ĐẠI HỌC QUỐC GIA HÀ NỘI

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

BÁO CÁO

Chủ đề: Earthquake Damage Prediction

Sinh viên thực hiện:
Đoàn Hữu Hoan
Dương Quang Khải
Tô Quốc Thái Dương

Giảng viên hướng dẫn:
TS. Cao Văn Chung

Ngày 27 tháng 5 năm 2024

Mục lục

1 Giới thiệu

1.1 Tổng quan

Việc đánh giá mức độ thiệt hại của các công trình sau động đất thông qua tính toán luôn phức tạp, tốn nhiều thời gian và không phù hợp với những trường hợp có số lượng lớn các tòa nhà cần được đánh giá sau trận động đất. Một trong những bộ dữ liệu lớn nhất sau thảm họa, có được từ trận động đất ở Nepal vào tháng 4 năm 2015, giúp đào tạo mô hình để dự đoán mức độ thiệt hại dựa trên thông tin tòa nhà. Ở đây, chúng ta khảo sát các ứng dụng các phương pháp học máy khác nhau vào vấn đề mức độ hư hỏng cấu trúc.

1.2 Giới thiệu

Sự chuyển động của mặt đất trong trận động đất sẽ gây ra thiệt hại rất lớn cho các công trình xây dựng. Có nhiều cấp độ cường độ động đất khác nhau, nhưng không có sự phân loại có hệ thống về thiệt hại của các tòa nhà. Việc phân tích động học của một kết cấu đơn lẻ thường mất nhiều thời gian, điều này không phù hợp với những tình huống cần đánh giá một số lượng lớn tòa nhà sau một trận động đất. Sau trận động đất ở Nepal vào tháng 4 năm 2015, là thảm họa thiên nhiên tồi tệ nhất tấn công Nepal kể từ năm 1934, Ủy ban Kế hoạch Quốc gia, cùng với Phòng thí nghiệm sự sống ở Kathmandu và Cục Thống kê Trung ương, đã tạo ra các bộ dữ liệu lớn sau thảm họa, chứa thông tin có giá trị về tác động của động đất, điều kiện hộ gia đình và thống kê kinh tế xã hội - nhân khẩu học. Chúng tôi tin rằng thông qua việc áp dụng máy học trên bộ dữ liệu này, chúng ta có thể nhanh chóng phân loại mức độ thiệt hại của các công trình để thuận tiện cho việc lập kế hoạch cứu hộ, tái thiết và đánh giá tổn thất.

1.3 Bài toán

- Dự đoán biến thứ tự Damage grade, đại diện cho mức độ thiệt hại của tòa nhà bị ảnh hưởng bởi trận động đất. Có 3 mức độ thiệt hại:

  1 tượng trưng cho mức độ phá hủy thấp.

  2 tượng trưng cho mức độ phá hủy trung bình.

  3 tượng trưng cho sự phá hủy gần như hoàn toàn.

- Sử dụng tập dữ liệu 260,601 bản ghi và 39 trường dữ liệu chứa thông tin các tòa nhà.

- Áp dụng các kỹ thuật tiền xử lý dữ liệu, giảm chiều dữ liệu. Dự đoán và phân loại sử dụng các phương pháp hồi quy tuyến tính, Naïve Bayes; Multinomial Logistic.

- Sử dụng độ đo (accuracy, precision, recall) và thời gian chạy để đánh giá và so sánh giữa các mô hình.

1.4 Mục tiêu

- Áp dụng các kỹ thuật tiền xử lý và giảm chiều dữ liệu hiệu quả.

- Xây dựng mô hình dự đoán và phân loại chính xác.

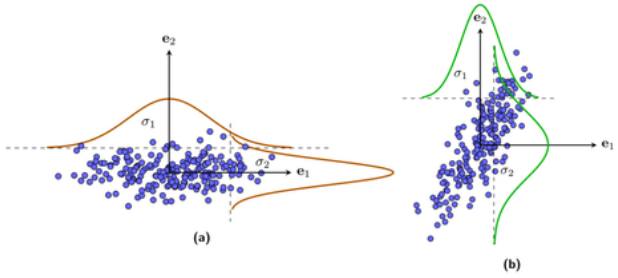
- Tìm hiểu và phân tích lý thuyết tường minh, cụ thể.

2 Principal Component Analysis (PCA)

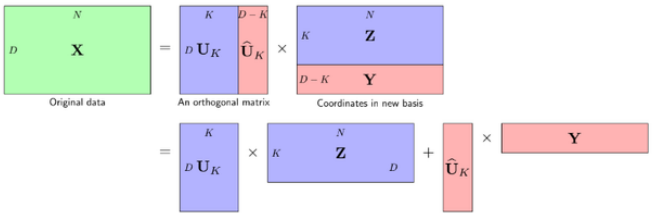
Trong machine learning, Principal Component Analysis (PCA) là một kỹ thuật giảm chiều dữ liệu phổ biến được sử dụng để giảm số lượng biến đầu vào trong dữ liệu mà vẫn giữ lại phần lớn thông tin quan trọng. Đây là một phần quan trọng của quá trình tiền xử lý dữ liệu và thường được áp dụng trước khi áp dụng các mô hình máy học như phân loại, hồi quy, v.v.

2.1 Ý tưởng

Giả sử dữ liệu ban đầu là $\mathrm{x} \in \mathbb{R}^D$ và dữ liệu đã được giảm chiều là $\mathbf{z} \in \mathbb{R}^K$ với $K < D$. Cách đơn giản nhất để giảm chiều dũ liệu từ D về $K < D$ là chỉ giữ lại K phần tử quan trọng nhất. Có hai câu hỏi lập tức được đặt ra ở đây. Thứ nhất, làm thế nào để xác định tầm quan trọng của mỗi chiều dữ liệu? Thứ hai, nếu tầm quan trọng của các chiều dữ liệu là như nhau, ta cần bỏ đi những chiều nào?
Để trả lời câu hỏi thứ nhất, ta hãy quan sát Hình 1. Giả sử các điểm dữ liệu có thành phần thứ hai (phương đứng) giống hệt nhau hoặc sai khác nhau rất ít (phương sai nhỏ). Như vậy, thành phần này hoàn toàn có thể được lược bỏ đi, và ta ngầm hiểu rằng nó sẽ được xấp xỉ bằng kỳ vọng của thành phần đó trên toàn bộ các điểm dữ liệu. Ngược lại, việc làm này nếu được áp dụng lên thành phần thứ nhất (phương ngang) sẽ khiến lượng thông tin bị mất đi rất nhiều do sai số xấp xỉ là quá lớn. Lượng thông tin theo mỗi thành phần, vì vậy, có thể được đo bằng phương sai của dữ liệu trên thành phần đó. Tổng lượng thông tin có thể được coi là tổng phương sai trên toàn bộ các thành phần. Lấy một ví dụ về việc có hai camera được đặt dùng để chụp một con người, một camera đặt phía trước người và một camera đặt trên đầu. Rõ ràng, hình ảnh thu được từ camera đặt phía trước người mang nhiều thông tin hơn so với hình ảnh nhìn từ phía trên đầu. Vì vậy, bức ảnh chụp từ phía trên đầu có thể được bỏ qua mà không có quá nhiều thông tin về hình dáng của người đó bị mất.
Câu hỏi thứ hai tương ứng với trường hợp Hình 2. Trong cả hai chiều, phương sai của dữ liệu đều lớn; việc bỏ đi một trong hai chiều đều dẫn đến việc lượng thông tin bị mất đi là lớn. Tuy nhiên, quan sát ban đầu của chúng ta là nếu xoay trục toạ độ đi một góc phù hợp, một trong hai chiều dữ liệu có thể được giảm đi vì dữ liệu có xu hướng phân bố xung quanh một đường thẳng.



Hình 1: Ví dụ về phương sai của dữ liệu trong không gian hai chiều. (a) Chiều thứ hai có phương sai (tỉ lệ với độ rộng của đường hình chuông) nhỏ hơn chiều thứ nhất. (b) Cả hai chiều có phương sai đáng kể. Phương sai của mỗi chiều là phương sai của thành phần tương ứng được lấy trên toàn bộ dữ liệu. Phương sai tỉ lệ thuận với độ phân tán của dữ liệu.



Hình 2: Ý tưởng chính của PCA: Tìm một hệ trực chuẩn mới sao cho trong hệ này, các thành phần quan trọng nhất nằm trong K thành phần đầu tiên.

Tóm lại, Principle component analysis (PCA) là một phương pháp đi tìm một phép xoay trục toạ độ để được một hệ trục toạ độ mới sao cho trong hệ mới này, thông tin của dữ liệu chủ yếu tập trung ở một vài thành phần. Phần còn lại chưa ít thông tin hơn có thể được lược bỏ.

2.2 Nguyên lý hoạt động

Nguyên lý hoạt động của PCA (Principal Component Analysis) trong machine learning là giảm số chiều của dữ liệu bằng cách tìm ra các thành phần chính (principal components) của dữ liệu, giữ lại thông tin quan trọng nhất và loại bỏ thông tin không quan trọng. Quá trình này giúp làm giảm độ phức tạp của dữ liệu và cải thiện hiệu suất của các mô hình máy học.

- **Tính toán ma trận hiệp phương sai (Covariance Matrix)**: Đầu tiên, PCA tính toán ma trận hiệp phương sai của dữ liệu. Ma trận này cho biết mức độ tương quan giữa các biến trong dữ liệu. Bằng cách này, PCA xác định các hướng chính của biến thiên trong không gian dữ liệu.
- **Tìm các vectơ riêng và giá trị riêng (Eigenvectors and Eigenvalues)**: Sau khi tính toán ma trận hiệp phương sai, PCA tiếp tục tìm ra các vectơ riêng và giá trị riêng của ma trận này. Các vectơ riêng biểu diễn các hướng chính của biến thiên trong dữ liệu, và giá trị riêng tương ứng cho biết mức độ biến thiên theo hướng đó.
- **Chọn các thành phần chính (Principal Components)**: PCA sắp xếp các giá trị riêng theo thứ tự giảm dần. Các thành phần chính tương ứng với các giá trị riêng lớn nhất, tức là các hướng chính của biến thiên trong dữ liệu. Bằng cách này, PCA giữ lại các thành phần quan trọng nhất của dữ liệu.
- **Biến đổi dữ liệu (Data Transformation)**: Cuối cùng, PCA biến đổi dữ liệu gốc thành dữ liệu mới, mỗi điểm dữ liệu mới được biểu diễn dưới dạng các giá trị của các thành phần chính đã chọn. Dữ liệu mới có số chiều ít hơn so với dữ liệu gốc, nhưng vẫn giữ lại một phần lớn thông tin quan trọng của dữ liệu gốc.

2.3 Các bước thực hiện PCA

Từ các suy luận phía trên, ta có thể tóm tắt lại các bước trong PCA như sau:

- Tính vector kỳ vọng của toàn bộ dữ liệu: $\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$.

