Dự đoán áp lực của máy thở bằng XGBoost

Bùi Trí Dũng [1] **Võ Khánh An** [1] **Phạm Ngọc Tân** [1]

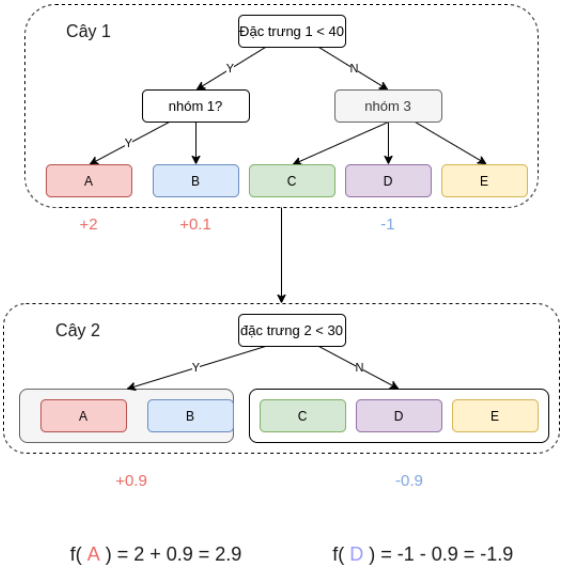
Tóm tắt

Hiện nay, đại dịch COVID-19 vẫn đang diễn biến phức tạp và gây ra những hậu quả vô cùng nặng nề. Những bệnh nhân bị trở nặng cần phải sử dụng máy thở dưới sự điều chỉnh và giám sát từ các nhân viên y tế có chuyên môn. Tuy vậy, việc số ca nhiễm tăng lên theo cấp số nhân tại một số khu vực trên thế giới đã đẩy hệ thống y tế rơi vào tình trạng quá tải. Do đó, việc xây dựng những máy thở có thể tự động điều chỉnh thông số trong quá trình điều trị là một việc làm vô cùng thiết thực trong giai đoạn này. Trong bài này, chúng tôi sẽ tiến hành dự đoán áp lực của máy thở lên bệnh nhân qua những thông số của máy thở và tình trạng của bệnh nhân bằng thuật toán XGBoost. Bên cạnh các kỹ thuật học sâu, XGBoost là một thuật toán nhằm giải quyết các bài toán học có giám sát cho độ chính xác khá cao và đã đạt nhiều chiến thắng trong các cuộc thi trên Kaggle. Hơn thế nữa, XGBoost còn có tốc độ huấn luyện nhanh và khả năng tính toán song song nhằn tận dụng sức mạnh của GPU. Tất cả những nghiên cứu của chúng tôi đều được công khai tại: https://github.com/DTA-UIT/Ventilator_Pressure_Prediction

1. Giới thiệu

Thông thường, khi một bệnh nhân gặp phải các vấn đề về hô hấp thì các bác sĩ sẽ tiến hành sử dụng máy thở để bơm Oxy vào bên trong phổi của bệnh nhân thông qua một đường ống được luồng vào khí quản. Tuy nhiên, việc sử dụng thành thạo máy thở là một công việc đòi hỏi rất nhiều kiến thức y khoa. Điều này là một trở ngại cực lớn trong bối cảnh tình hình đại dịch COVID-19 ngày càng trở nên căng thẳng tại nhiều quốc gia trên thế giới, dẫn đến hệ thống y tế bị quá tải và nhiều nơi không có đủ nguồn nhân lực là nhân viên y tế có chuyên môn trong việc điều chỉnh máy thở. Mặt khác, việc phát triển những công nghệ mới để có thể vận hành được máy thở y tế thường sẽ rất tốn kém và bị giới hạn về mặt y học. Vì vậy, việc dự đoán áp lực thở sẽ giúp các bác sĩ mô phỏng lại quá trình điều chỉnh các thông số. Qua đó giúp phá vỡ đi những rào cản hiện tại.

Từ trước tới nay đã có nhiều kỹ thuật và thuật toán trong học máy được đề xuất để giải nhiều bài toán nhằm tiết kiệm nguồn nhân lực là các chuyên gia trong ngành y tế. Một trong số đó phải kể đến các thuật toán thuộc lớp Gradient Tree Boosting [5] - một nhánh của họ thuật toán cây quyết định tổng hợp (ensemble decision tree) sử dụng phương pháp Boosting để gia tăng độ chính xác cho dữ liệu đầu ra. Phương pháp Boosting đã được kiểm chứng rộng rãi với những kết quả state-of-the-art trên nhiều benchmark khác nhau [11]. Ngoài ra, các thuật toán Tree-boosting còn được các kỹ sư phần mềm áp dụng trong việc thiết kế ra các pipeline thích hợp tạo nên những trải nghiệm tích cực hơn cho người dùng web [7].



