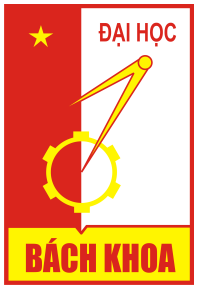
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

———————o0o———————



Ứng dụng kỹ thuật Boosting trong bài toán phân loại

ĐỒ ÁN I

Chuyên ngành: TOÁN TIN

Giảng viên hướng dẫn: **ThS. NGUYỄN TUẤN DŨNG** _____
Bộ môn: TOÁN TIN Chữ kí của GVHD
Sinh viên thực hiện: **LÊ ĐỨC TÀI**
Lớp: CTTN Toán Tin - K64

Hà Nội, .../2023

Nhận xét của giảng viên	Điểm

Hà Nội, ngày ... tháng ... năm 2023

Giảng viên hướng dẫn

ThS. Nguyễn Tuấn Dũng

Lời cảm ơn

Trong suốt quá trình học tập và tìm hiểu nội dung đồ án, tác giả luôn nhận được sự quan tâm, hướng dẫn giúp đỡ tận tận của các thầy, cô giáo thuộc viện Toán Ứng dụng và tin học cùng với sự góp ý từ bạn bè.

Đặc biệt tác giả gửi lời cảm ơn sâu sắc đến ThS.Nguyễn Tuấn Dũng thuộc Bộ môn Toán Tin, đã trực tiếp giúp đỡ, hướng dẫn em có thể hoàn thiện nội dung đồ án.

Em xin trân trọng cảm ơn!

Hà Nội, ngày ... tháng ... năm 2023

Tác giả đồ án

Lê Đức Tài

Lời mở đầu

Cấu trúc đồ án

Đồ án I này sẽ được trình bày gồm 4 chương chính như sau:

- **Chương 1- Giới thiệu chung:** Chương này giới thiệu bài toán phân loại, phương sai - độ chệch và khái quát về Ensemble Learning trong học máy.
- **Chương 2- Boosting:** Trong chương này trình bày cụ thể về 3 thuật toán của Boosting đó là: Adaptive Boosting, Gradient Boosting và Extreme Gradient Boosting và chạy thực nghiệm trên bộ dữ liệu thực tế.
- **Chương 3- Kết luận**

Dù đã rất cố gắng trong quá trình nghiên cứu và soạn thảo đồ án nhưng vẫn không tránh khỏi những hạn chế nhất định. Vì vậy, rất mong nhận được sự đóng góp, chỉ bảo và ý kiến của các quý thầy cô. .

Quy tắc viết đồ án và ký hiệu

Trong báo cáo này, tác giả sử dụng một số tên viết tắt cho các thuật ngữ như sau:

DT	Decision Tree
LR	Logistic Regression
kNN	k-Nearest Neighbor
AdaBoost	Adaptive Boosting
GBM	Gradient Boosting
XGB	eXtreme Gradient Boosting

Bảng 1: Bảng thuật ngữ viết tắt

Mục lục

Danh sách bảng

Danh sách hình vẽ

Chương 1

Giới thiệu chung

1.1 Bài toán phân loại

1.1.1 Giới thiệu bài toán phân loại

Bài toán phân loại thuộc lớp bài toán học có giám sát trong học máy trong đó mô hình cố gắng dự đoán nhãn chính xác của dữ liệu đầu vào. Trong phân loại, mô hình được đào tạo bằng cách sử dụng dữ liệu đào tạo và sau đó được đánh giá trên dữ liệu thử nghiệm trước khi sử dụng để dự đoán trên dữ liệu mới. Ví dụ: Gmail xác định xem một email có phải là spam hay không; các hãng tín dụng xác định xem một khách hàng có khả năng thanh toán nợ hay không.

Bài toán phân loại được chia thành các loại sau:

- **Phân loại nhị phân** (Binary Classification): là bài toán phân loại dữ liệu cơ bản nhưng có nhiều ứng dụng thực tiễn. phân loại nhị phân sẽ phân chia dữ liệu thanh hai lớp độc lập. Một số bài toán thực tiễn của phân loại nhị phân như phân loại dữ liệu liên quan đến bệnh tật: nhiễm bệnh và không nhiễm bệnh, phân loại tin nhắn: tin nhắn rác và tin nhắn thông thường, phân tích đánh giá phản hồi của khác hàng: tích cực và tiêu cực, v.v. Thông thường, bài toán phân loại dữ liệu nhị phân dựa trên mô hình phân phối xác suất Bernoulli.

  Một số thuật toán thường được sử dụng cho bài toán phân loại dữ liệu nhị phân: Logistic Regression , Support Vector Machines , K-Nearest Neighbors , Naïve Bayes và Decision Trees.

- **Phân loại đa lớp** (Multi-Class Classification): dữ liệu được phân chia thành nhiều nhóm khác nhau dựa vào các nhãn được chỉ định. Số lượng các nhãn có thể rất lớn và phụ thuộc vào từng bài toán phân loại. Một ví dụ thực tiễn của bài toán phân loại đa lớp là bài toán dịch thuật văn bản. Mỗi một từ trong câu có thể được dự đoán liên quan đến bài toán phân loại đa lớp. Trong đó, số lượng các lớp tương ứng với số lượng từ vựng trong từ điển.

  Một số thuật toán phổ biến thường được sử dụng cho bài toán phân loại đa lớp như kNN, Decision Trees, Naïve Bayes, Random Forest , và Gradient Boosting . Một số phương pháp sử dụng cho phân

lớp nhị phân có thể mở rộng để áp dụng cho bài toán phân loại đa lớp.