- Trừ mỗi điểm dữ liệu đi vector kỳ vọng của toàn bộ dữ liệu để được dữ liệu chuẩn hoá:

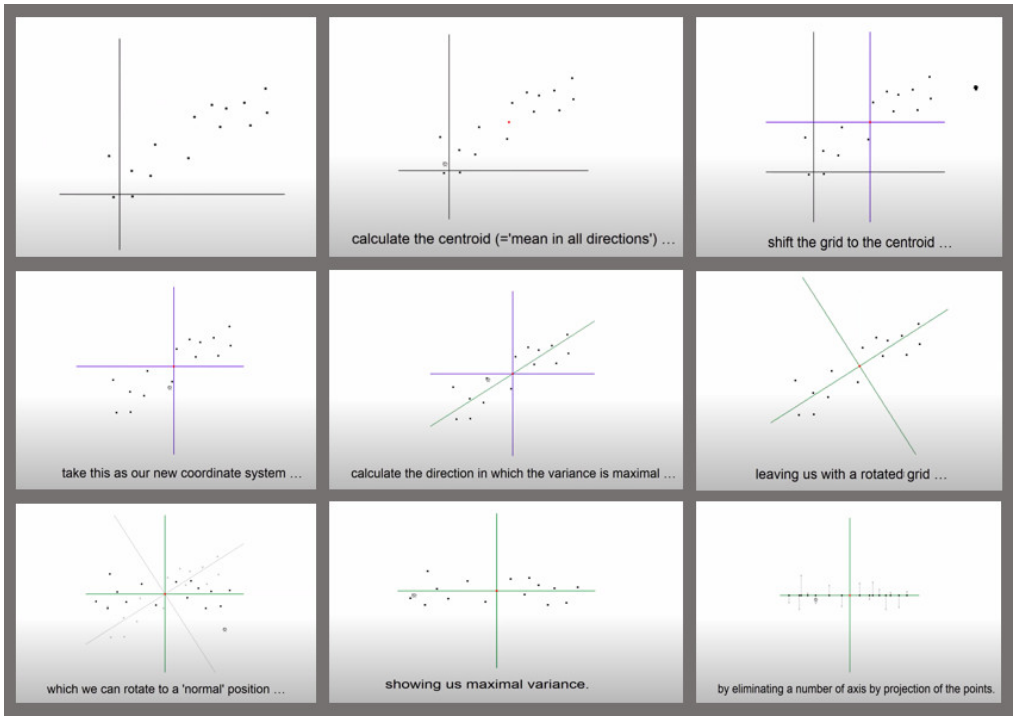
$$\hat{\mathbf{x}}_n = \mathbf{x}_n - \overline{\mathbf{x}}$$

- Đặt $\widehat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_D]$ là ma trận dữ liệu chuẩn hoá, tính ma trận hiệp phương sai

$$\mathbf{S} = \frac{1}{N}\widehat{\mathbf{X}}\widehat{\mathbf{X}}^T$$

- Tính các trị riêng và vector riêng tương ứng có ℓ_2 norm bằng 1 của ma trận này, sắp xếp chúng theo thứ tự giảm dần của trị riêng.
- Chọn K vector riêng ứng với K trị riêng lớn nhất để xây dựng ma trận $\mathbf{U}_K$ có các cột tạo thành một hệ trực giao. K vectors này, còn được gọi là các thành phần chính, tạo thành một không gian con gần với phân bố của dữ liệu ban đầu đã chuẩn hoá.
- Chiếu dữ liệu ban đầu đã chuẩn hoá $\widehat{\mathbf{X}}$ xuống không gian con tìm được.
- Dữ liệu mới chính là toạ độ của các điểm dữ liệu trên không gian mới: $\mathbf{Z} = \mathbf{U}_K^T\widehat{\mathbf{X}}$.

  Dữ liệu ban đầu có thể tính được xấp xỉ theo dữ liệu mới bởi $\mathbf{x} \approx \mathbf{U}_K\mathbf{Z} + \overline{\mathbf{x}}$. Một điểm dữ liệu mới $\mathbf{v} \in \mathbb{R}^D$ (có thể không nằm trong tập huấn luyện) sẽ được giảm chiều bằng PCA theo công thức $\mathbf{w} = \mathbf{U}_K^T(\mathbf{v} - \overline{\mathbf{x}}) \in \mathbb{R}^K$. Ngược lại, nếu biết $\mathbf{w}$, ta có thể xấp xỉ $\mathbf{v}$ bởi $\mathbf{U}_K\mathbf{w} + \overline{\mathbf{x}}$. Các bước thực hiện PCA được minh hoạ trong Hình 3.



Hình 3: Các bước thực hiện PCA.

3 Hồi quy tuyến tính

Linear Regression là một trong những thuật toán đơn giản nhất và phổ biến nhất trong machine learning, được sử dụng để dự đoán giá trị đầu ra dựa trên các biến đầu vào. Thuật toán này giải quyết bài toán dự đoán một biến liên tục dựa trên các biến đầu vào, trong đó mối quan hệ giữa biến đầu vào và biến đầu ra được mô tả bằng một hàm tuyến tính.

3.1 Các khái niệm

- **Hàm mục tiêu (Objective function)**: Trong Linear Regression, hàm mục tiêu thường là hàm bình phương của sai số giữa giá trị dự đoán và giá trị thực tế. Mục tiêu là tối thiểu hóa hàm mục tiêu để mô hình có thể dự đoán chính xác nhất.

- **Tham số (Parameters)**: Trong mô hình tuyến tính, các tham số là các trọng số (hoặc hệ số) được gán cho mỗi biến đầu vào. Các tham số này cần được điều chỉnh sao cho mô hình dự đoán gần nhất với dữ liệu thực tế.

- **Tối thiểu hóa bình phương tối thiểu (Least Squares Minimization)**: Phương pháp này được sử dụng để tìm ra các giá trị tối thiểu của hàm mục tiêu (thường là tổng bình phương sai số) bằng cách điều chỉnh các tham số của mô hình.

- **Hàm mất mát (Loss function)**: Trong Linear Regression, hàm mất mát thường là tổng bình phương sai số giữa giá trị dự đoán và giá trị thực tế. Công thức toán học cho hàm mất mát thường là:

  $L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

  n là số lượng mâuu trong tập huấn luyện.

  y_i là giá trị thực tế của mâu thứ i.

  $\hat{y}_i$ là giá trị dự đoán của mâu thứ i.

- **Gradient Descent**: Để tối thiểu hóa hàm mất mát, một trong những phương pháp phổ biến là Gradient Descent. Thuật toán này cập nhật các tham số của mô hình theo hướng âm của đạo hàm của hàm mất mát để tiến gần đến điểm tối ưu.

  Đánh giá hiệu suất (Performance Evaluation): Các phương pháp đánh giá hiệu suất của mô hình Linear Regression thường bao gồm:

  Mean Squared Error (MSE): Đo lường trung bình của bình phương sai số giữa giá trị dự đoán và giá trị thực tế.

  R-squared (Coefficient of Determination): Đo lường tỷ lệ biến thiên của dữ liệu mà mô hình có thể giải thích.

- **Regularization**: Trong một số trường hợp, regularization có thể được áp dụng để giảm overfitting trong Linear Regression. Các kỹ thuật regularization bao gồm L1 regularization (Lasso) và L2 regularization (Ridge).

3.2 Nguyên lý hoạt động

Nguyên lý hoạt động của Linear Regression trong machine learning là tìm ra một mô hình tuyến tính tốt nhất để mô tả mối quan hệ giữa biến đầu vào và biến mục tiêu. Cụ thể, mô hình tuyến tính này được xây dựng dựa trên các điểm dữ liệu huấn luyện để dự đoán giá trị của biến mục tiêu dựa trên giá trị của các biến đầu vào.

- **Mô hình tuyến tính**: Linear Regression giả định rằng mối quan hệ giữa biến đầu vào và biến mục tiêu là tuyến tính. Điều này có nghĩa là có một mối liên hệ tuyến tính giữa biến đầu vào và biến mục tiêu, có thể được biểu diễn bằng một đường thẳng (trong trường hợp một biến đầu vào) hoặc một siêu phẳng (trong trường hợp nhiều biến đầu vào).

- **Hàm mục tiêu (Objective function)**: Mục tiêu của Linear Regression là tìm ra một mô hình tuyến tính sao cho sai số giữa giá trị dự đoán và giá trị thực tế là nhỏ nhất. Thông thường, Linear Regression sử dụng phương pháp bình phương tối thiểu để tối thiểu hóa tổng bình phương sai số giữa giá trị dự đoán và giá trị thực tế trên tập dữ liệu huấn luyện.

- **Tối thiểu hóa hàm mất mát**: Linear Regression sử dụng các phương pháp như phương pháp Gradient Descent hoặc các phương pháp toán học khác để điều chỉnh các tham số của mô hình, tức là các hệ số (weights) của các biến đầu vào, để tối thiểu hóa hàm mất mát.

- **Ứng dụng**: Linear Regression có thể được áp dụng trong nhiều lĩnh vực, bao gồm dự đoán giá cổ phiếu, dự đoán giá nhà, dự đoán doanh số bán hàng, và nhiều ứng dụng khác trong thực tế.

3.3 Phân tích toán học

3.3.1 Dạng của Linear Regression

Tìm hiểu ví dụ: một căn nhà rộng x_1 m^2, có x_2 phòng ngủ và cách trung tâm thành phố x_3 km có giá là bao nhiêu. Giả sử chúng ta đã có số liệu thống kê từ 1000 căn nhà trong thành phố đó, liệu rằng khi có một căn nhà mới với các thông số về diện tích, số phòng ngủ và khoảng cách tới trung tâm, chúng ta có thể dự đoán được giá của căn nhà đó không? Nếu có thì hàm dự đoán $y = f(\mathbf{x})$ sẽ có dạng như thế nào. Ở đây $\mathbf{x} = [x_1, x_2, x_3]$ là một vector hàng chứa thông tin input, y là một số vô hướng (scalar) biểu diễn output (tức giá của căn nhà trong ví dụ này).
Lưu ý về ký hiệu toán học: trong các bài viết của tôi, các số vô hướng được biểu diễn bởi các chữ cái viết ở dạng không in đậm, có thể viết hoa, ví dụ x_1, N, y, k. Các vector được biểu diễn bằng các chữ cái thường in đậm, ví dụ $\mathbf{y}, \mathbf{x}_1$. Các ma trận được biểu diễn bởi các chữ viết hoa in đậm, ví dụ $\mathbf{X}, \mathbf{Y}, \mathbf{W}$.
Một cách đơn giản nhất, chúng ta có thể thấy rằng: i) diện tích nhà càng lớn thì giá nhà càng cao; ii) số lượng phòng ngủ càng lớn thì giá nhà càng cao; iii) càng xa trung tâm thì giá nhà càng giảm. Một hàm số đơn giản nhất có thể mô tả mối quan hệ giữa giá nhà và 3 đại lượng đầu vào là:

$$y \approx f(\mathbf{x}) = \hat{y}$$
$$f(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0$$

trong đó, w_1, w_2, w_3, w_0 là các hằng số, w_0 còn được gọi là bias. Mối quan hệ $y \approx f(\mathbf{x})$ bên trên là một mối quan hệ tuyến tính (linear). Bài toán chúng ta đang làm là một bài toán thuộc loại

regression. Bài toán đi tìm các hệ số tối ưu $\{w_1, w_2, w_3, w_0\}$ chính vì vậy được gọi là bài toán Linear Regression.
Trong phương trình phía trên, nếu chúng ta đặt $\mathbf{w} = [w_0, w_1, w_2, w_3]^T =$ là vector (cột) hệ số cần phải tối ưu và $\overline{\mathbf{x}} = [1, x_1, x_2, x_3]$ (đọc là x bar trong tiếng Anh) là vector (hàng) dữ liệu đầu vào mở rộng. Số 1 ở đầu được thêm vào để phép tính đơn giản hơn và thuận tiện cho việc tính toán. Khi đó, phương trình (1) có thể được viết lại dưới dạng:

$$y \approx \overline{\mathbf{x}}\mathbf{w} = \hat{y}$$

Chú ý rằng $\overline{\mathbf{x}}$ là một vector hàng.

3.3.2 Sai số dự đoán

Chúng ta mong muốn rằng sự sai khác e giữa giá trị thực y và giá trị dự đoán $\hat{y}$ (đọc là y hat trong tiếng Anh) là nhỏ nhất. Nói cách khác, chúng ta muốn giá trị sau đây càng nhỏ càng tốt:

$$\frac{1}{2}e^2 = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(y - \overline{\mathbf{x}}\mathbf{w})^2$$

trong đó hệ số $\frac{1}{2}$ (lại) là để thuận tiện cho việc tính toán (khi tính đạo hàm thì số $\frac{1}{2}$ sẽ bị triệt tiêu). Chúng ta cần e^2 vì $e = y - \hat{y}$ có thể là một số âm, việc nói e nhỏ nhất sẽ không đúng vì khi $e = -\infty$ là rất nhỏ nhưng sự sai lệch là rất lớn. TaTa có thể tự đặt câu hỏi: tại sao không dùng trị tuyệt đối $|e|$ mà lại dùng bình phương e^2 ở đây? Câu trả lời sẽ có ở phần sau.

3.3.3 Hàm mất mát

Điều tương tự xảy ra với tất cả các cặp (input, output) $(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N$, với N là số lượng dữ liệu quan sát được. Điều chúng ta mong muốn-trung bình sai số là nhỏ nhất-tương đương với việc tìm w để hàm số sau đạt giá trị nhỏ nhất:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \mathbf{x}_i^T\mathbf{w}\right)^2$$

Hàm số $\mathcal{L}(\mathbf{w})$ chính là hàm mất mát của linear regression với tham số mô hình $\theta = \mathbf{w}$. Chúng ta luôn mong muốn rằng sự mất mát là nhỏ nhất, điều này có thể đạt được bằng cách tối thiểu hàm mất mát theo w:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\mathcal{L}(\mathbf{w})$$

w* là nghiệm cần tìm, đôi khi dấu * được bỏ đi và nghiệm có thể được viết gọn lại thành $\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmin}}\mathcal{L}(\mathbf{w})$ Trung bình sai số
Trước khi đi xây dựng nghiệm cho bài toán tối ưu hàm mất mát, ta thấy rằng hàm số này có thể được viết gọn lại dưới dạng ma trận, vector, và norm như dưới đây:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \mathbf{x}_i^T\mathbf{w}\right)^2 = \frac{1}{2N}\left\|\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}\mathbf{w}\right\|_2^2 = \frac{1}{2N}\left\|\mathbf{y} - \mathbf{X}^T\mathbf{w}\right\|_2^2$$

với $\mathbf{y} = [y_1, y_2, \ldots, y_N]^T, \mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$. Như vậy $\mathcal{L}(\mathbf{w})$ là một hàm số liên quan tới bình phương của ℓ_2 norm.