Hình 1. Ví dụ về cây quyết định tổng hợp
với dữ liệu đầu ra là tổng của các dự đoán từ các cây quyết định

Trong bài báo cáo này, chúng tôi sẽ giới thiệu XGBoost [1],

[1]Trường Đại học Công nghệ Thông tin, ĐHQG HCM. Bùi Trí Dũng <19521386@gm.uit.edu.vn>, Võ Khánh An <19520007@gm.uit.edu.vn>, Phạm Ngọc Tân <19520925@gm.uit.edu.vn>.

một thuật toán máy học có thể thích nghi (scalable machine learning system) nằm trong họ thuật toán Tree-boosting. XGBoost đã được chứng tỏ được tính ổn định và độ chính xác trong nhiều cuộc thi về học máy và khai phá dữ liệu. Ngoài ra, XGboost còn là một công cụ được đánh giá rất linh hoạt và có thể giải quyết hầu hết các bài toán liên quan đến hồi quy (regression), phân loại (classification) và xếp hạng (ranking). Cụ thể hơn, khi vừa mới được giới thiệu vào năm 2014, XGBoost đã đem lại chiến thắng cho 17 đội trên tổng số 29 cuộc thi trên Kaggle chỉ trong năm 2015. Trong số 17 đội chiến thắng sử dụng XGBoost, chỉ có 8 đội sử dụng mô hình XGBoost đơn thuần, 9 đội còn lại chọn cách kết hợp XGBoost với các kỹ thuật học máy khác.

Trong bài nghiên cứu này, chúng tôi sẽ tiến hành giới thiệu về các công trình liên quan đến phạm vi của XGBoost ở phần 2. Sau đó, chúng tôi sẽ đưa ra nền tảng của XGBoost ở phần 3 và bàn luận chi tiết về thuật toán XGBoost cũng như các ưu, nhược điểm của nó ở phần 4. Hai phần tiếp theo sẽ được tập trung để nói về cách chúng tôi thiết kế thực nghiệm thông qua giới thiệu và phân tích tập dữ liệu được sử dụng (phần 5) cũng như chi tiết về độ đo, chi tiết triển khai, kết quả đạt được và triển khai ứng dụng (phần 6). Sau cùng, chúng tôi sẽ bàn luận mở rộng về XGBoost và đưa ra những bàn luận và kết luận cuối cùng về XGBoost ở phần 7 và 8.

2. Các công trình liên quan

Các thuật toán gradient boosting đang dần trở nên phổ biến đối với dữ liệu dạng bảng. Trong những năm vừa qua, chúng ta đã chứng kiến rất nhiều thuật toán boosting ra đời, mỗi thuật toán lại có những đặc điểm riêng biệt. Cụ thể, các thuật toán khác nhau ở việc triển khai cũng như tính tương thích và hạn chế về mặt kỹ thuật của chúng. XGBoost là thuật toán Gradient Boosting Machine (GBM) đầu tiên cải thiện đáng kể thời gian huấn luyện. Tiếp theo là LightGBM [9] và CatBoost [4], mỗi thuật toán có các kỹ thuật riêng, chủ yếu liên quan đến cơ chế phân tách (spliting mechanism). Các bản cập nhật của những thuật toán này liên tục được ra mắt với nhiều tính năng và khiến chúng ngày càng mạnh mẽ hơn.

3. Nền tảng

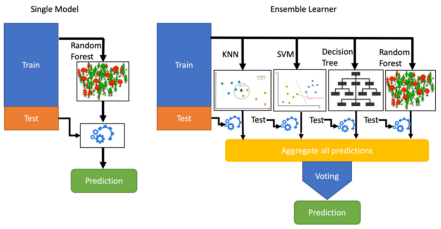
3.1. Boosting

Boosting được hiểu đơn giản là tăng hiệu quả của mô hình. Trong học máy, boosting là một kỹ thuật học tổng hợp (ensemble learning) để chuyển đổi một giả thuyết yếu thành mạnh để làm tăng độ chính xác của mô hình.

Những quy tắc yếu được tạo ra ở mỗi vòng lặp bởi các thuật toán học cơ sở (base learning algorithms), thứ có thể được chia làm 2 loại: mô hình dựa trên cây (tree-based learner) và mô hình dựa trên tuyến tính (linear-based learner). Nói chung, cây quyết định là mô hình cơ sở (base learner) mặc định của boosting.

3.2. Học tổng hợp

Học tổng hợp (ensemble learning) [15] là một quá trình mà trong đó các quyết định từ nhiều mô hình học máy được kết hợp để giảm lỗi và cải thiện việc dự đoán khi so sánh một mô hình học máy đơn. Sau đó kỹ thuật biểu quyết tối đa (maximum voting) được sử dụng trên các cây quyết định tổng hợp để suy ra dự đoán cuối cùng.



Hình 2. Mô hình dự đoán đơn và mô hình tổng hợp [6]

Phương pháp học tổng hợp có thể thực hiện theo hai cách: Bagging (tổng hợp song song) hoặc Boosting (tổng hợp tuần tự). Trong bài này, chúng tôi chỉ tập trung vào Boosting do nó là nền tảng của XGBoost.