- **Phân loại đa nhãn** (Multi-label Classification): cũng liên quan đến nhiệm vụ phân loại dựa trên nhiều nhãn khác nhau, nhưng một hoặc nhiều nhãn khác nhau có thể được sử dụng để dự đoán cho mỗi một mẫu. Điều này hoàn toàn khác với phân loại đa lớp, một nhãn chỉ dùng để tiên đoán một mẫu. Một ví dụ về phân loại đa nhãn là phân loại ảnh. Khi đó, trong một bức ảnh cho trước có thể chứa nhiều đối tượng khác nhau như người, xe, hoa, v.v. Mô hình phân loại đa nhãn có thể tiên đoán sự hiện diện của nhiều đối tượng đó.Các thuật toán sử dụng cho phân loại nhị phân và phân loại đa lớp không thể áp dụng trực tiếp cho bài toán phân loại đa nhãn. Các phương pháp phổ biến thường được dùng cho phân loại đa nhãn là Multi-label Decision Trees, Multi-label Gradient Boosting và Multi-label Random Forests.

- **Phân loại dữ liệu không cân bằng** (Imbalanced Classification): liên quan đến các nhiệm vụ phân loại mà ở đó số lượng mẫu của mỗi lớp được phân bố không đồng đều. Về cơ bản, nó cũng là một dạng của phân loại nhị phân. Một số ví dụ thực tiễn của phân loại không cân bằng là bài toán phát hiện lừa đảo, phát hiện tin giả, chẩn đoán bệnh trong y khoa, v.v. Bởi phân loại không cân bằng có thể xem như là một dạng của phân loại nhị phân, nên ta có thể sử dụng các kĩ thuật của phân loại nhị phân cho dạng phân loại này. Tuy nhiên, các thuật toán cần phải tập trung nhiều vào lớp không chính yếu. Một số thuật toán phổ biến thường dùng cho bài toán phân loại dữ liệu không cân bằng là Cost-sensitive Decision Trees, Cost-sensitive Logistic Regression và Cost-sensitive Support Vector Machines.

1.1.2 Các độ đo để đánh giá mô hình phân loại trong học máy

Xét ma trận nhầm lẫn (confusion matrix) đối với phân loại nhị phân được xác định ở bảng sau đây:

Prediction / **Reality**	**Yes**	**No**
Yes	True Positives (TP)	False Negatives (FN)
No	False Positives (FP)	True Negatives (TN)

Bảng: Confusion Matrix

- **Độ chính xác (Accuracy)**: Phần trăm dữ liệu được phân loại đúng, được tính theo công thức:

$$\textbf{Accuracy} = \frac{\textbf{TP} + \textbf{TN}}{\textbf{TP} + \textbf{TN} + \textbf{FP} + \textbf{FN}}$$

- **Độ nhạy (Recall/Sensitivity)**: Đo lường tỷ lệ dự báo chính xác các trường hợp Positive trên toàn bộ các mẫu thuộc nhóm Positive, được tính theo công thức:

$$\textbf{Recall} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FN}}$$

- **Độ chính xác phép đo (Precision)**: Cho biết thực sự có bao nhiêu dự đoán Positive là thật sự True và được tính theo công thức:

$$\textbf{Precision} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FP}}$$

- **Điểm F1 (F1 score)**: Là trung bình điều hòa của giá trị độ nhạy và độ chính xác phép đo. Điểm F1 được tính theo công thức:

$$\textbf{F1 scores} = 2.\frac{\textbf{Precision} \times \textbf{Recall}}{\textbf{Precision} + \textbf{Recall}}$$

Với bài toán phân lớp đa lớp, ta lần lượt xem một lớp là positive, các lớp còn lại là negative. Khi đó độ nhạy, độ chính xác phép đo và điểm F1 cho tất cả các lớp được tính theo theo hai kiểu:

- **Macro-averaging**: Tính toán độ chính xác phép đo và độ nhạy dựa trên trung bình của độ chính xác phép đo và độ nhạy của từng lớp.

$$\textbf{Macro-averaging Precision} = \frac{\textbf{Precision1} + \textbf{Precision2} + ... + \textbf{Precisionk}}{\textbf{k}}$$
$$\textbf{Macro-averaging Recall} = \frac{\textbf{Recall1} + \textbf{Recall2} + ... + \textbf{Recallk}}{\textbf{k}}$$

  Trong đó k là số lớp .

- **Weighted-averaging**: Tính toán độ chính xác phép đo và độ nhạy dựa trên trung bình của độ chính xác phép đo và độ nhạy của từng lớp, với trọng số được tính dựa trên số lượng mẫu thuộc mỗi lớp.

$$\textbf{Weighted-averaging Precision} = \frac{n_1 \times \textbf{Precision1} + n_2 \times \textbf{Precision2} + ... + n_k \times \textbf{Precisionk}}{n_1 + n_2 + ... + n_k}$$
$$\textbf{Weighted-averaging Recall} = \frac{n_1 \times \textbf{Recall1} + n_2 \times \textbf{Recall2} + ... + n_k \times \textbf{Recallk}}{n_1 + n_2 + ... + n_k}$$

  Trong đó, k là số lượng lớp, và n_i là số lượng mẫu thuộc lớp thứ i.

1.2 Mối quan hệ giữa độ chệch và phương sai

1.2.1 Lỗi trong học máy

Trong học máy, lỗi được sử dụng để xem mô hình có thể dự đoán chính xác như thế nào trên dữ liệu mà nó sử dụng để học, cũng như dữ liệu mới chưa nhìn thấy. Dựa trên lỗi, có thể chọn mô hình học máy hoạt động tốt nhất cho một tập dữ liệu cụ thể.