3.3.4 Nghiệm cho bài toán Linear Regression

Cách phổ biến nhất để tìm nghiệm cho một bài toán tối ưu (chúng ta đã biết từ khi học cấp 3) là giải phương trình đạo hàm (gradient) bằng 0! Tất nhiên đó là khi việc tính đạo hàm và việc giải phương trình đạo hàm bằng 0 không quá phức tạp. Thật may mắn, với các mô hình tuyến tính, hai việc này là khả thi.
Đạo hàm theo $\mathbf{w}$ của hàm mất mát là:

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = \overline{\mathbf{X}}^T(\overline{\mathbf{X}}\mathbf{w} - \mathbf{y})$$

Đến đây tôi xin quay lại câu hỏi ở phần Sai số dự đoán phía trên về việc tại sao không dùng trị tuyệt đối mà lại dùng bình phương. Câu trả lời là hàm bình phương có đạo hàm tại mọi noi, trong khi hàm trị tuyệt đối thì không (đạo hàm không xác định tại 0).
Phương trình đạo hàm bằng 0 tương đương với:

$$\overline{\mathbf{X}}^T\overline{\mathbf{X}}\mathbf{w} = \overline{\mathbf{X}}^T\mathbf{y}\mathbf{b}$$

(ký hiệu $\overline{\mathbf{X}}^T\mathbf{y}\mathbf{b}$ nghĩa là đặt $\overline{\mathbf{X}}^T\mathbf{y}$ bằng $\mathbf{b}$). Nếu ma trận vuông $\mathbf{A}\overline{\mathbf{X}}^T\overline{\mathbf{X}}$ khả nghịch (non-singular hay invertible) thì phương trình (4) có nghiệm duy nhất: $\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}$.
Với khái niệm giả nghịch đảo, điểm tối ưu của bài toán Linear Regression có dạng:

$$\mathbf{w} = \mathbf{A}^{\dagger}\mathbf{b} = \left(\overline{\mathbf{X}}^T\overline{\mathbf{X}}\right)^{\dagger}\overline{\mathbf{X}}^T\mathbf{y}$$

4 Naive Bayes

Naive Bayes là một trong những thuật toán phân loại trong Machine Learning, dựa trên việc áp dụng định lý Bayes với giả định "ngây thơ"(naive), tức là giả định rằng các đặc trưng của dữ liệu là độc lập với nhau.

4.1 Các khái niệm

- **Định lý Bayes (Bayes' Theorem):** Định lý Bayes là một công cụ toán học quan trọng trong xác suất thống kê, được sử dụng để tính xác suất của một sự kiện dựa trên thông tin về các điều kiện liên quan đến sự kiện đó. Định lý Bayes được biểu diễn như sau:

  $$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

  Trong đó:

  - $P(A \mid B)$ là xác suất điều kiện của sự kiện A xảy ra khi biết rằng sự kiện B đã xảy ra.
  - $P(B \mid A)$ là xác suất điều kiện của sự kiện B xảy ra khi biết rằng sự kiện A đã xảy ra.
  - $P(A)$ và $P(B)$ là xác suất không điêu kiện của các sự kiện A và B.

- **Xác suất của lớp (Class Probability)**: Trong bài toán phân loại, xác suất của lớp là xác suất mà một điểm dữ liệu thuộc vào từng lớp khác nhau. Trong Naive Bayes, chúng ta tính xác suất này dựa trên tập dữ liệu huấn luyện.

- **Xác suất điều kiện (Conditional Probability)**: Là xác suất của một sự kiện xảy ra dựa trên việc một sự kiện khác đã xảy ra. Trong Naive Bayes, chúng ta cần tính xác suất điều kiện của các đặc trưng (features) cho mỗi lớp.
- **Giả định "ngây thơ"(Naive Assumption)**: Giả định này trong Naive Bayes đề cập đến việc giả định rằng các đặc trưng của dữ liệu là độc lập với nhau. Mặc dù giả định này không thực tế đối với mọi tình huống, nó giúp cho việc tính toán dễ dàng hơn và thường cho kết quả khá tốt trong thực tế.
- **Biến thể của Naive Bayes**: Naive Bayes có các biến thể phổ biến như Multinomial Naive Bayes, Gaussian Naive Bayes và Bernoulli Naive Bayes, mỗi loại được sử dụng cho loại dữ liệu cụ thể và giả định khác nhau về phân phối của dữ liệu.
- **Hậu xác suất (Posterior Probability)**: Là xác suất mà một điểm dữ liệu thuộc vào một lớp cụ thể sau khi đã quan sát các đặc trưng của nó. Trong Naive Bayes, hậu xác suất được tính dựa trên định lý Bayes.

4.2 Bộ phân loại Naive Bayes

Naive Bayes là phương pháp phân loại dựa vào xác suất được sử dụng rộng rãi trong lĩnh vực máy học và nhiều lĩnh vực khác như trong các công cụ tìm kiếm, các bộ lọc mail.
Mục đích chính là làm sao tính được xác suất $\Pr(\mathrm{C_j}, \mathrm{d}')$, xác suất để tài liệu d'nằm trong lớp C_1. Theo luật Bayes, tài liệu d' sẽ được gán vào lớp $\mathrm{C_j}$ nào có xác suất $\Pr(\mathrm{C_j}, \mathrm{d}')$ cao nhất. Công thức để tính $\Pr(\mathrm{C_j}, \mathrm{d}')$ như sau: - $\mathrm{TF}(\mathrm{w_i}, \mathrm{d}')$ là số lần xuất hiện của từ $\mathrm{w_i}$ trong tài liệu d′ - $|\mathrm{d}'|$ là số lượng các từ trong tài liệu d′ - wilà một từ trong không gian đặc trưng F với số chiều là |F| - $\Pr(C_j)$ được tính dựa trên tỷ lệ phần trăm của số tài liệu mỗi lớp tương ứng

$$\Pr(C_j) = \frac{\|C_s\|}{\|C\|} = \frac{\|C_j\|}{\sum_{C \in C'} \|C'\|}$$

trong tập dữ liệu huấn luyện

$$\Pr(w_i \mid C_j) = \frac{1 + TF(w_i, c_j)}{|F| + \sum_{w=|F|} TF(w', c_j)}$$

Ngoài ra còn có các phương pháp NB khác có thể kể ra như ML Naive Bayes, MAP Naive Bayes, Expected Naive Bayes. Nói chung, Naïve Bayes là một công cụ rất hiệu quả trong một số trường hợp.
Thuật toán Naive Bayes dựa trên nguyên lý Bayes được phát biểu như sau:

$$P(Y/X) = \frac{P(XY)}{P(X)} = \frac{P(X/Y)P(Y)}{P(X)}$$

Áp dụng trong bài toán phân loại, các dữ kiện gồm có: - D: tập dứ liệu huấn luyện đã được vector dạng $\vec{x} = (x_1, x_2, \ldots, x_n)$ - C i : phân lớp im với i = {1, 2, .., m} - Các thuộc tính độc lập điều kiện đôi một với nhau.
Theo định lý Bayes:

$$P(C_i \mid X) = \frac{P(X \mid C_i)\, P(C_i)}{P(X)}$$

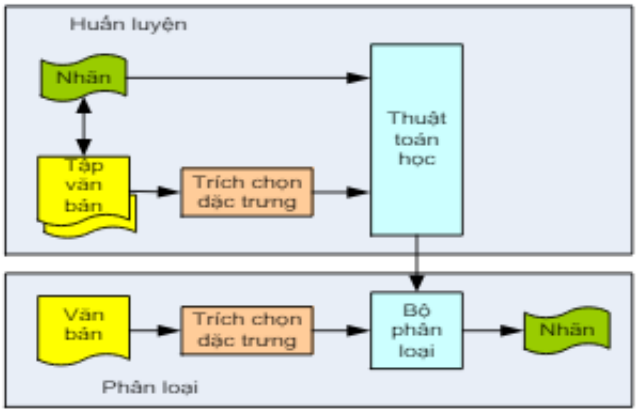
Theo tính chất độc lập điều kiện:

$$P\left(X \mid C_i\right) = \prod_{k=1}^{n} P\left(x_k \mid C_i\right.$$

Trong đó: - P $(C_i \mid X)$: là xác suất thuộc phân lớp i khi biết trước mẩu X - P $(\mathrm{C_i})$: Xác suất phân lớp i $-\mathrm{P}\left(\mathrm{x_k} \mid C_i\right)$: Xác suất thuộc tính thứ k mang giá trị $\mathrm{x_k}$ khi biết X thuộc phân lớp i. Các bước thực hiện thuật toán Naive Bayes:
Bước 1: Huấn luyện Naive Bayes (dựa vào tập dữ liệu), tính P $(\mathrm{C_i})$ và P $\left(\mathrm{x_k} \mid C_i\right)$
Bước 2: Phân lớp $X^{\text{sev}} = (x_1, x_2, \ldots, x_n)$, ta cần tính xác suất thuộc từng phân lớp khi đã biết trước $X^{\text{new.}}$. X^{new} được gán vào lớp có xác suất lớn nhất theo công thức

$$\left(P(Ci) \prod_{k=1}^{\max} P\left(x_k \mid C_i\right)\right)$$

Mô hình tổng quát việc phân loại:



Hình 4: Mô tả bước xây dựng bộ phân lớp

5 Multinomial Logistic

5.1 Các khái niệm

Trong machine learning, mô hình Logistic Multinomial (hay còn được gọi là Softmax Regression) là một phương pháp phân loại được sử dụng khi có ba hoặc nhiều hơn các lớp cần phân loại. Đây là một phương pháp mở rộng của logistic regression, mà chỉ áp dụng cho bài toán phân loại nhị phân.

- **Softmax Function**: Là một hàm kích hoạt được sử dụng trong lớp cuối cùng của mô hình để chuyển đổi các điểm số thành xác suất. Cho một vector đầu vào $z = (z_1, z_2, \ldots, z_k)$, hàm softmax tính toán xác suất cho mỗi lớp i bằng công thức: $P(y = i \mid z) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$ Trong đó k là số lớp, z_i là điểm số cho lớp i, và $P(y = i \mid z)$ là xác suất của lớp i khi biết đầu vào z.

- **One-Hot Encoding**: Một phương pháp mã hóa biến định tính (categorical variable) thành vector số nhị phân, trong đó chỉ có một phần tử có giá trị 1 (đại diện cho lớp) và các phần tử khác có giá trị 0 . Ví dụ, lớp 1 sẽ được mã hóa thành $[1, 0, 0]$, lớp 2 sẽ là $[0, 1, 0]$, và lớp 3 sẽ là $[0, 0, 1]$.
- **Cross-Entropy Loss Function**: Là một hàm mất mát thường được sử dụng trong huấn luyện mô hình phân loại. Đối với một dự đoán xác suất $\hat{y}$ và một nhãn thực tế y, hàm mất mát cross-entropy được định nghĩa là: $H(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$. Trong đó y là vector one-hot encoding của nhãn thực tế và $\hat{y}$ là vector dự đoán xác suất.
- **Gradient Descent Optimization**: Phương pháp tối ưu hóa được sử dụng để điều chỉnh các tham số của mô hình (ví dụ: các trọng số và độ lệch) để giảm thiểu hàm mất mát. Gradient descent tính toán đạo hàm của hàm mất mát theo các tham số và cập nhật chúng theo hướng giảm độ dốc.
- **Multinomial Logistic Regression Model**: Là một mô hình phân loại được sử dụng khi có ba hoặc nhiều hơn các lớp cần phân loại. Mô hình này áp dụng hàm softmax để tính xác suất cho mỗi lớp và sử dụng cross-entropy loss function để đánh giá sự khác biệt giữa xác suất dự đoán và nhãn thực tế.