3.3. Cách hoạt động của thuật toán Boosting

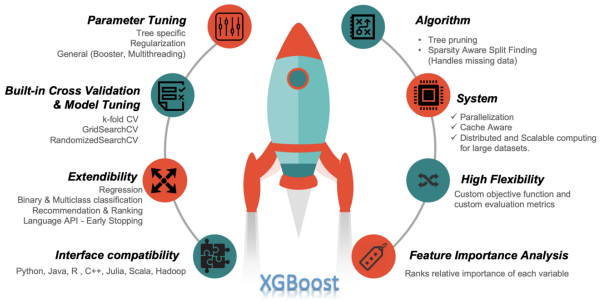
Thuật toán Boosting mô hình yếu (weak learners) mới và kết hợp tuần tự các dự đoán của chúng để cải thiện hiệu suất tổng thể của mô hình. Đối với bất kỳ dự đoán sai nào, các mẫu phân loại sai sẽ được gán trọng số lớn hơn và các mẫu phân loại đúng sẽ được gán trọng số nhỏ hơn. Các mô hình học yếu (weak learner models) hoạt động tốt hơn có trọng số cao hơn trong mô hình tổng hợp cuối cùng. Boosting không bao giờ thay đổi dự đoán trước đó và chỉ sửa chữa những dự đoán tiếp theo bằng cách học hỏi những sai lầm. Vì Boosting sử dụng chiến lược tham lam, chúng ta nên đặt tiêu chí dừng như hiệu suất mô hình (dừng sớm) để ngăn chặn tình trạng quá khớp (overfitting) khi huấn luyện.

4. Phương pháp

XGBoost, hay còn được biết đến với tên đầy đủ là *eXtreme Gradient Boosting*, là thuật toán máy học tổng hợp dựa trên cây quyết định (*decision-tree-based ensemble learning algorithm*). XGBoost được phát triển bởi Tianqi Chen và hiện là một phần của bộ sưu tập thư viện mã nguồn mở rộng và đang được phát triển bởi Cộng đồng Máy học Phân tán (Distributed Machine Learning Community). Dù cho

Hình 3. Cách thuật toán Boosting hoạt động [6]

các mô hình mạng neural nhân tạo [12] (*artificial neural network*) có hiệu suất vượt trội hơn trên những tập dữ liệu phi cấu trúc như hình ảnh, âm thanh, những mô hình dựa trên cây quyết định thường sẽ chiếm ưu thế so với các mô hình mạng neural đối với dữ liệu có cấu trúc nhỏ hay các dữ liệu bảng biểu.

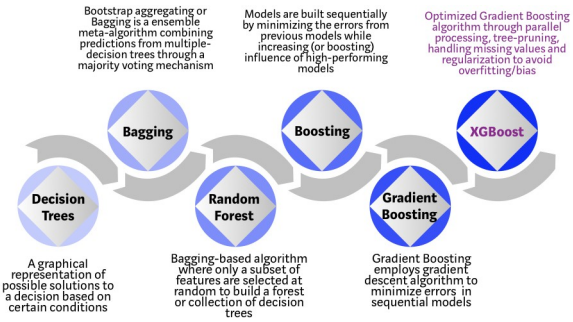


Hình 4. Tổng quan về XGBoost [6]

4.1. Sơ lược về các kĩ thuật phát triển XGBoost

Dù cho hình thái đơn giản nhất của XGBoost - mô hình cây quyết định (*decision tree*) rất dễ để ta có thể hình dung và hiểu được bản chất của thuật toán, việc phát triển nền tảng cho các mô hình tổng hợp dựa trên cây có phần khó khăn hơn thông qua việc sử dụng nhiều kĩ thuật khác nhau để xấp xỉ được dữ liệu đầu ra với độ lỗi là nhỏ nhất. Một vài kĩ thuật và thuật toán nổi bật trong việc phát triển các mô hình cây quyết định có thể được kể đến như:

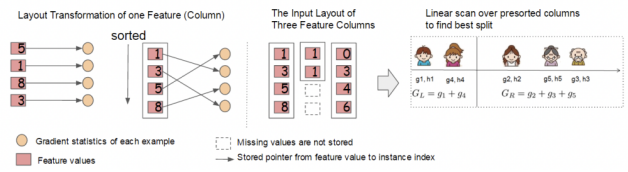
- **Bagging**: là hướng tiếp cận sử dụng cơ chế bình chọn (voting) giữa tất cả các đặc trưng để có thể cho ra được dự đoán đầu ra cuối cùng.
- **Random Forest**: là thuật toán sử dụng cơ chế voting của bagging, tuy nhiên, Random Forest [8] chỉ sử dụng voting trên một số lượng đặc trưng nhất định được chọn ngẫu nhiên để đưa ra lựa chọn tốt nhất cho đầu ra của mình.
- **Boosting**: là hướng tiếp cận sử dụng cơ chế phản hồi (feedback) đến từ những lựa chọn đầu ra để gia tăng độ chính xác đầu ra của mô hình thông qua quá trình đánh giá tích cực (dynamic evaluation process).
- **Gradient Boosting**: là thuật toán boosting sử dụng gradient descent để tối thiểu hóa độ lỗi đầu ra nhưng không làm phức tạp hóa mô hình cây quyết định. Sự khác biệt giữa boosting và gradient boosting chính là cách thức cả hai thuật toán này cải thiện mô hình dựa trên những dự đoán lỗi.