Có 2 loại lỗi chính trong bất kỳ mô hình học máy nào. Đó là lỗi có thể giảm được(Reducible Errors) và lỗi không thể giảm bớt (Irreducible Errors).

- Lỗi không thể giảm bớt là độc lập với mô hình hay thuật toán được sử dụng và không thể giảm bằng bất kỳ mô hình hay thuật toán nào. Lỗi không thể giảm bớt là cận dưới (lower bound) của lỗi dự đoán. Nghĩa là, ngay cả khi có lượng dữ liệu vô hạn và môt mô hình lý tưởng, thì lỗi không thể giảm bớt là lỗi sai nhỏ nhất mà một mô hình có thể đạt được.

- Lỗi có thể giảm được là những lỗi có thể giảm thêm để cải thiện mô hình. Nguyên nhân của lỗi này là do hàm đầu ra của mô hình không khớp hàm đầu ra mong muốn và có thể được tối ưu. Những lỗi như vậy có thể được phân loại thành độ chệch (bias) và phương sai (variance).

1.2.2 Độ chệch

Độ chệch là một lỗi sai hệ thống (systematic error) xảy ra do các giả định không chính xác (incorrect assumptions) của mô hình về dữ liệu và mối quan hệ giữa các biến đầu vào (features) và biến đầu ra (target variables/labels) trong quá trình học.

Về mặt kỹ thuật, độ chệch được xác định như là sự khác biệt giữa trung bình các dự đoán của mô hình và giá trị thực sự của nó (ground truth).
Một mô hình độ chệch cao (high bias) nghĩa dự đoán mà mô hình đưa ra so với giá trị thực sự của nó sẽ khác xa nhau, điều này xảy ra khi mô hình quá đơn giản, quá cơ bản (basic). Một mô hình quá đơn giản, quá cơ bản sẽ có quá ít tham số, khi đó nó chỉ có thể biểu diễn được hạn chế 1 tập nhỏ các mẫu hay mối quan hệ có trong dữ liệu, không phản ánh được sự đa dạng đầy đủ (full diversity) của dữ liệu.
Một mô hình như vậy sẽ không thể nắm bắt được các đặc trưng quan trọng hay mối quan hệ phức tạp có cả trong dữ liệu đào tạo (training data) và dữ liệu mà mô hình chưa được nhìn thấy (test data). Trong trường hợp này, mô hình đang bị underfitting.

1.2.3 Phương sai

Phương sai là khái niệm đặc trưng cho sự phân tán của các dự đoán của mô hình hay độ nhạy của mô hình đối với các dao động trong dữ liệu. Phương sai cho thấy dự đoán của mô hình sẽ thay đổi như thế nào trên các bộ dữ liệu khác nhau. Một mô hình với phương sai cao (high variance) sẽ đưa ra các dự đoán rất khác nhau cho các bộ dữ liệu khác nhau (thậm chí chỉ là một sự thay đổi nhỏ trong dữ liệu). High variance xảy ra khi mô hình quá phức tạp với nhiều tham số, khi đó nó có thể "fit" dữ liệu đào tạo vô cùng tốt. Tuy nhiên, thay vì khám phá ra các mẫu hay mối quan hệ thiết yếu (essential patterns or relationships), nó sẽ nắm bắt tất cả đặc trưng của dữ liệu, bao gồm cả những đặc trưng tầm thường, không quan trọng hay nhiễu (noise).
Điều này cũng có nghĩa là nó đang "ghi nhớ" các ví dụ cụ thể của dữ liệu (specific examples) thay vì khái quát hóa nó. Và khi gặp dữ liệu mới, các ví dụ mới, ví dụ mà có thể không có các đặc trưng giống như các đăc trưng trong các ví dụ mà nó đã "ghi nhớ", mô hình sẽ thất bại để đưa ra một dự đoán tốt. Trong trường hợp này, mô hình đang bị overfitting.

1.2.4 Sự đánh đổi giữa độ lệch và phương sai

Do lỗi không thể giảm bớt là lỗi sai luôn có trong bất kỳ mô hình nào và không thể giảm được, nên để giảm thiểu lỗi dự đoán, ta chỉ có thể giảm phương sai hoặc độ chệch. Nhưng ta không thể giảm đồng thời cả độ chệch và phương sai, vì có một sự đánh đổi (được cái này-mất cái kia) giữa chúng: khi giảm độ chệch thì phương sai sẽ tăng và ngược lại.

Khi mô hình có độ chệch cao, nó quá đơn giản để có thể nắm bắt được mối quan hệ phức tạp(complexity relationships) trong dữ liệu. Để giảm độ chệch, ta cần tăng độ phức tạp hay số lượng tham số của mô hình. Nhưng việc tăng độ phức tạp lại dẫn đến tăng phương sai của mô hình, khiến nó nhạy cảm hơn với các dao động của dữ liệu và dễ bị overfitting.
Tương tự, khi mô hình có phương sai cao và có nguy cơ bị overftting, nó sẽ nắm bắt các nhiễu và thuộc tính không quan trọng (unimportant features) của dữ liệu. Để giảm phương sai, ta cần giảm độ phức tạp hay số lượng tham số. Nhưng việc giảm độ phức tạp này lại dẫn đến tăng độ chệch của mô hình, khiến nó giảm khả năng để "fit" dữ liệu huấn luyên tốt và nguy cơ bị underfititng.

Vì vậy, với bất kỳ mô hình nào, ta cần tìm sự cân bằng (balance) giữa độ chệch và phương sai. Điều này đảm bảo rằng mô hình có thể nắm bắt được các mẫu thiết yếu trong dữ liệu trong khi bỏ qua nhiễu và thuộc tính không quan trong.

