # Dự đoán trên tập train
    print("\nPerformance on Training Set:")
    y_train_pred = predict(X_train, weights)
    evaluate(y_train, y_train_pred)

    # Dự đoán trên tập test
    print("\nPerformance on Test Set:")
    y_test_pred = predict(X_test, weights)
    evaluate(y_test, y_test_pred)
```

[21]

```
Performance on Training Set:
Accuracy: 83.49%

Performance on Test Set:
Accuracy: 83.53%
```

TÀI LIỆU THAM KHẢO

[1]. Cox, David R. “The regression analysis of binary sequences.” Journal of the Royal Statistical Society. Series B (Methodological) (1958): 215-242.

[2] Cramer, Jan Salomon. “The origins of logistic regression.” (2002).

[3] Tiệp, V. H. (n.d.). Bài 32: Naive Bayes Classifier.

[4] Exercise 6: Naive Bayes - Machine Learning - Andrew Ng

[5] Text Classification and Naive Bayes - Stanford