Hình 5. Sự tiến hoá của XGBoost bắt đầu từ mô hình cây quyết định [6]

4.2. Ưu điểm của XGBoost

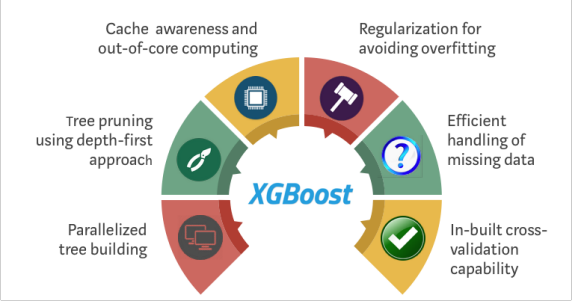
Ưu điểm lớn nhất của XGBoost đó chính là việc XGBoost được thiết kế để giải quyết việc khuyết dữ liệu trong nội tại. Việc xử lí dữ liệu bị khuyết được thực hiện thông qua việc mô hình phát hiện và dự đoán xu hướng của dữ liệu bị thiếu trong tập dữ liệu đầu vào để có thể đưa ra được giá trị phù hợp cho phần khuyết đó. Cụ thể, XGBoost sẽ thử sai nhiều hướng tiếp cận khác nhau mỗi khi nó gặp một điểm dữ liệu bị thiếu ở mỗi node trên cây. Do đó, nó có khả năng giải quyết bài toán hợp lý hơn khi xử lý những dữ liệu không đầy đủ ở các giai đoạn sau (ví dụ trong giai đoạn kiểm tra hoặc giai đoạn triển khai phần mềm).



Hình 6. Cấu trúc hình khối minh họa việc học song song (parallel learning) của XGBoost [1]

Ngoài ra, XGBoost còn có khả năng khai thác khả năng xử lý song song của máy tính và do đó nó có hiệu năng tính toán cao hơn rất nhiều so với các thuật toán GBM khác hay các thuật toán khác cùng dựa trên cây quyết định [14]. Dù cho Boosting là một quá trình tuần tự, nó vẫn có khả năng tính toán song song. Mỗi cây quyết định trong XGBoost chỉ được tạo ra sau khi những cây cha (ancestor trees) của nó đã được tạo và tất cả các cây con có thể được xây dựng một cách đồng thời thông qua việc sử dụng toàn bộ nhân (core) của CPU. Điều này giúp cho XGBoost là một trong những thuật toán nhanh nhất trong họ các thuật toán dựa trên cây quyết định.

Một ưu điểm khác giúp XGBoost nổi trội hơn so với các thuật toán khác chính là nó cho phép người dùng có thể tự do tùy chỉnh những mục tiêu tối ưu và những độ đo trong việc đánh giá và xếp hạng những lời giải cho bài toán.



Hình 7. Cách XGBoost tối ưu hoá thuật toán GBM tiêu chuẩn [6]

4.3. Khuyết điểm của XGBoost

Dù XGBoost có nhiều lợi thế hơn so với các thuật toán khác nhưng nó vẫn còn tồn tại nhiều thiếu sót và bất cập trong việc triển khai cho nhiều loại bài toán cũng như nhiều loại dữ liệu khác nhau. Như đã đề cập, các kỹ thuật sử dụng mô hình mạng neural nhân tạo sẽ có độ chính xác ổn định và vượt trội hơn so với những thuật toán học máy khác như SVM [2], AdaBoost [13], Logistic Regression [3] hay thậm chí là XGBoost. Cho đến nay, chỉ có những phương pháp học sâu mới có khả năng huấn luyện trên một lượng dữ liệu huấn luyện khổng lồ mà không bị bão hòa về mặt hiệu suất. Vì vậy, khi số lượng các mẫu huấn luyện trở nên rất lớn (chẳng hạn như 100.000 hay 1.000.000 mẫu) thì việc sử dụng các kỹ thuật học sâu sẽ mang lại hiệu quả cao hơn và tối ưu hơn về mặt thời gian.

Ngoài ra, dù XGBoost có độ chính xác cao hơn đối với đại đa số vấn đề và có các đặc tính tính toán tốt, nó không mang lại sự khác biệt đáng kể so với nhiều phương pháp truyền thống khác như SVM hoặc Random Forest, hay thậm chí có nhiều lúc hiệu suất của XGBoost trở nên cực kỳ tệ khi so sánh các thuật toán khác đơn giản hơn trong cùng một bài toán. Ta chỉ nên sử dụng mô hình XGBoost khi sở hữu một lượng đủ lớn (nhưng không quá lớn) các mẫu dữ liệu. Lý do là bởi vì mô hình này sẽ cố gắng tập trung vào mẫu dữ liệu mà người huấn luyện sẽ truyền vào, khi dữ liệu càng phong phú và đa dạng cũng như phạm vi các trường hợp hay đặc trưng lớn thì mô hình này có thể phát huy được tốt tác dụng của nó.