5.2 Nguyên lý hoạt động

- **Xây dựng mô hình**: Đầu tiên, ta cần xác định kiến trúc của mô hình, bao gồm số lớp đầu vào và số lớp đầu ra (số lớp cần phân loại). Mô hình sẽ bao gồm một lớp đầu vào, một hoặc nhiều lớp ẩn (tùy thuộc vào kiến trúc mạng nơ-ron sử dụng), và một lớp đầu ra với hàm softmax để chuyển đổi điểm số thành xác suất.
- **Biểu diễn dữ liệu đầu vào**: Dữ liệu đầu vào cần được biểu diễn một cách phù hợp để có thể đưa vào mô hình. Đối với mô hình Logistic Multinomial, ta thường sẽ mã hóa các biến đầu vào dưới dạng vector (ví dụ: one-hot encoding cho biến định tính).
- **Huấn luyện mô hình**: Trong quá trình huấn luyện, mô hình sẽ được điều chỉnh các tham số (ví dụ: trọng số và độ lệch) để làm cho các dự đoán của nó càng gần với nhãn thực tế nhất có thể. Để làm điều này, ta cần sử dụng một thuật toán tối ưu hóa như gradient descent để điều chỉnh các tham số dựa trên độ dốc của hàm mất mát.
- **Đánh giá mô hình**: Sau khi mô hình đã được huấn luyện, ta cần đánh giá hiệu suất của nó bằng cách sử dụng tập dữ liệu kiểm tra hoặc cross-validation. Điều này giúp xác định xem mô hình có khả năng tổng quát hóa tốt (tức là hoạt động tốt trên dữ liệu mới không được sử dụng trong quá trình huấn luyện) hay không.
- **Dự đoán**: Khi mô hình đã được đánh giá và chấp nhận, ta có thể sử dụng nó để dự đoán lớp của các điểm dữ liệu mới bằng cách đưa chúng qua mạng nơ-ron và chọn lớp có xác suất cao nhất.

5.3 Cơ sở lý thuyết

Trong mô hình Multinomial Logistic Regression, chúng ta dự đoán xác suất của một điểm dữ liệu bằng cách sử dụng hàm softmax.

$$\hat{y}_k = \frac{e^{T_k}}{\sum_{i=1}^{K} e^{I_1}}$$

Trong đó:

$$\text{z} = z_1, z_2, z_3, \ldots, z_K \text{ với K là số lớp}$$

$\widehat{y}_k$ là xác suất của lớp k

Ta có hàm mât mát cross-entropy thường được sử dụng đề đo lường sự sai lệch giữa phân phối xác suất dự đoán và phân phối xác suất thực tế của các lớp

$$L(y, \hat{y}) = -\sum_{k} y_k \log(\hat{y}_k)$$

y và $\widehat{y}$ là các vector xác suất thực tế và dự đoán của các lớp, $\hat{y}_k$ và y_k lần lượt là xác suất của lớp k trong y và $\hat{y}$

6 ANN

6.1 Giới thiệu

6.1.1 Perceptron cho các hàm logic cơ bản

Chúng ta cùng xét khả năng biểu diễn của perceptron (PLA) cho các bài toán biểu diễn các hàm logic nhị phân: NOT, AND, OR, và XOR . Để có thể sử dụng perceptron (với đầu ra là 1 hoặc -1), chúng ta sẽ thay các giá trị bằng 0 (false) của tại đầu ra của các hàm này bởi -1. Quan sát hàng trên của Hình 5, các điểm hình vuông màu xanh là các điểm có nhãn bằng 1, các điểm hình tròn màu đỏ là các điểm có nhãn bằng -1. Hàng dưới của Hình 5 là các mô hình perceptron với các hệ số tương ứng.
Nhận thấy rằng với các bài toán NOT, AND, và OR, dữ liệu hai lớp là linearly separable, vì vậy ta có thể tìm được các hệ số cho perceptron giúp biểu diễn chính xác mỗi hàm số. Chẳng hạn với hàm NOT, khi x1 = 0, ta có a = sgn(2 × 0 + 1) = 1; khi x1 = 1, a = sgn(2 × 1 + 1) = 1. Trong cả hai trường hợp, đầu ra dự đoán đều giống với đầu ra thực sự. Bạn đọc có thể tự kiểm chứng các hệ số với hàm AND và OR.

6.1.2 Biểu diễn hàm XOR với nhiều perceptron

Hàm XOR, vì dữ liệu không linearly separable, không thể biểu diễn bằng một perceptron. Nếu thay perceptron bằng logistic regression tức thay hàm kích hoạt từ hàm sign sang hàm sigmoid, ta cũng không tìm được các hệ số thỏa mãn, vì về bản chất, logistic regression (hay cả softmax regression) cũng chỉ tạo ra các ranh giới có dạng tuyến tính. Như vậy là các mô hình neural network chúng ta đã biết không thể biểu diễn được hàm số logic đơn giản này.
Nhận thấy rằng nếu cho phép sử dụng hai đường thẳng, bài toán biểu diễn hàm XOR có thể được giải quyết như Hình 6. Các hệ số tương ứng với hai đường thẳng trong Hình 6a được minh họa trên Hình 6b bằng các mũi tên xuất phát từ các điểm màu lục và cam. Đầu ra a (1) 1 bằng 1 với các điểm nằm về phía (+) của đường thẳng $3 - 2x_1 - 2x_2 = 0$, bằng -1 với các điểm nằm về phía (-).

Tương tự, đầu ra a_2 bằng 1 với các điểm nằm về phía (+) của đường thẳng $-1 + 2x_1 + 2x_2 = 0$. Như vậy, hai đường thẳng ứng với hai perceptron này tạo ra hai đầu ra tại các node $a_1^{(1)}, a_2^{(1)}$. Vì hàm XOR chỉ có một đầu ra nên ta cần làm thêm một bước nữa: coi a_1, a_2 như là đầu vào của một perceptron khác. Trong perceptron mới này, input là các node màu cam (đừng quên bias node luôn có giá trị bằng 1), đầu ra là node màu đỏ. Các hệ số được cho trên Hình 6 b. Kiểm tra lại một chút, với các điểm hình vuông xanh (Hình 6a), $a_1^{(1)} = a_2^{(1)} = 1$, khi đó $a^{(2)} = \text{sgn}(-1+1+1) = 1$. Với các điểm hình tròn đỏ, vì $a_1^{(1)} = -a_2^{(1)}$ nên $a^{(2)} = \text{sgn}\left(-1 + a_1^{(1)} + a_2^{(1)}\right) = \text{sgn}(-1) = -1$. Trong cả hai trường hợp, đầu ra dự đoán đều giống với đầu ra thực sự. Vậy, nếu sử dụng ba perceptron tương ứng với các đầu ra $a_1^{(1)}, a_2^{(1)}, a^{(2)}$, ta sẽ biểu diễn được hàm XOR. Ba perceptron kể trên được xếp vào hai layers. Layer thứ nhất: đầu vào - lục, đầu ra - cam. Layer thứ hai: đầu vào - cam, đầu ra - đỏ. Ở đây, đầu ra của layer thứ nhất chính là đầu vào của layer thứ hai. Tổng hợp lại ta được một mô hình mà ngoài layer đầu vào (lục) và đầu ra (đỏ), ta còn có một layer nữa (cam).

Một neural network với nhiều hơn hai layer còn được gọi là multilayer neural network, multilayer perceptrons (MLPs), deep feedforward network hoặc feedforward neural network. Từ feedforward được hiểu là dữ liệu đi thẳng từ đầu vào tới đầu ra theo các mũi tên mà không quay lại ở điểm nào, tức là network có dạng một acyclic graph (đồ thị không chứa chu trình kín). Tên gọi perceptron ở đây có thể gây nhầm lẫn một chút2 , vì cụm từ này để chỉ neural network với nhiều layer và mỗi layer không nhất thiết, nếu không muốn nói là rất hiếm khi, là một hoặc nhiều perceptron. Hàm kích hoạt có thể là các hàm phi tuyến khác thay vì hàm sgn.

Cụ thể hơn, một multilayer neural network là một neural network có nhiều layer, làm nhiệm vụ xấp xỉ mối quan hệ giữa các cặp quan hệ $(\mathbf{x}, \mathbf{y})$ trong tập huấn luyện bằng một hàm số có dạng

$$\mathbf{y} \approx g^{(L)}\left(g^{(L-1)}\left(\ldots\left(g^{(2)}\left(g^{(1)}(\mathbf{x})\right)\right)\right)\right)$$

Trong đó, layer thứ nhất đóng vai trò như hàm $\mathbf{a}^{(1)}g^{(1)}(\mathbf{x})$; layer thứ hai đóng vai trò như hàm $\mathbf{a}^{(2)}g^{(2)}\left(g^{(1)}(\mathbf{x})\right) = f^{(2)}\left(\mathbf{a}^{(1)}\right)$, v.v..

Trong phạm vi cuốn sách, chúng ta quan tâm tới các layer đóng vai trò như các hàm có dạng

$$g^{(l)}\left(\mathbf{a}^{(l-1)}\right) = f^{(l)}\left(\mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right)$$

với $\mathbf{W}^{(l)}, \mathbf{b}^{(l)}$ là ma trận và vector với số chiều phù hợp, $f^{(l)}$ là một hàm số được gọi là hàm kích hoạt (activation function).

Một vài lưu ý:

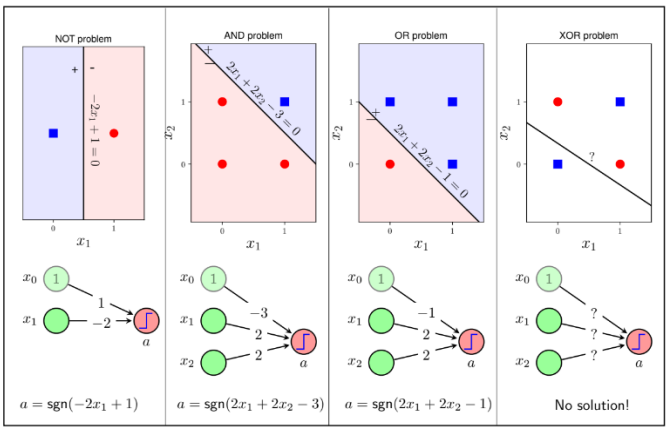
- Để cho đơn giản, chúng ta sử dụng ký hiệu $\mathbf{W}^{(l)T}$ để thay cho $\left(\mathbf{W}^{(l)}\right)^T$ (ma trận chuyển vị). Trong Hình 6 b, ký hiệu ma trận $\mathbf{W}^{(2)}$ được sử dụng, mặc dù đúng ra nó phải là vector, để biểu diễn tổng quát cho trường hợp output layer có thể có nhiều hơn một node. Tương tự với bias $\mathbf{b}^{(2)}$.

- Khác với các chương trước về neural network, khi làm việc với multilayer neural network, ta nên tách riêng phần bias và ma trận hệ số. Điều này đồng nghĩa với việc vector input $\mathbf{x}$ là vector KHÔNG mở rộng.

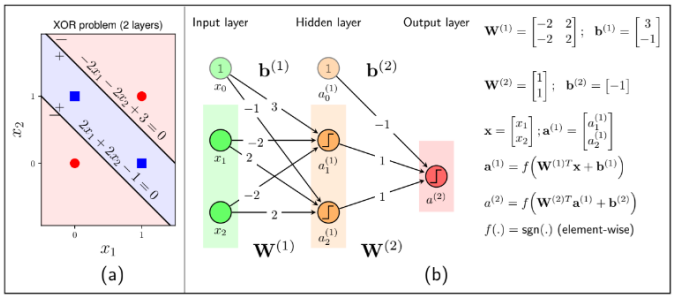
Đầu ra của multilayer neural network loại này ứng với một đầu vào $\mathbf{x}$ có thể được tính theo

$$\begin{aligned}
\mathbf{a}^{(0)} &= \mathbf{x} \\
\mathbf{z}^{(l)} &= \mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad l = 1, 2, \ldots, L \\
\mathbf{a}^{(l)} &= f^{(l)}\left(\mathbf{z}^{(l)}\right), \quad l = 1, 2, \ldots, L \\
\hat{\mathbf{y}} &= \mathbf{a}^{(L)}
\end{aligned}$$

Đây chính là đầu ra dự đoán. Bước này được gọi là feedforward vì cách tính toán được thực hiện từ đầu đến cuối của network. Hàm mất mát thoả mãn đạt giá trị nhỏ khi đầu ra này gần với đầu ra thực sự. Tuỳ vào bài toán, là classification hoặc regression, chúng ta cần thiết kế các hàm mất mát phù hợp.



Hình 5: Biểu diễn các hàm logic cơ bản sử dụng perceptron learning algorithm.



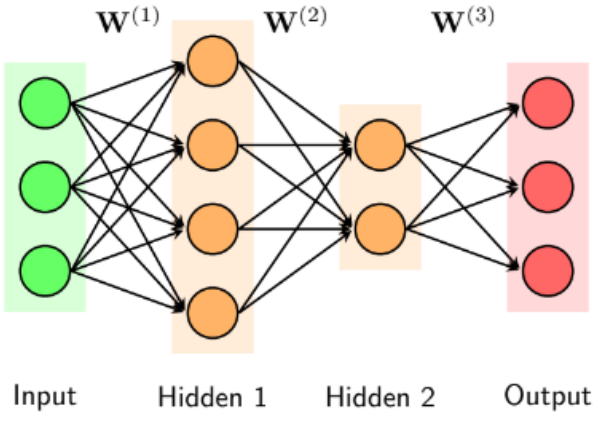
Hình 6: Ba perceptron biểu diễn hàm XOR.