Trên thực tế, sự đánh đổi này có thể được xử lý bằng bằng các kỹ thuật như: cross-valiation, regularization và ensemble learning. Cross-validation giúp tìm ra mô hình có khả năng khái quát hóa vào new unseen data tốt bằng cách huấn luyện và đánh giá mô hình nhiều lần. Regularization giúp ngăn chặn overfitting bằng việc thêm 1 hằng số phạt vào tham số của mô hình để hạn chế độ phức tạp của mô hình. Và ensemble learning kết hợp dự đoán của nhiều mô hình để giảm phương sai và tăng hiệu suất tổng thể (overall performance).

1.3 Giới thiệu Ensemble Learning

Ensemble Learning là một kỹ thuật học máy để kết hợp các dự báo của nhiều mô hình để tạo ra một dự báo chính xác và mạnh mẽ hơn. Mục đích chính của Ensemble Learning là để giải quyết các giới hạn của các mô hình đơn, như cao bias hoặc cao variance, bằng cách tổng hợp kết quả của nhiều mô hình. Điều này giúp cải thiện hiệu suất và ổn định của các dự báo.

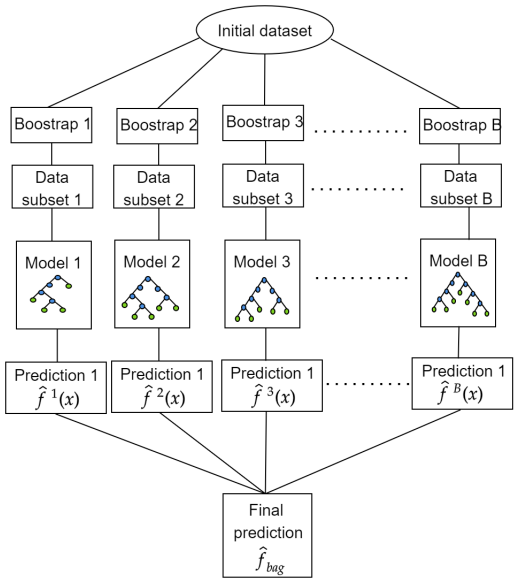
Trong ensemble learning gồm 3 kỹ thuật chính là Bagging, Boosting và Stacking. Bagging là việc sử dụng nhiều mô hình độc lập và tạo ra nhiều dự báo khác nhau để tạo ra một dự báo cuối cùng. Boosting sử dụng mô hình con để cải thiện hiệu suất của mô hình tổng quát bằng cách tập trung vào các vấn đề mà các mô hình trước đó chưa giải quyết được. Stacking là một biến thể của mô hình máy học kết hợp - ensemble learning còn được gọi là phương pháp meta-learning, bao gồm một hệ thống phân cấp các bộ phân loại khác nhau. xây dựng một bộ phân loại cấp độ meta có thể dự đoán nhãn đích của tập dữ liệu bằng cách kết hợp kết quả các dự đoán từ các bộ phân loại riêng biệt.

1.3.1 Bagging

Bagging (Bootstrap Aggregating) là một kỹ thuật trong Ensemble Learning, đặc biệt hữu ích trong việc giảm độ phương sai (variance) và tăng độ chính xác (accuracy) của mô hình. Kỹ thuật này sử dụng phương pháp Bootstrapping để tạo ra nhiều bản sao dữ liệu từ dữ liệu gốc và sử dụng các bản sao đó để xây dựng nhiều mô hình con (sub-models). Kết quả cuối cùng sẽ được tính trung bình hoặc trọng số của các mô hình con đó.

Trong kỹ thuật bagging, mỗi bản sao dữ liệu được tạo ra bằng cách lấy một tập dữ liệu ngẫu nhiên từ dữ liệu gốc, với các điểm dữ liệu có thể được chọn lại nhiều lần. Sau đó, mỗi bản sao dữ liệu được sử dụng để huấn luyện một mô hình con. Việc huấn luyện mô hình con trên mỗi bản sao dữ liệu sẽ giúp giảm độ chênh lệch và tăng độ chính xác của mô hình tổng quát.

Kết quả cuối cùng của bagging có thể được tính bằng cách trung bình hoặc tổng hợp các dự đoán của các mô hình con. Điều này có nghĩa là chúng ta có thể sử dụng nhiều mô hình con để cải thiện độ chính xác và giảm thiểu tình trạng overfitting.

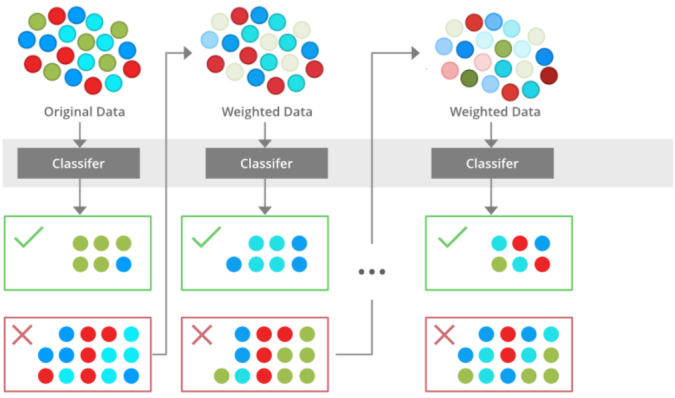


Hình 1.1: Ví dụ minh họa về Bagging

1.3.2 Boosting

Boosting là một phương pháp trong học máy dựa trên việc tạo ra mô hình tổng thể bằng cách sử dụng nhiều mô hình cơ bản và hợp nhất chúng để giảm độ chệch và phương sai để cải thiện độ chính xác của mô hình. Phương pháp này được sử dụng để giải quyết các bài toán phân loại và hồi quy.