4.4. Các siêu tham số quan trọng khi sử dụng XGBoost

Việc tìm hiểu về các tham số của một mô hình là rất quan trọng bởi vì chúng giúp người sử dụng (hay người lập trình) có thể dễ dàng chỉnh sửa cũng như tiến hành tinh chỉnh (tuning) các tham số để làm cho mô hình tốt hơn. Dưới đây, chúng tôi sẽ mô tả khái quát về một vài siêu tham số quan trọng để tối ưu hóa mô hình XGBoost:

Đầu tiên ta phải nhắc đến tham số $n\_estimators$. Khi huấn luyện các mô hình học sâu, chúng ta có một siêu tham số là số lượng epochs. Mỗi epoch là một lần chúng ta cho dữ liệu chạy qua mô hình. Tham số $n\_estimators$ của XGBoost cũng giống như số lượng epochs trong các mô hình học sâu. Ở mỗi epoch, thuật toán sẽ tiến hành sửa lỗi mà mô hình ở epoch trước mắc phải (hay còn được biết đến dưới dạng quy trình boosting).

Bên cạnh $n\_estimators$, chúng ta có thể cải thiện hiệu suất của XGBoost thông qua tham số max_depth. Tham số này thường được sử dụng để xác định độ sâu của các cây quyết định trong mô hình học tổng hợp: độ sâu càng lớn thì sẽ giúp cho các dự đoán (hay các quyết định) của cây tốt hơn thông qua việc phức tạp hóa mô hình hiện có. Tuy nhiên, nếu độ sâu quá lớn sẽ khiến cho mô hình chúng ta bị quá khớp.

Trong trường hợp dữ liệu chúng ta bị mất cân bằng (imbalanced data), XGBoost cung cấp phương thức để điều chỉnh độ cân bằng giữa các trọng số dương và các trọng số âm thông qua tham số siêu $scale\_pos\_weight$: nếu thuật toán chúng ta gặp trường hợp hội tụ sớm khi dữ liệu quá mất cân bằng, ta có thể cân nhắc điều chỉnh tham số này với một giá trị lớn hơn 0 hoặc có thể gán tham số này với giá trị là tỉ lệ giữa tổng các mẫu âm với tổng các mẫu dương ($\frac{\sum \text{negative instances}}{\sum \text{positive instances}}$)

Ngoài ra, XGBoost còn cho phép lập trình viên điều chỉnh thuật toán theo ý muốn thông qua việc cung cấp các phương thức như: phương pháp boosting sử dụng cho thuật toán (ví dụ như *gbtree* hay *dart* để xây dựng XGBoost dưới dạng các cây quyết định hay *gblinear* để xây dựng XGBoost dưới dạng mô hình tuyến tính) hay điều chỉnh thuật toán xây dựng cây (*auto* - tự tìm thuật toán xây dựng cây nhanh nhất thông qua hàm heuristic, *exact* - sử dụng giải thuật tham lam, *approx*, *hist* - sử dụng biểu đồ tần suất (histogram) để tăng

tốc việc xấp xỉ cây bằng giải thuật tham lam, hay thậm chí là *gpu_hist* - sử dụng GPU cho phương thức hist đã đề cập) thông qua việc lựa chọn tham số $booster$ và $tree\_method$ thích hợp.

Như bao mô hình học máy truyền thống khác, XGBoost cung cấp cho chúng ta phương thức để điều chỉnh tốc độ học trong quá trình huấn luyện thuật toán. Tham số để điều chỉnh tốc độ học cho thuật toán được xác định là tham số eta hay $learning\_rate$: tốc độ học có tác dụng thay đổi tốc độ mô hình đi tìm cực trị khi ta tiến hành thực thi thuật toán, eta càng thấp (trong khoảng $(0; 1)$) thì tốc độ mô hình đi tìm cực trị sẽ càng chậm, và eta càng cao (trong khoảng $(1; +\infty)$) thì tốc độ mô hình đi tìm cực trị sẽ càng nhanh. Ta cần phải chọn giá trị eta thích hợp mô hình vì nếu eta quá cao thì mô hình khó có thể tìm được điểm cực trị gần với giá trị tối ưu. Tuy nhiên, nếu eta quá thấp thì mô hình sẽ mất rất nhiều thời gian để có thể tìm đến với cực trị của bài toán.

Cuối cùng, XGBoost cũng cung cấp cho chúng ta cách xử lý khi thuật toán bị quá khớp trong giai đoạn kiểm thử. Khi mô hình XGBoost bị quá khớp, chúng ta có thể tiến hành chính quy hoá (regularization) mô hình để ràng buộc các hệ số với các chuẩn (norm) L1 và L2, tương tự với việc sử dụng mô hình Lasso và Ridge trong Hồi quy tuyến tính. Để thay đổi các hệ số chính quy hóa, ta sẽ tinh chỉnh tham số $alpha$ (với chuẩn L1) và tham số $lambda$ (với chuẩn L2). Việc sử dụng chuẩn L1 và L2 cho XGBoost sẽ mang lại ý nghĩa tương tự khi ta tiến hành giảm bậc của bài toán giúp cho bài toán trở nên đơn giản hơn.