6.2 Các ký hiệu và khái niệm

6.2.1 Layer

Ngoài input layer và output layer, một multilayer neural network có thể có nhiều hidden layer ở giữa. Các hidden layer theo thứ tự từ input layer đến output layer được đánh số thứ thự là hidden layer 1, hidden layer 2, v.v.. Hình 7 là một ví dụ về một multilayer neural network với hai hidden layer.

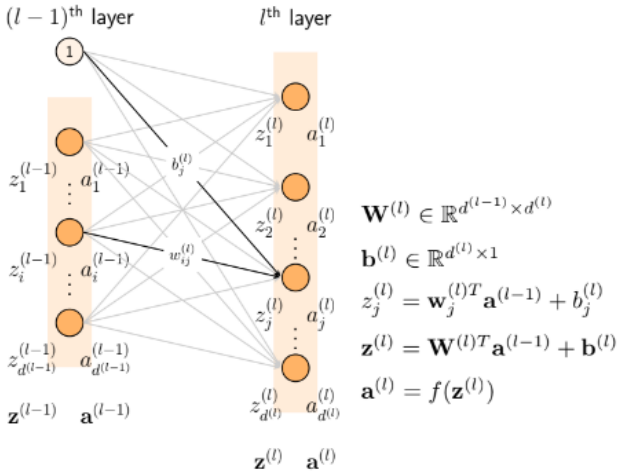
Số lượng layer trong một multilayer neural network, được ký hiệu là L, được tính bằng số hidden layer cộng với một. Khi đếm số layer của một multilayer neural network, ta không tính input layer. Trong Hình 7, L = 3.



Hình 7: MLP với hai hidden layers (các biases đã bị ẩn).

6.2.2 Unit

Quan sát Hình 8, mỗi node hình tròn trong một layer được gọi là một unit. Unit ở input layer, các hidden layer, và output layer được lần lượt gọi là input unit, hidden unit, và output unit. Đầu vào của hidden layer thứ l được ký hiệu bởi $\mathbf{z}^{(l)}$, đầu ra của mỗi unit thường được ký hiệu là $\mathbf{a}^{(l)}$ (thể hiện activation, tức giá trị của mỗi unit sau khi ta áp dụng activation function lên đầu vào $\mathbf{z}^{(l)}$). Đầu ra của unit thứ i trong layer thứ l được ký hiệu là $a_i^{(l)}$. Giả sử thêm rằng số unit trong layer thứ l (không tính bias) là $d^{(l)}$. Vector biểu diễn output của layer thứ l được ký hiệu là $\mathbf{a}^{(l)} \in \mathbb{R}^{d^{(l)}}$.



Hình 8: Các ký hiệu sử dụng trong multilayer neural network.

6.2.3 Weights và Biases

Có L ma trận trọng số cho một multilayer neural network có L layer. Các ma trận này được ký hiệu là $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}, l = 1, 2, \ldots, L$ trong đó $\mathbf{W}^{(l)}$ thể hiện các kết nối từ layer thứ $l-1$ tới layer thứ l (nếu ta coi input layer là layer thứ 0). Cụ thể hơn, phần tử $w_{ij}^{(l)}$ thể hiện kết nối từ node thứ i của layer thứ $(l-1)$ tới node từ j của layer thứ (l). Các bias của layer thứ (l) được ký hiệu là $\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)}}$. Các trọng số này được ký hiệu như trên Hình 8. Khi tối ưu một multilayer neural network cho một công việc nào đó, chúng ta cần đi tìm các weight và bias này. Tập hợp các weight và bias lần lượt được ký hiệu là $\mathbf{W}$ và $\mathbf{b}$.

6.3 Activation function–Hàm kích hoạt

Mỗi output của một layer (trừ input layer) được tính dựa vào công thức

$$\mathbf{a}^{(l)} = f^{(l)}\left(\mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right)$$

Trong đó $f^{(l)}(.)$ là một hàm kích hoạt phi tuyến. Nếu hàm kích hoạt tại một layer là một hàm tuyến tính, layer này và layer tiếp theo có thể rút gọn thành một layer vì hợp của các hàm tuyến tính là một hàm tuyến tính. Hàm kích hoạt thường là một hàm số áp dụng lên từng phần tử của ma trận hoặc vector đầu vào, nói cách khác, hàm kích hoạt thường là element-wise.

6.3.1 Hàm sgn không được sử dụng trong MLP

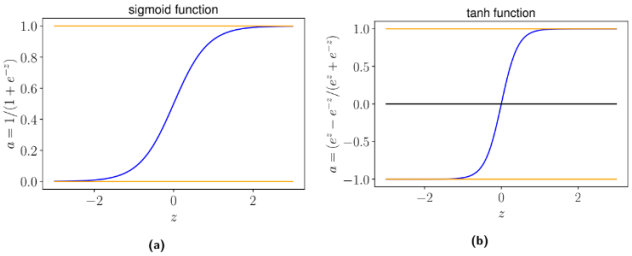
Hàm sgn chỉ được sử dụng trong perceptron. Trong thực tế, hàm sgn không được sử dụng vì và đạo hàm tại hầu hết các điểm bằng 0 (trừ tại điểm 0 không có đạo hàm). Việc đạo hàm bằng 0 này khiến cho các thuật toán dựa trên gradient không hoạt động.

6.3.2 Sigmoid và tanh

Hàm sigmoid có dạng $\text{sigmoid}(z) = 1/(1+\exp(-z))$ với đồ thị như trong Hình 9a. Nếu đầu vào lớn, hàm số sẽ cho đầu ra gần với 1 . Với đầu vào nhỏ (rất âm), hàm số sẽ cho đầu ra gần với 0 . Trước đây, hàm kích hoạt này được sử dụng nhiều vì có đạo hàm rất dẹp. Những năm gần đây, hàm số này ít khi được sử dụng. Một hàm tương tự thường được sử dụng và mang lại hiệu quả tốt hơn là hàm tanh với $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$. Hàm số này có tính chất đầu ra chạy từ -1 đến 1 , khiến cho nó có tính chất zero-centered, thay vì chỉ dương như hàm sigmoid. Gần đây, hàm sigmoid chỉ được sử dụng ở output layer khi yêu cầu của đầu ra là các giá trị nhị phân. Một nhược điểm dễ nhận thấy là khi đầu vào có trị tuyệt đối lớn (rất âm hoặc rất dương), đạo hàm của cả sigmoid và tanh sẽ rất gần với 0 . Điều này đồng nghĩa với việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật khi sử dụng công thức cập nhật gradient desent. Thêm nữa, khi khởi tạo các hệ số cho multilayer neural network với hàm kích hoạt sigmoid, chúng ta phải tránh trường hợp đầu vào một hidden layer nào đó quá lớn, vì khi đó đầu ra của hidden layer đó sẽ rất gần với 0 hoặc 1 , dẫn đến đạo hàm bằng 0 và gradient desent hoạt động không hiệu quả.

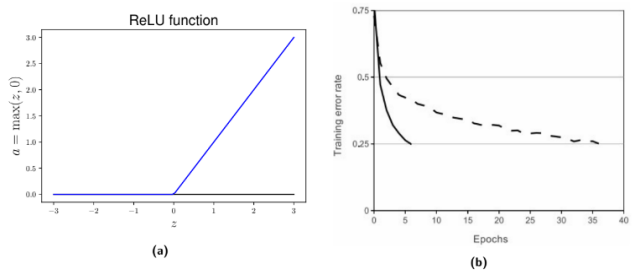
6.3.3 ReLU

ReLU (Rectified Linear Unit) được sử dụng rộng rãi gần đây vì tính đơn giản của nó. Đồ thị của hàm ReLU được minh họa trên Hình 10a. Hàm ReLU có công thức toán học f(z) = max(0, z) - rất



Hình 9: Ví dụ về đồ thị của hàm (a)sigmoid và (b)tanh.

đơn giản, rất lợi về mặt tính toán. Đạo hàm của nó bằng 0 tại các điểm âm, bằng 1 tại các điểm dương. ReLU được chứng minh giúp cho việc huấn luyện các multilayer neural network và deep network (rất nhiều hidden layer) nhanh hơn rất nhiều so với hàm tanh [KSH12]. Hình 10b so sánh sự hội tụ của hàm mất mát khi sử dụng hai hàm kích hoặc ReLU và tanh. Sự tăng tốc này được cho là vì ReLU được tính toán gần như tức thời và gradient của nó cũng được tính cực nhanh.
Mặc dù cũng có nhược điểm đạo hàm bằng 0 với các giá trị đầu vào âm, ReLU được chứng minh bằng thực nghiệm rằng có thể khắc phục việc này bằng việc tăng số hidden unit . ReLU trở thành hàm kích hoạt đầu tiên chúng ta nên thử khi thiết kế một multilayer neural network. Hầu hết các network đều có hàm kích hoạt là ReLU trong các hidden unit, trừ hàm kích hoạt ở output layer phụ thuộc vào đầu ra thực sự của mỗi bài toán (có thể nhận giá trị âm, hoặc nhị phân, v.v.).
Ngoài ra, các biến thể của ReLU như leaky rectified linear unit (Leaky ReLU), parametric rectified linear unit (PReLU) và randomized leaky rectified linear units (RReLU) [XWCL15] cũng được sử dụng và được báo cáo có kết quả tốt. Trong thực tế, trước khi thiết kế, ta thường không biết chính xác hàm kích hoạt nào sẽ cho kết quả tốt nhất. Tuy nhiên, ta nên bắt đầu bằng ReLU, nếu kết quả chưa khả quan thì có thể thay thế bằng các biến thể của nó và so sánh kết quả.



Hình 10: Hàm ReLU và tốc độ hội tụ khi so sánh với hàm tanh.

6.4 Backpropagation

Phương pháp phổ biến nhất để tối ưu multilayer neural network chính là gradient descent (GD). Để áp dụng GD, chúng ta cần tính được đạo hàm của hàm mất mát theo từng ma trận trọng số $\mathbf{W}^{(l)}$ và vector bias $\mathbf{b}^{(l)}$.
Giả sử $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$ là một hàm mất mát của bài toán, trong đó $\mathbf{W}, \mathbf{b}$ là tập hợp tất cả các ma trận trọng số giữa các layer và vector bias của mỗi layer. X, Y là cặp dữ liệu huấn luyện với mỗi cột tương ứng với một điểm dữ liệu. Để có thể áp dụng các phương pháp gradient descent, chúng ta cần tính được các $\nabla_{\mathbf{W}^{(l)}} J; \nabla_{\mathbf{b}^{(l)}} J, \forall l = 1, 2, \ldots, L$.