Trong boosting, mỗi mô hình cơ bản được huấn luyện trên tập dữ liệu đã được chỉnh sửa để nâng cao độ chính xác của mô hình. Quá trình này được lặp đi lặp lại cho đến khi mô hình đạt độ chính xác tối đa hoặc đến số lần quy định. Mô hình tổng thể được tạo ra bằng cách sử dụng tổng trọng số của các mô hình cơ bản.

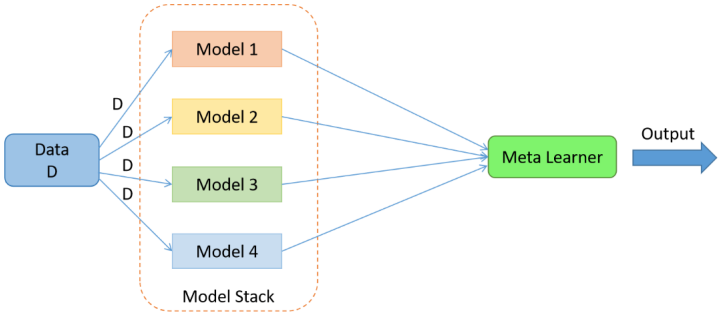


Hình 1.2: Ví dụ minh họa về Boosting

1.3.3 Stacking

Stacking là một phương pháp trong ensemble learning, trong đó một tập hợp các mô hình được dự báo và các dự báo đó được sử dụng như đầu vào cho một mô hình cuối cùng để tạo ra một dự báo cuối cùng. Mô hình cuối cùng này thường được xem như một mô hình tổng hợp hoặc một mô hình meta-model, và có thể là một mô hình đơn giản như một mô hình hồi quy hoặc một mô hình phức tạp hơn như một mô hình neural network.

Stacking có thể tăng cường khả năng dự báo của các mô hình tổng hợp bằng cách cải thiện sự tổng hợp của các mô hình con, và có thể giúp giảm overfitting bằng cách chủ yếu tập trung vào dự báo của mô hình tổng hợp.



Hình 1.3: Ví dụ minh họa về Stacking

Chương 2

Boosting

2.1 Adaptive Boosting

2.1.1 Thuật toán Forward Stagewise Additive Modeling

Boosting là một phương pháp được xây dựng bằng cách sử dụng tập hợp các hàm cơ bản. Trong trường hợp này thì hàm cở bản là mô hình phân loại $G_m(x) \in \{-1, 1\}$. Tổng quát hơn, khai triển hàm cơ bản có dạng:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{2.1}$$

trong đó $\beta_m, m = 1, 2, ..., M$ là các hệ số mở rộng và $b(x, \gamma) \in \mathbb{R}$ là các hàm cơ bản của biến x và γ là tham số đặc trưng.

Thuật toán Forward Stagewise Additive Modeling (FSAM) là một phương pháp học máy dựa trên mô hình tuyến tính, được sử dụng để giải quyết các bài toán dự đoán. FSAM là một phương pháp tiếp cận tuyến tính nhưng có thể xây dựng các mô hình phi tuyến.

Trong FSAM, mô hình được xây dựng dưới dạng tổng trọng số của các hàm số cơ bản (base functions). Mỗi hàm số cơ bản là một hàm số đơn giản có thể được sử dụng để xấp xỉ dữ liệu. Ví dụ về hàm số cơ bản là các hàm bậc thấp (linear, quadratic), các hàm số chuẩn (Gaussian), hoặc các hàm số cơ bản phi tuyến (spline, polynomial).

Algorithm 1: Forward Stagewise Additive Modeling

1. Initialize $f_0(x) = 0$
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Chi tiết thuật toán Forward Stagewise Additive Modeling được biểu diễn trong thuật toán 1. Ban đầu, mô hình được khởi tạo với trọng số bằng 0. Tiếp theo, FSAM thực hiện một quá trình tìm kiếm tối ưu để tăng dần trọng số của các hàm số cơ bản để cải thiện mô hình. Quá trình này được thực hiện bằng cách sử dụng thuật toán gradient descent hoặc các phương pháp tối ưu khác.

Ở mỗi vòng lặp, FSAM chọn một hàm số cơ bản mới để thêm vào mô hình bằng cách tìm kiếm hàm số cơ bản có ảnh hưởng lớn nhất đến giá trị dự đoán của mô hình. Hàm số cơ bản được chọn được thêm vào mô hình với một trọng số nhỏ để tăng dần trọng số của mô hình. Quá trình thêm hàm số cơ bản được lặp lại cho đến khi đạt được độ chính xác mong muốn hoặc không có hàm số cơ bản nào còn có ảnh hưởng đến giá trị dự đoán của mô hình.

2.1.2 Thuật toán AdBoost.M1

Đầu tiên, chúng ta bắt đầu bằng cách mô tả thuật toán boosting phổ biến do Freund và Schapire (1997) gọi là "AdaBoost.M1". Xét bài toán hai lớp, với đầu ra biến được mã hóa là $Y \in \{-1, 1\}$. Cho trước vector gồm các biến dự đoán X, mô hình phân loại $G(x)$ để dự đoán một trong hai giá trị $\{-1, 1\}$. Tỷ lệ lỗi trong quá trình huấn luyên là:

$$\bar{err} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i))$$

và kỳ vọng tỷ lệ lỗi về dự đoán trong tương lai là $E_{XY}I(Y \neq G(X))$.

Một thuật toán loại yếu (weak classifier)là thuật toán có tỷ lệ lỗi chỉ tốt hơn một chút so với dự đoán ngẫu nhiên. Mục đích của boosting là áp dụng tuần tự thuật toán phân loại yếu cho các phiên bản dữ liệu được sữa đổi nhiều lần, do đó sẽ tạo ra một chuỗi các mô hình phân loại yếu.