Ngoài việc sử dụng L1 và L2, ta còn có thể giúp cho mô hình tránh được tình trạng quá khớp thông qua việc sử dụng các tham số $subsample$. Ví dụ với $subsample = 0.5$, mô hình sẽ lựa chọn random 50% mẫu trong dữ liệu ban đầu ra để huấn luyện. Ngoài ra, XGBoost còn các siêu tham số như $colsample\_bytree$, $colsample\_bylevel$ hay $colsample\_bynode$ để giúp cho mô hình tránh trường hợp quá khớp. Các siêu tham số trong lớp $subsample$ và $colsample$ thường sẽ nằm trong nửa đoạn $(0; 1]$. Nếu ta tinh chỉnh các siêu tham số này ở mức vừa phải (dưới 0.5), mô hình XGBoost sẽ sử dụng ít đặc trưng và do đó tránh được tình trạng quá khớp. Tuy nhiên, cần phải lưu ý rằng nếu như ta tinh chỉnh các tham số này ở mức quá cao (trên 0.5), mô hình chúng ta sẽ dễ rơi vào tình trạng chưa khớp (underfitting).

Có thể thấy, XGBoost cung cấp cho ta rất nhiều sự lựa chọn để thay đổi các siêu tham số nhằm tùy biến mô hình. Tuy nhiên, chúng ta phải tinh chỉnh các siêu tham số ở mức hợp lý để đạt được độ chính xác hay mức độ điều hòa (harmonic) tốt.

5. Bộ dữ liệu

5.1. Mô tả dữ liệu

Trong bài này, chúng tôi sử dụng bộ dữ liệu Ventilator Pressure Prediction của Google Brain[1]. Bộ dữ liệu này cung cấp thông tin về các chuỗi thời gian thở với mục tiêu là học cách dự đoán áp lực đường thở trong mạch hô hấp dựa trên những mốc thời gian khác nhau.

Mỗi mốc thời gian đại diện cho một pha hô hấp kéo dài khoảng 3 giây. Bộ dữ liệu được sắp xếp sao cho mỗi hàng là một mốc thời gian của hơi thở và đưa ra hai tín hiệu điều khiển, kết quả là áp lực thở, cũng như các thuộc tính liên quan của phổi và máy thở, được mô tả chi tiết ở hình 8.

	id	breath_id	R	C	time_step	u_in	u_out	pressure
0	1	1	20	50	0.000000	0.083334	0	5.837492
1	2	1	20	50	0.033652	18.383041	0	5.907794
2	3	1	20	50	0.067514	22.509278	0	7.876254
3	4	1	20	50	0.101542	22.808822	0	11.742872
4	5	1	20	50	0.135756	25.355850	0	12.234987

Hình 8. Ví dụ về các thuộc tính trong 5 hàng đầu tiên của tập dữ liệu

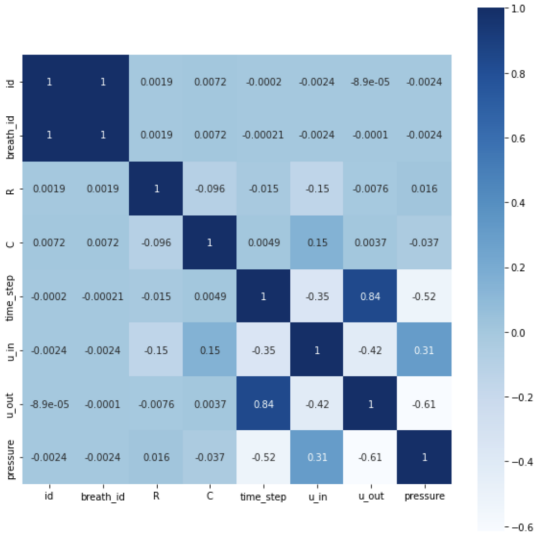
Trong đó:

- *id* – định danh mốc thời gian cho toàn bộ tập dữ liệu
- *breath_id* – định danh mốc thời gian giữa các pha hô hấp
- *R* – thuộc tính phổi cho biết mức độ tắc nghẽn đường dẫn khí (có đơn vị là $cmH_2O/l/s$, trong đó: l được kí hiệu cho lít và s được kí hiệu cho giây). Về mặt vật lý, đây là sự thay đổi áp suất trên mỗi lần thay đổi lưu lượng phổi (lưu lượng thở trên một đơn vị thời gian).
- *C* – thuộc tính phổi cho biết mức độ giãn nở của phổi (có đơn vị là mL/cmH_2O). Về mặt vật lý, đây là sự thay đổi thể tích trên mỗi lần thay đổi áp lực thở.
- *time_step* – mốc thời gian thực
- *u_in* – đầu vào điều khiển cho van điện tử hít vào (inspiratory solenoid valve), giá trị nằm trong phạm vi từ 0 đến 100.
- *u_out* – đầu vào điều khiển cho van điện từ thở ra (exploratory solenoid valve), mang giá trị 0 hoặc 1.
- *pressure* – áp lực thở trong hệ hô hấp, tính bằng cmH_2O.

[1] https://www.kaggle.com/c/ventilator-pressure-prediction

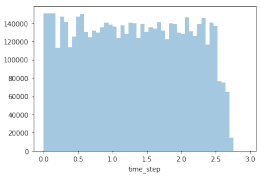
5.2. Phân tích dữ liệu

Để có cái nhìn chi tiết hơn về bộ dữ liệu, chúng tôi đã trực quan hóa mức độ tương quan giữa các thuộc tính thông qua biểu đồ nhiệt (heatmap) ở hình 9.