Nhắc lại quá trình feedforward

$$\begin{aligned}\mathbf{a}^{(0)} &= \mathbf{x}\\ \mathbf{z}^{(l)} &= \mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad l = 1, 2, \ldots, L\\ \mathbf{a}^{(l)} &= f^{(l)}\left(\mathbf{z}^{(l)}\right), \quad l = 1, 2, \ldots, L\\ \hat{\mathbf{y}} &= \mathbf{a}^{(L)}\end{aligned}$$

Một ví dụ của hàm mất mát là hàm mean square error (MSE):

$$J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{y}_n - \hat{\mathbf{y}}_n\|_2^2 = \frac{1}{N}\sum_{n=1}^{N}\left\|\mathbf{y}_n - \mathbf{a}_n^{(L)}\right\|_2^2$$

với N là số cặp dữ liệu $(\mathbf{x}, \mathbf{y})$ trong tập huấn luyện. Theo các công thức này, việc tính toán trực tiếp các giá trị đạo hàm là cực kỳ phức tạp vì hàm mất mát không phụ thuộc trực tiếp vào các ma trận hệ số và vector bias. Phương pháp phổ biến nhất được dùng có tên là backpropagation giúp tính đạo hàm ngược từ layer cuối cùng đến layer đầu tiên. Layer cuối cùng được tính toán trước vì nó gần gũi hơn với đầu ra dư đoán và hàm mất mát. Việc tính toán đạo hàm của các ma trận hệ số trong các layer trước được thực hiện dựa trên một quy tắc chuỗi quen thuộc cho đạo hàm của hàm hợp.
Stochastic gradient descent có thể được sử dụng để tính gradient cho các ma trận trọng số và biases dựa trên một cặp điểm training $\mathbf{x}, \mathbf{y}$. Để cho đơn giản, ta coi J là hàm mất mát nếu chỉ xét cặp điểm này, ở đây J là hàm mất mát bất kỳ, không chỉ hàm MSE như ở trên. Đạo hàm của hàm mất mát theo chỉ một thành phần của ma trận trọng số của output layer

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = e_j^{(L)} a_i^{(L-1)}$$

trong đó $e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}$ thường là một đại lượng không quá khó để tính toán và $\frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = a_i^{(L-1)}$ vì $z_j^{(L)} = \mathbf{w}_j^{(L)T}\mathbf{a}^{(L-1)} + b_j^{(L)}$. Tương tự, đạo hàm của hàm mất mát theo bias của layer cuối cùng là

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} \cdot \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}$$

Với đạo hàm theo hệ số ở các lớp l thấp hơn, chúng ta hãy xem Hình 11. Ở đây, tại mỗi unit, đầu vào z và đầu ra a được viết riêng để chúng ta tiện theo dõi.
Dựa vào Hình 11, bằng quy nạp ngược từ cuối, ta có thể tính được

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} a_i^{(l-1)}$$

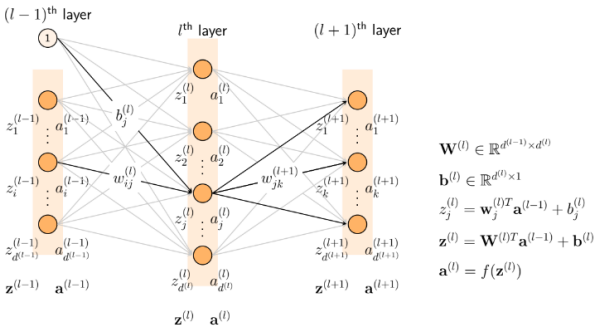
với

$$\begin{aligned}e_j^{(l)} &= \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}\\ &= \left(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}}\right) f^{(l)'}\left(z_j^{(l)}\right) = \left(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)}\right) f^{(l)'}\left(z_j^{(l)}\right)\end{aligned}$$

trong đó $\mathbf{e}^{(l+1)} = \left[e_1^{(l+1)}, e_2^{(l+1)}, \ldots, e_{d^{(l+1)}}^{(l+1)}\right]^T \in \mathbb{R}^{d^{(l+1)}\times 1}$ và $\mathbf{w}_{j:}^{(l+1)}$ được hiểu là hàng thứ j của ma trận $\mathbf{W}^{(l+1)}$ (chú ý dấu hai chấm, khi không có dấu này, chúng ta mặc định dùng nó để ký hiệu cho vector cột). Dấu $\sum$ tính tổng ở dòng thứ hai trong phép tính trên xuất hiện vì $a_j^{(l)}$ đóng góp vào việc tính tất cả các $z_k^{(l+1)}, k = 1, 2, \ldots, d^{(l+1)}$. Biểu thức đạo hàm ngoài dấu ngoặc lớn là vì $a_j^{(l)} = f^{(l)}\left(z_j^{(l)}\right)$. Tới đây, ta có thể thấy rằng việc activation function có đạo hàm đơn giản sẽ có ích rất nhiều trong việc tính toán. Với cách làm tương tự, bạn đọc có thể suy ra

$$\frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)}.$$

Nhận thấy rằng trong các công thức trên đây, việc tính các $e_j^{(l)}$ đóng một vài trò quan trọng. Hơn nữa, để tính được giá trị này, ta cần tính được các $e_j^{(l+1)}$. Nói cách khác, ta cần tính ngược các giá trị này từ cuối. Cái tên backpropagation cũng xuất phát từ việc này.



Hình 11: Mô phỏng cách tính backpropagation. Layer cuối có thể là output layer.

6.4.1 Backpropagation cho batch (mini-batch) gradient descent

Nếu ta muốn thực hiện batch hoặc mini-batch GD thì thế nào? Trong thực tế, mini-batch GD được sử dụng nhiều nhất với các bài toán mà tập huấn luyện lớn. Nếu lượng dữ liệu là nhỏ, batch GD trực tiếp được sử dụng. Khi đó, cặp (input, output) sẽ ở dạng ma trận (X, Y). Giả sử rằng mỗi lần tính toán, ta lấy N dữ liệu để tính toán. Khi đó, $\mathbf{X} \in \mathbb{R}^{d^{(0)}\times N}, \mathbf{Y} \in \mathbb{R}^{d^{(L)}\times N}$. Với $d^{(0)} = d$ là chiều của dữ liệu đầu vào (không tính bias).

7 Thực nghiệm

7.1 Dữ liệu

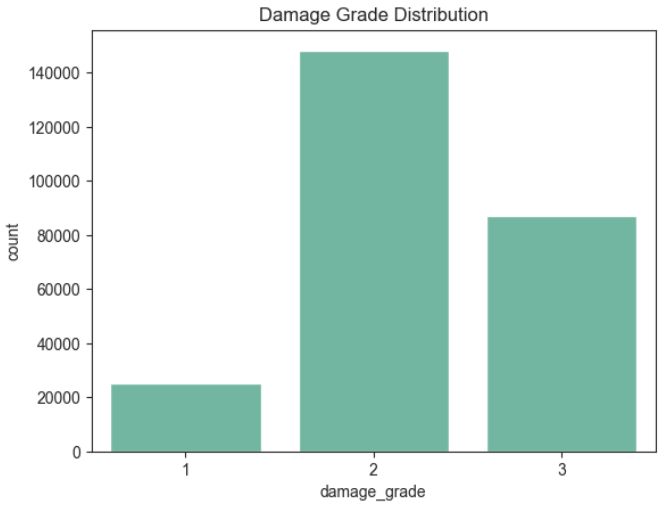
3 tập dữ liệu gồm:

- train_values: 260,601 bản ghi và 39 trường dữ liệu: chứa thông tin các tòa nhà
- train_labels: 260,601 bản ghi và 2 trường dữ liệu: chứa mức độ thiệt hại của các tòa nhà có trong train_values.

- test_values: 86,868 bản ghi × 39 trường dữ liệu chứa thông tin các tòa nhà (Bộ dữ liệu này sẽ được sử dụng để so sánh kết quả chạy mô hình với kết quả chuẩn đã được xác nhận).

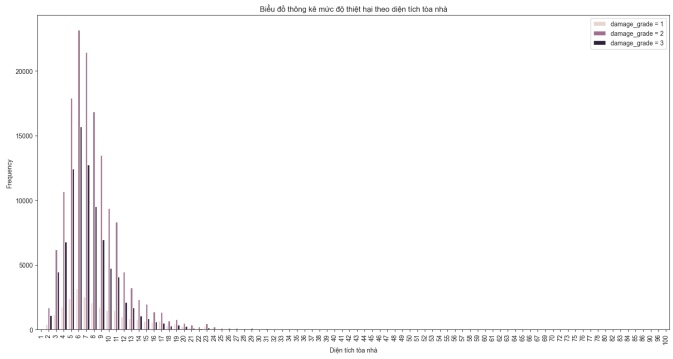
Integer:	**Binary:**	**Categorical:**
1. geo_level_1_id	13. has_superstructure_adobe_mud	6. land_surface_condition
2. geo_level_2_id	14. has_superstructure_mud_mortar_stone	7. foundation_type
3. geo_level_3_id	15. has_superstructure_stone_flag	8. roof_type
4. count_floors_pre_eq	16. has_superstructure_cement_mortar_stone	9. ground_floor_type
5. age	17. has_superstructure_mud_mortar_brick	10. other_floor_type
6. area_percentage	18. has_superstructure_cement_mortar_brick	11. position
7. height_percentage	19. has_superstructure_timber	12. plan_configuration
8. count_families	20. has_superstructure_bamboo	13. legal_ownership_status
	21. has_superstructure_rc_non_engineered	
	22. has_superstructure_rc_engineered	
	23. has_superstructure_other	
	24. has_secondary_use	
	25. has_secondary_use_agriculture	
	26. has_secondary_use_hotel	
	27. has_secondary_use_rental	
	28. has_secondary_use_institution	
	29. has_secondary_use_school	
	30. has_secondary_use_industry	
	31. has_secondary_use_health_post	
	32. has_secondary_use_gov_office	
	33. has_secondary_use_use_police	
	34. has_secondary_use_other	

Hình 12: Các trường được chia theo kiểu dữ liệu

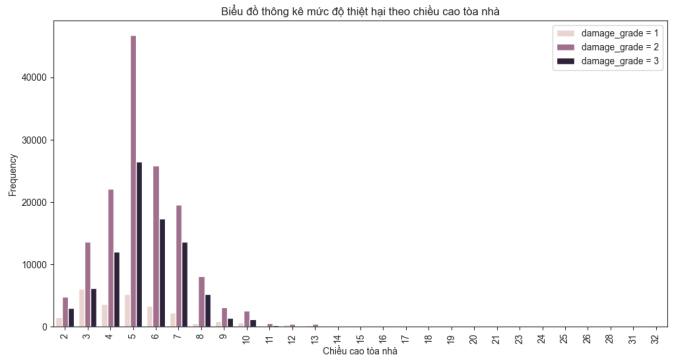


Hình 13: Có thể dễ dàng nhận thấy rằng tập dữ liệu rất mất cân bằng. Lý do khiến tập dữ liệu này mất cân bằng có thể là do dữ liệu được thu thập sau trận động đất ở Nepal năm 2015 và rất có thể mức độ thiệt hại của các tòa nhà có mối tương quan cao với cường độ của trận động đất.

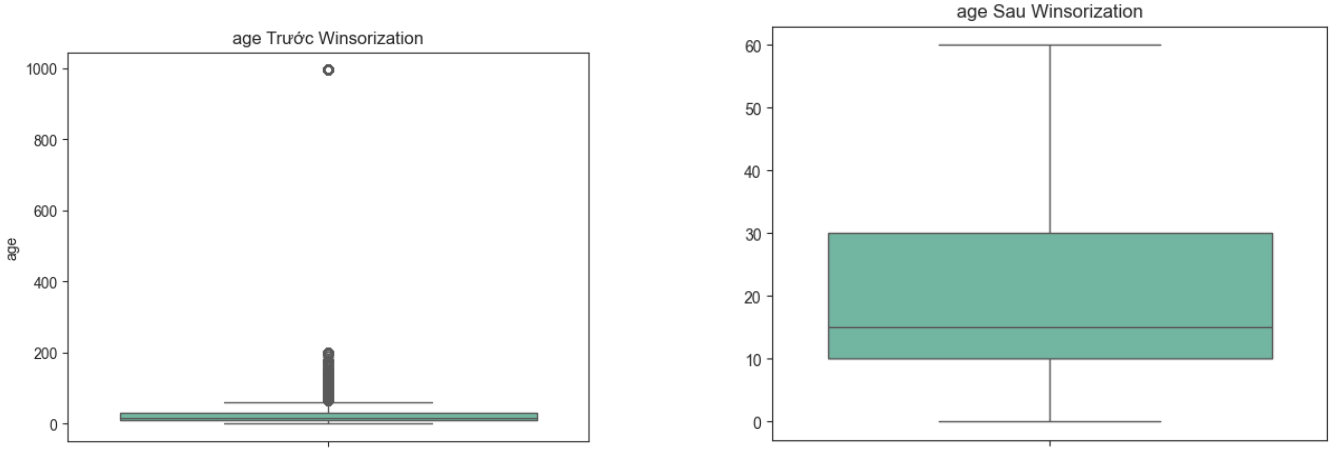
Dữ liệu khá "clean", không chứa các giá trị đặc biệt. Tuy nhiên, vẫn còn nhiều ngoại lệ trong một số trường dữ liệu nên được xử lí.