Tất cả các dự đoán được tổng hợp lại thông qua trọng số để đưa ra dự đoạn cuối cùng:

$$G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x)) \tag{2.2}$$

Trong đó $\alpha_1, \alpha_2, ..., \alpha_M$ được tính toàn bằng thuật toán boosting và trong số sự đóng góp của từng mô hình phân loại $G_m(x)$.

Dữ liệu được sữa đổi tại mỗi bước boosting bao gồm áp dụng trong số $w_1, w_2, .., w_N$ cho mỗi quan sát huấn luyện $(x_i, y_i), i = 1, 2, ..., N$. Đầu tiên khởi tạo tất cả các trọng số là$w_i = \frac{1}{N}$. Với các lần lặp liên tiếp $m = 2, 3, ..., M$ các trọng số của quan sát được tính toán lại và thuật toán phân loại được áp dụng lại cho các quan sát có trọng số. Ở bước m, những quan sát bị phân loại sai bởi mô hình phân loại $G_{m-1}(x)$ gây ra ở bước trước có trọng số của chúng tăng lên. Khi các lần lặp lại liên tục, các quan sát khó phân loại chính xác sẽ nhận được ảnh hưởng ngày càng tăng. Do đó, mỗi mô hình phân loại kế tiếp phải tập chung phân loại những quan sát mà mô hình phân loại trước đó đã phân loại sai.

Algorithm 2: AdaBoost.M1

1. Initialize the observation weights t $w_i = 1/N, \quad i = 1, 2, ..., N$
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i)}{\sum_{i=1}^{N} w_i}$$
   (c) Compute $\alpha_m = log((1 - err_m)/err_m)$
   (d) Set $w_i \longleftarrow w_i . \exp[\alpha_m . I(y_i \neq G_m(x_i)], i = 1, 2, .., N.$
3. Output $G(x) = sign[\sum_{m=1}^{M} \alpha_m G_m(x)]$.

Chi tiết thuật toán AdaBoost.M1 được hiển thị trong thuật toán 2. Tại bước 2a, mô hình phân loại $G_m(x)$ được xây dựng dựa trên trọng số của các quan sát. Tỷ lệ lỗi của trọng số được tính ở bước 2b. Bước 2c, tính trọng số α_m của $G_m(x)$ để ghép vào $G(x)$ (bước 3). Trọng số của mỗi quan sát được cập nhật lại cho vòng lặp tiếp theo tại bước 2d. Trọng số của những quan sát phân loại sai bởi $G_m(x)$ được thay đổi theo một tỷ lệ $e^{(\alpha_m)}$ để tăng sức ảnh hưởng của chúng trong việc xây dựng mô hình phân loại tiếp theo $G_{m+1}(x)$.

Chúng ta chứng minh AdaBoost.M1(thuật toán 2) và Forward Stagewise Additive Modeling (thuật toán 1) tương đương nhau với hàm mất mát là:

$$L(y, f(x)) = \exp(-yf(x)) \tag{2.3}$$

Với AdaBoost, các hàm cơ bản là các mô hình phân loại $G_m(x) \in \{-1, 1\}$. Sử dụng hàm mát là hàm mũ, ta cần tìm:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} \exp\left[-y_i(f_{m-1}(x_i) + \beta G(x_i))\right]$$

với mô hình phân loại G_m và hệ số tương ứng β_m được thêm vào mỗi bước. Điều này được diễn tả như sau:

$$(\beta_m, G_m) = \arg\min_{\beta, G} \sum_{i=1}^{N} w^{(m)} \exp\left(\beta G(x_i)\right) \tag{2.4}$$

với $w_i^{(m)} = \exp(-y_i f_{-y_i f_{m-1}(x_i)})$. Vì mỗi $w_i^{(m)}$ không phụ thuộc vào β hoặc $G(x)$ nên nó có thể coi là một trọng số cho mỗi quan sát. Trong số phụ thuộc vào $f_{m-1}(x_i)$ vì vậy giá trị của trọng số thay đổi với mỗi vòng lặp m.

Để giải bài toán (2.4) thì ta sẽ giải qua hai bước. Đầu tiên, với mọi $\beta \geq 0$, bài toán (2.4) với $G_m(x)$ là:

$$G_m = \arg\min_{G} \sum_{i=1}^{N} w_i^m I(y \neq G(x_i)), \tag{2.5}$$

đây là mô hình phân loại cực tiểu hóa tỷ lệ lỗi của trọng số trong việc dự đoán y. Khi đó (2.4) có thể viết lại dưới dạng:

$$e^{-\beta} . \sum_{y_i = G(x_i)} w_i^{(m)} + e^{\beta} . \sum_{y_i \neq G(x_i)} w_i^{(m)}, \tag{2.6}$$

(2.6) có thể được viết lại như sau:

$$(e^{\beta} - e^{-\beta}).\sum_{i=1}^{N} w_i^m I(y_i \neq G(x_i)) + e^{-\beta}.\sum_{i=1}^{N} w_i^{(m)}. \tag{2.7}$$

Khi đó, bài toán (2.4) ta giải được β:

$$\beta_m = \frac{1}{2}\log(\frac{1 - err_m}{err_m}) \tag{2.8}$$

trong đó err_m là tỷ lệ lỗi của trọng số được cực tiểu:

$$err_m = \frac{\sum_{i=1}^{N} w_i^m I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i^{(m)}} \tag{2.9}$$