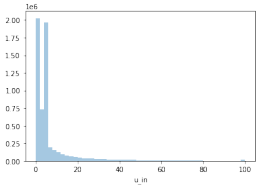


Hình 9. Mức độ tương quan giữa các thuộc tính trong bộ dữ liệu

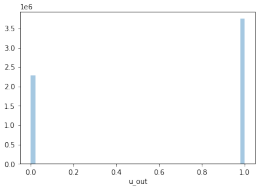
Ta có thể thấy được rằng: Ngoài hai tổ hợp đặc trưng id và breath_id, hai đặc trưng có mức độ tương quan lớn nhất chính là time_step và u_out. Điều nay là bởi ta có thể quan sát được tiến trình hô hấp của bệnh nhân đang sử dụng máy thở (đang hít vào hay thở ra).



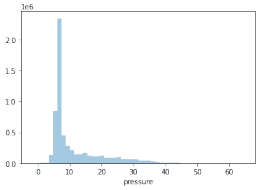
Hình 10. Tần số time_step



Hình 11. Tần số u_in



Hình 12. Tần số u_out



Hình 13. Tần số pressure

Ngoài ra, đặc trưng pressure cũng có một sự phụ thuộc khá cao vào đặc trưng u_in. Điều này có thể được lý giải thông qua nguyên lý của máy thở: khi ta thay đổi giá trị của u_in, nghĩa là ta đang thay đổi áp suất không khí có trong phế quản của bệnh nhân, tương ứng với giá trị của biến số pressure trong tập dữ liệu.

Bên cạnh pressure, thuộc tính C cũng có mức độ phụ thuộc đáng kể với u_in. Khi ta tăng hay giảm giá trị của van điện tử hít vào, mức độ giãn nỡ của phổi cũng sẽ thay đổi theo (biến đổi theo thể tích không khí truyền vào phổi). Những đặc trưng còn lại có mức độ tương quan thấp hoặc không đáng kể lẫn nhau.

Bên cạnh mức độ tương quan, ta còn có thể hiểu rõ hơn về cách phân bố dữ liệu thông qua một số đồ thị histogram như ở hình 10, 11, 12 và 13.

6. Thực nghiệm

6.1. Thang đo

Trong quá trình thực nghiệm, chúng tôi sử dụng thang đo là *sai số tuyệt đối trung bình* (Mean Absolute Error). Đây là một phương pháp đo lường sự khác biệt giữa hai biến liên tục. Giả sử rằng X và Y là hai biến liên tục thể hiện kết quả dự đoán của mô hình và kết quả thực tế. chúng ta có độ đo MAE được tính theo công thức sau:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{1}$$

Trong đó, n là số lượng điểm dữ liệu, x_i là giá trị thực của tập huấn luyện tại điểm thứ i, y_i là giá trị dự đoán tại điểm thứ i.

Độ đo này thường được sử dụng để đánh giá sự sai khác giữa mô hình dự đoán và tập kiểm tra trong các bài toán hồi quy. Chỉ số này càng nhỏ thì mô hình học máy càng chính xác.

6.2. Chi tiết triển khai

Chúng tôi sử dụng phương pháp Tìm kiếm theo lưới (Grid Search) [10] để lựa chọn các siêu tham số cho XGBoost. Tìm kiếm theo lưới là một kỹ thuật tinh chỉnh các siêu tham số để tìm ra bộ siêu tham số tối ưu nhất. Đây là một kỹ thuật tìm kiếm vét cạn được thực hiện trên một không gian tìm kiếm cụ thể được định nghĩa trước. Tìm kiếm theo lưới tuy đơn giản nhưng lại giúp chúng ta tiết kiệm thời gian, công sức và tài nguyên.

Nhằm đánh giá XGBoost một cách hiệu quả nhất, chúng tôi tiến hành so sánh XGBoost với một số mô hình như Linear Regression và LightGBM [9]. Bên cạnh đó, chúng tôi cũng sử dụng kỹ thuật K-Fold Cross Validation và Grid Search cho cả ba thuật toán XGBoost, Light GBM và Hồi quy tuyến tính (Linear Regression).

Sau khi chạy Grid Search chúng tôi quyết định lựa chọn các

siêu tham số của LightGBM và XGBoost như sau:

Siêu tham số	Giá trị
booster	'gbtree'
gamma	3.2
gpu_id	0
learning_rate	0.1
max_depth	8
n_estimators	5000
predictor	'gpu_predictor'
reg_alpha	15.9
reg_lambda	66.1
subsample	0.95
tree_method	'gpu_hist'

Bảng 1. Siêu tham số được sử dụng cho XGBoost

Siêu tham số	Giá trị
bagging_fraction	1.0
bagging_freq	0
boosting_type	'gbdt'
feature_fraction	0.90
feature_pre_filter	False
lambda_l1	0.0
lambda_l2	0.0
learning_rate	0.1
max_depth	8
metric	'l1'
min_child_samples	2
num_leaves	350
objective	'regression_l1'
random_state	42
verbose	-1