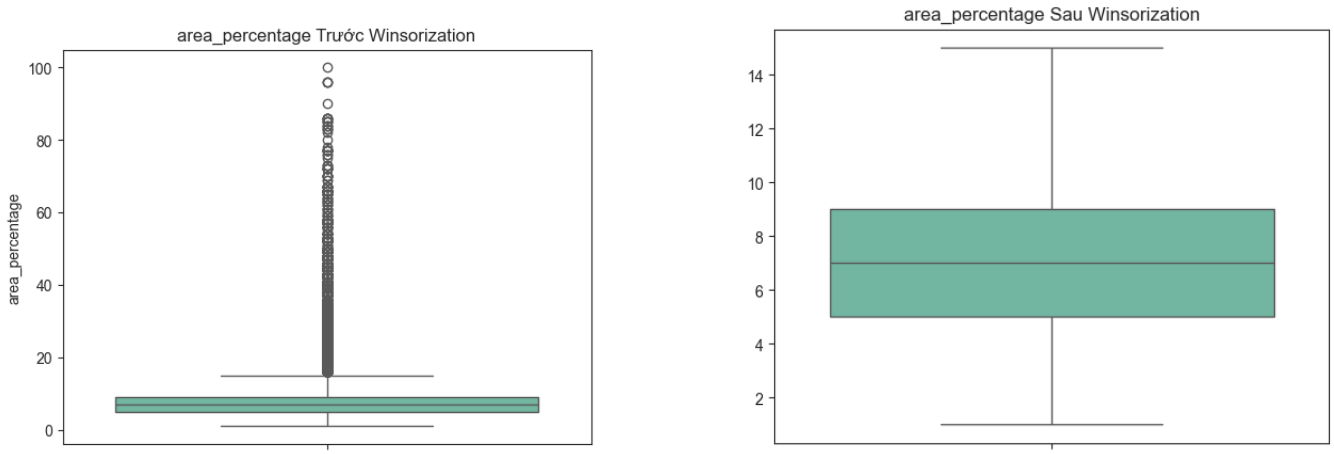
Hình 14: Diện tích tiêu chuẩn của diện tích tòa nhà là dưới 20% đối với hầu hết các tòa nhà



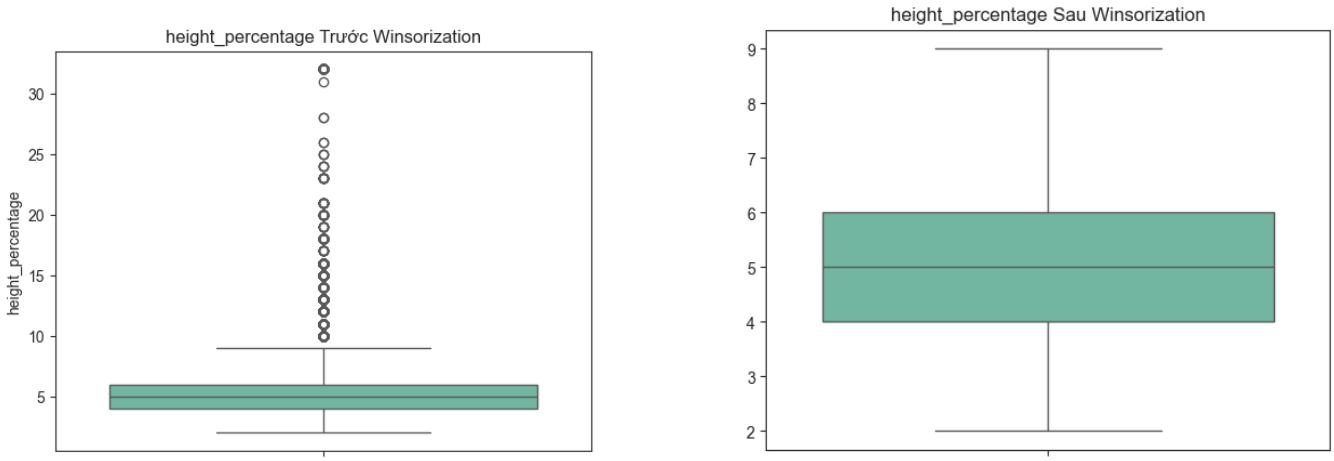
Hình 15: Hầu hết dữ liệu cho chiều cao tòa nhà nhỏ hơn 10% nhưng chỉ có một số ít tòa nhà có giá trị lên tới 32%.



Hình 16: Tuổi của tòa nhà sau khi được Winsorization



Hình 17: Diện tích của tòa nhà sau khi được Winsorization



Hình 18: Chiều cao của tòa nhà sau khi được Winsorization

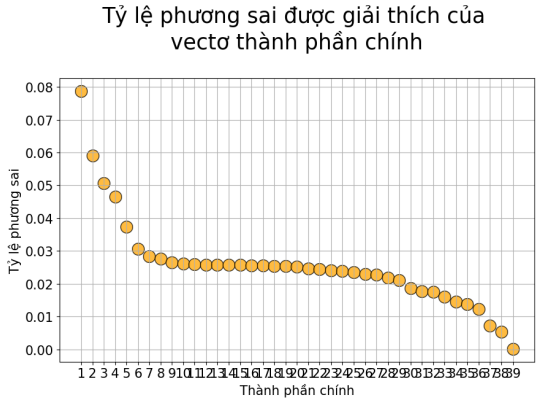
7.2 PCA

Trước khi PCA, ta cần chuẩn hóa các trường về dạng số. Sử dụng LabelEncoder() từ thư viện scikit-learn sẽ giải quyết vấn đề này.

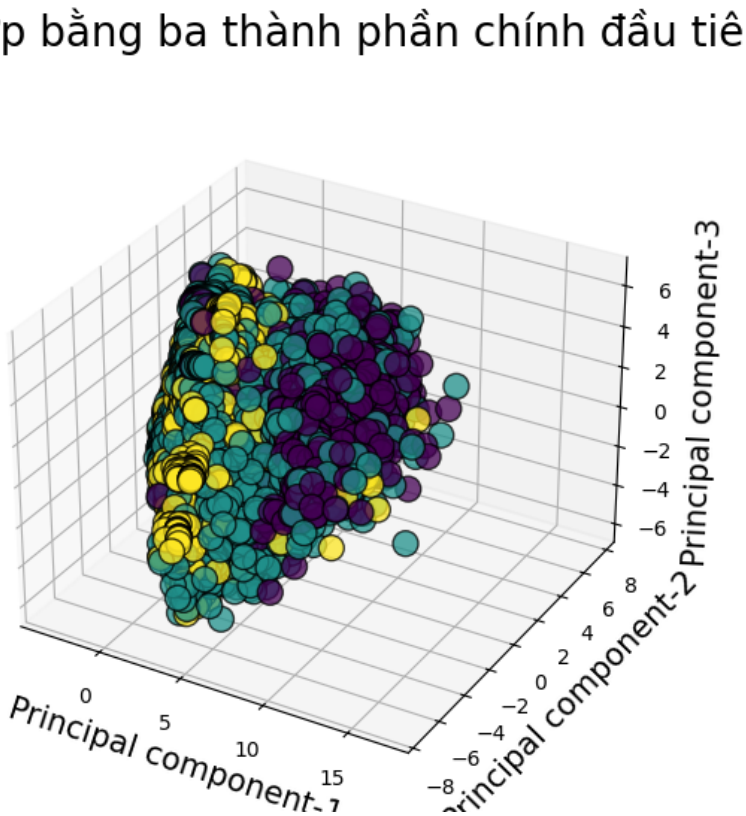
```
from sklearn.preprocessing import LabelEncoder
def convert(df):
    string_columns = [i for i, dtype in enumerate(df.dtypes) if dtype == 'object']

    label_encoders = []
    for col in string_columns:
        label_encoder = LabelEncoder()
        df.iloc[:, col] = label_encoder.fit_transform(df.iloc[:, col])
        label_encoders.append(label_encoder)
```

PCA yêu cầu chia tỷ lệ/chuẩn hóa dữ liệu để hoạt động bình thường. Ta sử dụng StandardScaler() từ thư viện scikit-learn.
X = scaler.fit_transform(X): Ở đây, dữ liệu đầu vào X được chuẩn hóa bằng cách sử dụng scaler đã được tạo trước đó. Phương thức fit_transform của StandardScaler sẽ tính toán giá trị trung bình và độ lệch chuẩn của mỗi đặc trưng từ X và sau đó chuẩn hóa dữ liệu dựa trên các giá trị này.



Hiển thị trực quan 3 thành phần chính sau khi thực hiện giảm chiều:



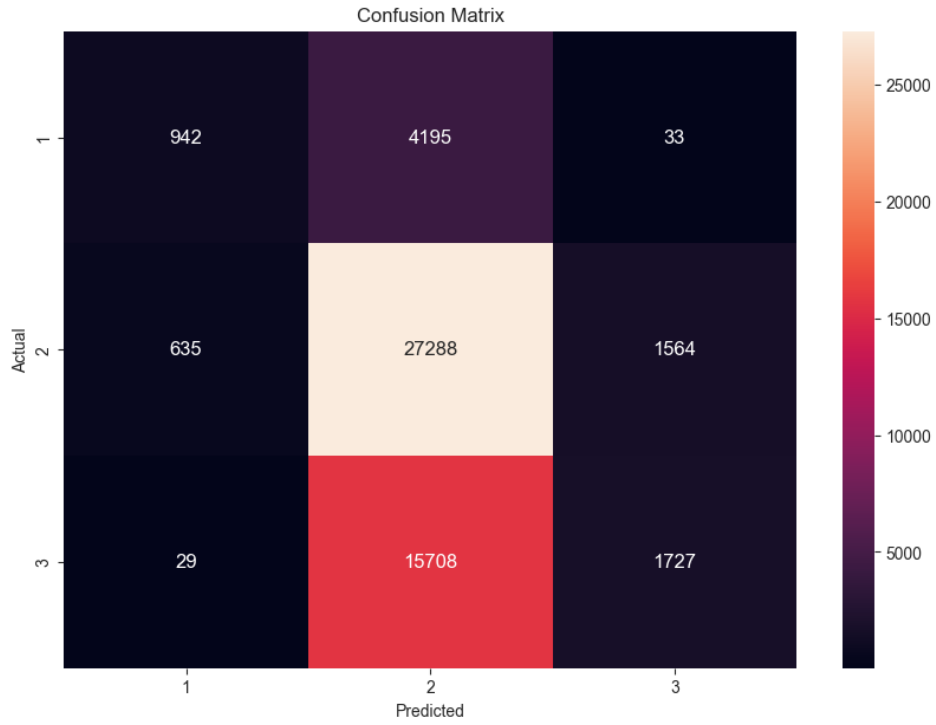
7.3 Bài toán dự đoán và phân loại dựa trên dữ liệu nguyên bản

7.3.1 Hồi quy tuyến tính

Ta chia tập dữ liệu huấn luyện thành 80% train và 20% test

```
X = train_values
y = train_labels
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
    random_state = 42)
```

Tiếp theo sử dụng LinearRegression() từ thư viện scikit-learn để thực hiện các bước hồi quy tuyến tính kết quả đo độ chính xác accuracy cho ra kết quả là: 0.5747587344832218.
Để có thể đo accuracy, ta phải chuẩn hóa lại y_pred bằng cách làm tròn dữ liệu để so sánh nhãn
Sử dụng độ chính xác (accuracy) để đánh giá mô hình Linear Regression không phải là cách tiếp cận tốt nhất vì:
1. Linear Regression là mô hình hồi quy: Mục tiêu của nó là dự đoán một giá trị liên tục, không phải phân loại các đối tượng vào các lớp. Do đó, việc sử dụng độ chính xác, một chỉ số thường được sử dụng cho các mô hình phân loại, không phản ánh chính xác hiệu suất của mô hình hồi quy.
2. Giá trị độ chính xác 0.5747587344832218 chỉ ra rằng: Khi chuyển đổi kết quả dự đoán từ mô hình hồi quy thành các lớp bằng cách làm tròn, chỉ có khoảng 57% dự đoán được phân loại chính xác. Tuy nhiên, con số này không cung cấp thông tin về mức độ sai lệch của các dự đoán so với giá trị thực tế trong bối cảnh hồi quy.