Sau đó, hàm $f_m(x)$ được cập nhật:

$$f_m(x) = f_{m-1}(x) + \beta_m G_m(x)$$

trọng số được cập nhật lại cho lần lặp tiếp theo là:

$$w_i^{(m+1)} = w_i^{(m)}.e^{-\beta_m y_i G_m(x_i)} \tag{2.10}$$

Ta nhận thấy rằng $-y_i G_m(x_i) = 2.I(y_i \neq G_m(x_i)) - 1$, khi đó (2.10) trở thành:

$$w_i^{(m+1)} = w_i^{(m)}.e^{\alpha_m I(y_i \neq G_m(x_i))}.e^{-\beta_m} \tag{2.11}$$

trong đó $\alpha_m = 2\beta_m$ là đại lượng được định nghĩa bước 2c trong thuật toán AdaBoost (Thuật toán 2). $e^{-\beta_m}$ trong phương trình (2.11) nhân cho tất cả trọng số với một giá trị giống nhau, do đó nó không có ảnh hưởng gì nên ta có thể bỏ đi được. Như vậy, phương trình (2.11) tương đương với bước 2(d) của thuật toán 2.

Bước 2(a) của thuật toán Adaboost.M1 (thuật toán 2) như là một phương pháp giải bài toán tối ưu (2.7). Do đó, chúng ta có thể kết luận răng Adaboost.M1 là cực tiểu hóa hàm mất mát là hàm mũ (2.3) thông qua thuật toán forward-stagewise additive modeling (thuật toán 1).

2.2 Gradient Boosting

Ta chia những giá trị của biến dự đoán liên quan với nhau thành các các vùng riêng biêt $R_j, j = 1, 2, ..., J$, biểu diễn bởi các nút cuối của cây. Hằng số γ_j được gán cho mỗi vùng như vậy và quy tắc dự đoán là

$$x \in R_j \longrightarrow f(x) = \gamma_j$$

.

Vì vậy, mỗi cây được biểu diễn như sau:

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{2.12}$$

với tham số $\Theta = \{R_j, \gamma_j\}_1^J$. J được coi là một siêu tham số.
Các tham số được tìm bằng cách cực tiểu hóa hàm mất mát

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{x_j \in R_j} L(y_i, \gamma_j) \tag{2.13}$$

Mô hình cây boosted là tổng các cây con như vậy:

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{2.14}$$

được tạo ra từ phương pháp forward stagewise (thuật toán 1). Trong mỗi bước của phương pháp forward stagewise, chúng ta cần tìm

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \tag{2.15}$$

với cho tập các vùng và $\Theta_m = \{R_{jm}, \gamma_{jm}\}_1^{Jm}$ là hằng số của cây tiếp theo, cho mô hình $f_{m-1}(x)$.

Việc tìm hằng số tối ưu γ_{jm} cho mỗi vùng R_{jm} không quá phức tạp

$$\hat{\gamma}_{jm} = \arg\min_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm}) \tag{2.16}$$

Có thể dựa trên tối ưu hóa số (Numerical Optimization) để xây dựng các thuật toán để giải quyết (2.15) với bất kỳ hàm mất mát nào có thể đạo hàm. Sử dụng $f(x)$ để dự đoán y trên dữ liệu huấn luyện sẽ gây ra hàm mất mát là:

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{2.17}$$

Mục tiêu là tối ưu hóa $L(f)$ theo f, trong đó $f(x)$ là tổng của các cây (2.14) và cực tiểu (2.17) được xem là bài toán tối ưu hóa số học.

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f}} L(\mathbf{f}) \tag{2.18}$$

trong đó "tham số" $\mathbf{f} \in \mathbb{R}^N$ là giá trị của hàm xấp xỉ $f(x_i)$ tại mỗi điểm trong số N điểm dữ liệu x_i:

$$\hat{\mathbf{f}} = \{f(x_1), f(x_2), ..., f(x_N)\}$$

Quá trình giải bài tối tối ưu (2.18) như là tổng các vector thành phần

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m, \quad \mathbf{h}_m \in \mathbb{R}^N,$$

trong đó $\mathbf{f}_0 = \mathbf{h}_0$ là giả định ban đầu và mỗi $\mathbf{f}_m$ bước tiếp theo được tạo ra dựa trên vector tham số $\mathbf{f}_{m-1}$, kết được tổng cộng của các bước cập nhật đã được tăng trước đó. Các phương pháp tối ưu số khác nhau định nghĩa cách tính mỗi vector bổ sung (increment vector) $\mathbf{h}_m$.

2.2.1 Steepest Descent

Steepest Descent chọn $\mathbf{h}_m = -\rho_m \mathbf{g}_m$ trong đó ρ_m là một hệ số và $\mathbf{g}_m \in \mathbb{R}^N$ là gradient của $L(\mathbf{f})$ được tinh tại $\mathbf{f} = \mathbf{f}_{m-1}$. Gradient $\mathbf{g}_m$ được tính theo

$$g_{im} = \left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{f(x_i)=f_{m-1}(x_i)} \tag{2.19}$$

Độ bài bước nhảy ρ_m được tính theo

$$\rho_m = \arg\min_{\rho} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m) \tag{2.20}$$

Sau đó f_m được cập nhật lại

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m$$

và quá trình này được lặp lại tại vòng lặp tiếp theo. Steepest Descent có thể được xem là phương pháp đưa ra kết quả tối ưu nhanh, vì $-\mathbf{g}_m$ là hướng giảm trong $\mathbb{R}^N$ mà $L(\mathbf{f})$ giảm nhanh nhất.