Bảng 2. Siêu tham số được sử dụng cho LightGBM

6.3. Kết quả

Thuật toán	MAE	Thời gian huấn luyện (s)
XGBoost	4.6240	148.32
LightGBM	4.7457	30.9537
Hồi quy tuyến tính	8.09	1.46

Bảng 3. Sai số tuyệt đối trung bình và thời gian huấn luyện của ba thuật toán: XGBoost, LightGBM và Linear Regression

Kết quả cuối cùng chúng tôi đạt được trên tập kiểm tra (Bảng 3) cho thấy tuy XGBoost có thời gian huấn luyện lâu nhất nhưng vẫn ở mức chấp nhận được. Đổi lại, XGBoost đạt được MAE tốt nhất so với hai thuật toán LgihtGBM và Hồi quy tuyến tính. Bên cạnh đó, LightGBM cũng cho thấy mình là một đối thủ đáng gờm với XGBoost khi có MAE thua kém không đáng kể và thời gian huấn luyện vượt trội. Ngoài ra, chúng tôi cũng khảo sát một mô hình đơn giản là Hồi quy tuyến tính (Linear Regrression) nhưng kết quả không tốt lắm dù thời gian huấn luyện chỉ hơn 1 giây.

6.4. Triển khai ứng dụng

Để chứng minh tính ứng dụng của nghiên cứu này, chúng tôi có phát triển một ứng dụng web bằng NodeJS cho phép người dùng nhập vào các đặc trưng như trong bộ dữ liệu và đầu ra là áp lực của máy thở lên bệnh nhân. Video demo được công khai tại: https://github.com/DTA-UIT/Ventilator_Pressure_Prediction

7. Bàn luận

7.1. Có nên sử dụng XGBoost mọi lúc, mọi nơi?

Câu trả lời dĩ nhiên là không. Khi giải quyết một bài toán bằng học máy, chúng ta phải kiểm tra tất cả các thuật toán có thể trên một tập dữ liệu được cho sẵn mới có thể khẳng định được thuật toán nào là tốt nhất. Bên cạnh đó, việc lựa chọn đúng thuật toán là chưa đủ. Chúng ta còn phải lựa chọn bộ siêu tham số phù hợp cho thuật toán đó và điều này là không hề dễ dàng với những thuật toán có nhiều siêu tham số cũng như có phạm vi tìm kiếm siêu tham số rộng. Ngoài ra vẫn còn nhiều yếu tố khác để lựa chọn thuật toán như độ phức tạp, khả năng giải thích và tính khả thi khi triển khai. Do vậy, khi đụng phải một bài toán, chúng ta vẫn phải suy nghĩ xem là nên lựa chọn thuật toán nào dựa vào những điều trên chứ không nên sử dụng cố định một thuật toán nào đó.

7.2. XGBoost liệu có thể giữ vững được vị thế của mình?

Trí tuệ nhân tạo là một lĩnh vực luôn luôn vận động và phát triển. Do đó, đã có không ít các giải pháp được đề xuất để thay thế cho XGBoost. Năm 2016, Microsoft đã cho ra mắt thuật toán LightGBM và nó đã nhanh chóng cho thấy một tiềm năng rất lớn. CatBoost của Yandex ra mắt năm 2017 cũng mang lại những kết quả rất ấn tượng. Dù các thuật toán này chưa thể đánh bại được XGBoost hoàn toàn nhưng để một thuật toán nào đó có thể có hiệu năng vượt trội hơn XGBoost thì vấn đề chỉ là thời gian mà thôi. Tuy vậy, cho đến khi kẻ thách thức mới xuất hiện thì XGBoost vẫn là kẻ thống trị thế giới học máy.

8. Kết luận

Trong nghiên cứu này, chúng tôi cũng đã làm rõ được những lý thuyết nền tảng của XGBoost cũng như phân tích một số khía cạnh của bộ dữ liệu. Ngoài ra, chúng tôi đã tiến hành so

sánh XGBoost với một số thuật toán khác như LightGBM và Hồi quy tuyến tính trên tập dữ liệu dự đoán áp lực của máy thở lên bệnh nhân. Qua những kết quả thu được, XGBoost đã thể hiện hiệu năng tốt nhất so với các thuật toán khác cũng như đạt được thời gian huấn luyện chỉ hơn 2 phút.

Lời cảm ơn

Chúng tôi chân thành gửi lời cảm ơn đến TS. Nguyễn Vinh Tiệp đã giảng dạy môn Lập trình Python cho Máy học cũng như đưa ra những lời góp ý giúp chúng tôi chỉnh sửa bài báo cáo này.

Tài liệu

[1] Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016.

[2] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[3] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[4] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[5] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[6] Rohan Harode. Xgboost: A deep dive into boosting, February 2020.

[7] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD'14, page 1–9, New York, NY, USA, 2014. Association for Computing Machinery.

[8] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[9] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[10] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[11] Ping Li. Robust logitboost and adaptive base class (abc) logitboost, 2012.

[12] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[13] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[14] Taniya. Machine learning algorithms: A comparison of different algorithms and when to use them, May 2018.

[15] Cha Zhang and Yunqian Ma. *Ensemble Machine Learning: Methods and Applications*. Springer Publishing Company, Incorporated, 2012.