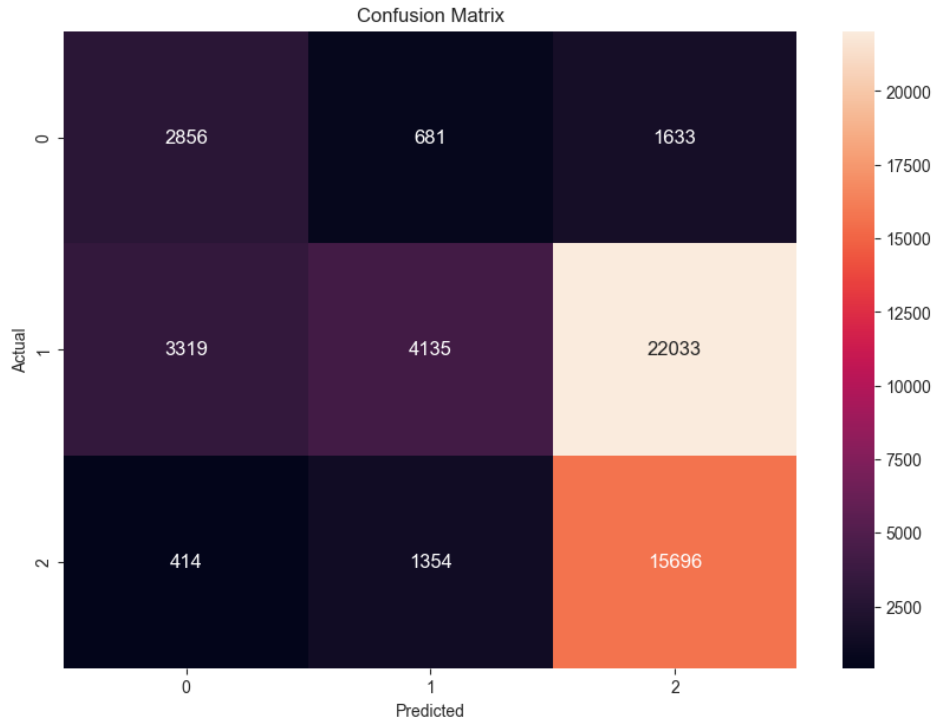
7.3.2 Bài toán phân loại

Như yêu cầu, trước tiên ta cần thay thế ba mức thiệt hại (1, 2, 3) trong labels thành các class 0, 1, 2 – tương ứng - và chuyển bài toán sang dạng phân loại dữ liệu. Việc này dễ dàng được thực hiện với LabelEncoder.

```python
def changeLabel(df):
    label_encoder = LabelEncoder()
    label_encoder.fit(df["damage_grade"])
    df["damage_class"] = label_encoder.transform(df["damage_grade"])
```

Naïve Bayes

Tương tự như Linear Regression, ta chia tập dữ liệu huấn luyện thành 80% train và 20% test. Sau đó sử dụng GaussianNB() từ thư viện scikit-learn để thực hiện mô hình, đồng thời sử dụng các độ đo accuracy, precision_score, recall_score để đánh giá.
Naïve Bayes Training Time: 0.6860580444335938 seconds
Naïve Bayes Accuracy: 0.43527560867980275
Naïve Bayes Precision: 0.5007960654944719
Naïve Bayes Recall: 0.530470751094987



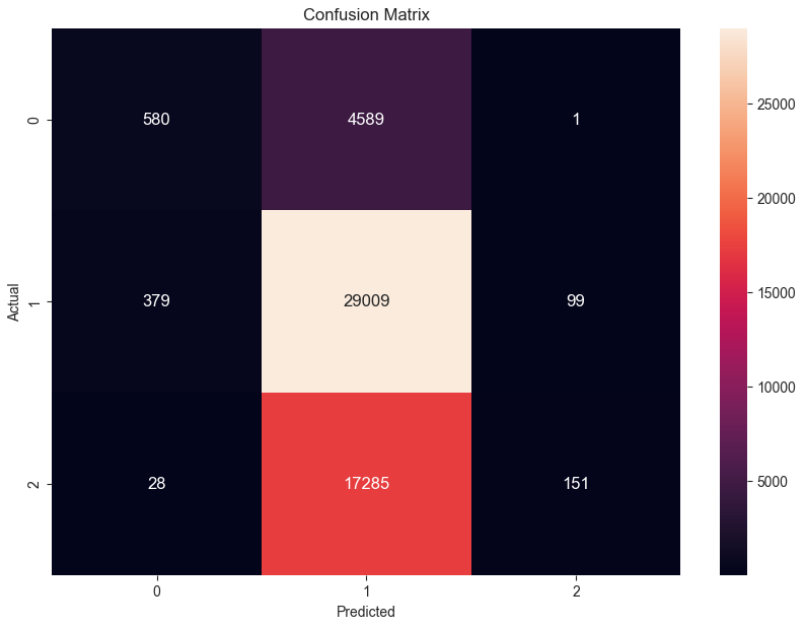
Multinomial Logistic

Ta sử dụng LogisticRegression từ thư viện scikit-learn để thực hiện mô hình với các tham số:

- solver='lbfgs': Đây là thuật toán được sử dụng để tối ưu hóa hàm mất mát trong quá trình huấn luyện mô hình. 'lbfgs' là một trong những thuật toán phổ biến được sử dụng cho hồi quy logistic. Thuật toán này thường hoạt động tốt cho các bài toán hồi quy nhỏ đến trung bình.

- multi_class='multinomial': Định nghĩa cách mà mô hình xử lý bài toán đa lớp. Trong trường hợp này, 'multinomial' cho biết mô hình sẽ sử dụng hồi quy softmax, phù hợp cho bài toán phân loại nhiều lớp.
- max_iter=1000: Đây là số lượng lớn nhất các vòng lặp được thực hiện trong quá trình tối ưu hóa hàm mất mát. Điều này giúp đảm bảo rằng thuật toán tối ưu hóa có đủ lượng thời gian để hội tụ đến giải pháp tốt nhất trong số lượng vòng lặp đã cho. Nếu thuật toán không hội tụ sau khi số lần lặp này, một cảnh báo có thể được hiển thị, và ta có thể cần tăng số lần lặp hoặc điều chỉnh các siêu tham số khác của mô hình.

Kết quả đánh giá mô hình:
Logistic Regression Training Time: 103.58686208724976 seconds
Multinomial Logistic Regression Accuracy: 0.57059533454461733
Multinomial Logistic Regression Precision: 0.5864482539030385
Multinomial Logistic Regression Recall: 0.3682071704736601



7.4 Bài toán phân loại dữ liệu dựa trên dữ liệu giảm chiều

7.4.1 Naive Bayes

Chỉ số Đánh Giá	Dữ Liệu Nguyên Bản	Dữ Liệu Đã Giảm Chiều (PCA)
Độ Chính Xác (Accuracy)	56.3%	45.9%
Độ Chính Xác Tích Cực (Precision)	33.2%	50.6%
Độ Nhạy (Recall)	56.3%	45.9%

Kết quả cho thấy sự khác biệt đáng kể giữa hiệu suất của mô hình Naive Bayes khi sử dụng dữ liệu gốc so với dữ liệu đã được giảm chiều bằng PCA (Principal Component Analysis).

- **Độ chính xác (Accuracy)**: Độ chính xác của mô hình trên dữ liệu gốc là 56.3%, cao hơn so với độ chính xác 45.9% trên dữ liệu đã qua PCA. Điều này cho thấy việc giảm chiều dữ liệu bằng PCA có thể làm giảm khả năng dự đoán chính xác của mô hình.
- **Độ chính xác tích cực (Precision)**: Độ chính xác tích cực của mô hình trên dữ liệu gốc là 33.2%, thấp hơn so với độ chính xác tích cực 50.6% trên dữ liệu đã qua PCA. Điều này cho thấy, mặc dù độ chính xác tổng thể giảm, nhưng khả năng dự đoán chính xác các trường hợp tích cực của mô hình lại tăng lên khi sử dụng dữ liệu đã qua PCA.
- **Độ nhạy (Recall)**: Độ nhạy của mô hình trên cả dữ liệu gốc và dữ liệu PCA đều tương ứng với độ chính xác của mô hình trên từng loại dữ liệu, lần lượt là 56.3% và 45.9%. Điều này cho thấy tỷ lệ dự đoán đúng các trường hợp tích cực so với tổng số trường hợp tích cực thực sự giảm sau khi áp dụng PCA.

7.4.2 Multinomial Logistic

Chỉ số Đánh Giá	Dữ Liệu Nguyên Bản	Dữ Liệu Đã Giảm Chiều (PCA)
Độ Chính Xác (Accuracy)	56.6%	56.9%
Độ Chính Xác Tích Cực (Precision)	32.0%	54.2%
Độ Nhạy (Recall)	56.6%	56.9%

Dựa trên bảng số liệu đã cho, ta có thể nhận xét về kết quả của mô hình phân loại dữ liệu multi logistic trên dữ liệu nguyên bản và dữ liệu giảm chiều PCA như sau:
Dữ liệu nguyên bản

- Độ chính xác (Accuracy) và Độ nhạy (Recall) đều là 56.6%, khá thấp.
- Độ chính xác tích cực (Precision) chỉ đạt 32.0%, rất thấp.

Dữ liệu giảm chiều PCA

- Độ chính xác (Accuracy) và Độ nhạy (Recall) đều là 56.9%, cải thiện nhẹ so với dữ liệu nguyên bản.
- Độ chính xác tích cực (Precision) đạt 54.2%, cải thiện đáng kể so với dữ liệu nguyên bản.

7.4.3 ANN

Dữ liệu sau khi được giảm chiều PCA sẽ được cho xử lí tại ANN với cấu trúc cơ bản như sau:

- Dense(64, activation='relu'): Tầng ẩn thứ nhất với 64 neurons và hàm kích hoạt ReLU.
- Dense(32, activation='relu'): Tầng ẩn thứ hai với 32 neurons và hàm kích hoạt ReLU.

- Dense(3, activation='softmax'): Tầng đầu ra với 3 neurons và hàm kích hoạt Softmax.

Kết quả thực nghiệm:
Neural Network accuracy on PCA data: 0.6284798833483625
Neural Network precision on PCA data: 0.6335206554476792
Neural Network recall on PCA data: 0.6284798833483625

Dựa trên các kết quả nghiên cứu, có thể đưa ra một số nhận xét như sau:

1. Mô hình Neural Network (ANN) thường cho kết quả tốt hơn so với các mô hình truyền thống như Naive Bayes hay Logistic Regression khi áp dụng trên dữ liệu đã qua giảm chiều PCA. Nguyên nhân là do ANN có khả năng học các mối quan hệ phi tuyến phức tạp trong dữ liệu tốt hơn.

2. Mặc dù giảm chiều PCA giúp giảm số lượng tham số của mô hình và tăng tốc huấn luyện, nhưng nếu giảm quá mức sẽ phản tác dụng do mất mát thông tin. Cần cân bằng giữa tốc độ và độ chính xác khi lựa chọn số chiều phù hợp.

3. Các mô hình đơn giản như Naive Bayes, Logistic có xu hướng bị ảnh hưởng nhiều hơn bởi việc giảm chiều PCA so với các mô hình phức tạp như Neural Network. Do đó, với những mô hình này, chỉ nên áp dụng PCA với mức độ vừa phải.

8 Kết luận

Dựa trên các kết quả cụ thể từ các mô hình và báo cáo phân loại:
1. PCA (Phân tích thành phần chính):
• Tỷ lệ phương sai giải thích (explained variance ratio) cho các thành phần chính không cao, chỉ khoảng 20% của phương sai tổng thể. Điều này cho thấy rằng việc sử dụng chỉ một số lượng nhỏ các thành phần chính không đủ để giải thích sự biến động lớn trong dữ liệu.

2. Multinomial Logistic Regression:
• Đối với dữ liệu nguyên bản, độ chính xác trên tập validation là khoảng 56.6%, đồng nghĩa với việc mô hình đưa ra dự đoán chính xác cho khoảng hơn một nửa số lượng mẫu trong tập dữ liệu kiểm định. Với dữ liệu đã giảm chiều cho ra 56.9%, các chỉ số Precision và Recall cũng cao hơn tương đối Nhìn chung, việc giảm chiều dữ liệu bằng PCA đã cải thiện được Độ chính xác tích cực của mô hình, nhưng không làm tăng đáng kể Độ chính xác và Độ nhạy. Điều này cho thấy PCA có thể hữu ích trong việc lọc bỏ nhiều yếu tố phụ và tập trung vào các đặc trưng quan trọng, giúp mô hình phân loại chính xác hơn các lớp dữ liệu tích cực.

3. Naïve Bayes:
Như đã thấy ở số liệu bên trên, nhìn chung việc áp dụng PCA để giảm chiều dữ liệu có thể làm giảm độ chính xác tổng thể của mô hình Naive Bayes nhưng lại tăng độ chính xác tích cực. Điều này có thể hữu ích trong các tình huống mà việc giảm số lượng sai lệch tích cực (giảm false positives) là quan trọng hơn việc duy trì độ chính xác tổng thể. Tuy nhiên, việc giảm chiều dữ liệu cũng có thể dẫn đến việc giảm độ nhạy, tức là tỷ lệ dự đoán đúng các trường hợp tích cực thực sự giảm. Điều này cần được cân nhắc kỹ lưỡng khi quyết định sử dụng PCA trong các mô hình phân loại.

Đánh giá tổng quan:
• Cả hai mô hình đều cho thấy hiệu suất tương đối tốt trong việc phân loại dữ liệu, nhưng Multinomial Logistic có vẻ hiệu quả hơn so với Naïve Bayes. Tuy nhiên cả hai mô hình chưa thực sự được tối ưu cao.
• Sự cải thiện của Multinomial Logistic có thể được giải thích bởi cách mà nó xử lý các dữ liệu số học và tập dữ liệu lớn.

4. Neural Network:
Mô hình ANN kết hợp với kỹ thuật giảm chiều PCA đã đạt kết quả khá tốt với độ chính xác trên 62%. Mô hình Neural Network thường phù hợp và cho kết quả tốt hơn khi áp dụng PCA so với các mô hình truyền thống. Tuy nhiên, vẫn cần cải thiện thêm để nâng cao hiệu quả của mô hình. Việc lựa chọn số chiều giảm phù hợp và tinh chỉnh các tham số của ANN sẽ giúp tăng độ chính xác của bài toán phân loại.