2.2.2 Thuật toán Gradient Boosting

Forward stage (Thuật toán 1)là một chiến lược tham lam. Tại mỗi bước, tìm một cây sao cho cây đó có làm cho (2.15) giảm nhiều nhất, đựa trên mô hình f_{m-1} và giá trị dự đoán của nó là $f_m(x_i)$. Do đó, các cây dự đoán $T(x_i, \Theta_m)$ tương đương với các thành phần của gradient âm (gradient negative) (2.19). Sự khác biệt giữa các thành phần cây $t_m = (T(x_1, \Theta_m), T(x_2, \Theta_m), ..., T(x_N, \Theta_m))$ không độc lập. Chúng bị ràng buộc có nút rẽ nhánh ở nút thứ j_m, trong khi gradient âm lại không bị ràng buộc và có hướng giảm tối đa.

Việc tìm $\hat{\gamma}_{jm}$ trong (2.16) trong phương pháp stagewise tương đương với (2.20). Điểm khác biệt (2.16) là tìm tìm các thành phần của t_m tương ứng với từng vùng cuối riêng biệt $\{T(x_i, \Theta_m)\}_{x_i \in R_{jm}}$.

Nếu việc cực tiểu hàm mất mát trên dữ liệu huấn luyện (2.17) là mục tiêu thì steepest descent là một phương pháp tốt để cực tiểu hàm mất mát. Gradient (2.19) có thể được tính với bất kỳ hàm mất mát $L(y, f(x))$ khả vi.

Gradient (2.19) chỉ được tính trên các điểm dữ liệu huấn luyện x_i, trong khi mục tiêu cuối cùng là tổng quát hóa $f_M(x)$ đến dữ liệu mới chưa được huấn luyện.
Một giải pháp để giải quyết vấn đề trên là tạo một cây $T(x; \Theta_m)$ tại vòng lặp thứ m có dự đoán t_m càng gần với gradient âm càng tốt. Ta sử dụng bình phương sai số để đo độ gần này:

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} \left(-g_{im} - T\left(x_i; \Theta\right)\right)^2 \tag{2.21}$$

Khớp cây T với các giá trị gradient âm (2.19) bằng phương pháp bình phương cực tiểu. Sau khi xây dựng được cây (2.21), các hằng số tương ứng của mỗi vùng được xác định bằng công thức (2.16).

Thuật toán Gradient Boosting với hàm mất mát bất kỳ như sau:

Algorithm 3: Gradient Boosting

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$
2. For $m = 1$ to M:
   (a) Compute so-called pseudo-residual:
   $$r_{im} = -\left[\frac{\partial L\left(y_i, f\left(x_i\right)\right)}{\partial f\left(x_i\right)}\right]_{f(x_i)=f_{m-1}(x_i)}$$
   (b) Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals.
   (c) Compute
   $$\gamma_m = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$$
   .
   (d) Update the model:
   $$f_m(x) = f_{m-1}(x) + \gamma_m h_m(x)$$
   .
3. Output $f_m(x)$.

2.2.3 Thuật toán Gradient cho bài toán phân loại K lớp

Đối với bài toán phân loại K lớp thì hàm mất mát được sử dụng là [3]

$$L(\{y_k, f_k(x)\}_1^K) = -\sum_{k=1}^{K} y_k log p_k(x) \tag{2.22}$$

trong đó $y_k = 1(\text{class} = \text{k}) \in \{0, 1\}$ và $p_k(x) = Pr(y_k = 1|x)$. Khi đó ta được

$$f_k(x) = log p_k(x) - \frac{1}{K}\sum_{l=1}^{K} log p_l(x) \tag{2.23}$$

hay tương đương với

$$p_k(x) = \frac{\exp(f_k(x)}{\sum_{l=1}^{K}\exp(f_l(x))} \tag{2.24}$$

Thay (2.24) vào (2.22) ta được

$$r_{ik} = -\left[\frac{\partial L\left(\{y_{il}, f_l\left(x_i\right)\}_{l=1}^{K}\right)}{\partial f_k\left(x_i\right)}\right]_{\{f_l(x)=f_{l,m-1}(x)\}_1^K} = y_{ik} - p_{k,m-1}(x_i) \tag{2.25}$$

Ở mỗi vòng lặp m, $p_{k,m-1}(\mathbf{x})$ được tính dựa trên $F_{k,m-1}(\mathbf{x})$ thông qua công thức (2.24). Do đó, tại mỗi vòng lặp này, chúng ta xây dựng K cây để dự đoán sai số hiện tại cho từng lớp trên thang xác suất. Mỗi cây có J nút lá, tương ứng với các vùng $R_{jkm} j = 1^J$. Các giá trị γ_{jkm} tương ứng với các vùng này là

$$\gamma_{jkm} = \frac{K-1}{K}\frac{\sum_{x_i \in R_{jkm}} r_{ik}}{\sum_{x_i \in R_{jkm}} |r_{ik}|(1 - |r_{ik}|}) \tag{2.26}$$

Khi đó ta được thuật toán Gradient Boosting cho bài toán phân loại K lớp như sau [4]:

Algorithm 4: Gradient Boosting for K-class Classification

1. Initialize $f_{k0}(x) = 0, \quad k = 1, 2, ..., K$

2. For $m = 1$ to M:

   (a) Set

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_j(x)}}, k = 1, 2, ...K.$$

   (b) For $k = 1$ to K:

   i. Compute $r_{ikm} = y_{ik} - p_k(x_i), \quad i = 1, 2, ..., N.$

   ii. Fit a regression tree to the targets $r_{ikm}, i = 1, 2, ..., N$,giving terminal regions $R_{jkm}, j = 1, 2, ..., Jm.$

   iii. Compute

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ikm}}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1 - |r_{ikm}|}), \quad j = 1, 2, ..., J_m$$

   .

   iv. Update $f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$.

3. Output $\hat{f}_k(x) = f_{kM}(x), \quad k = 1, 2, ..., K.$

2.3 Extreme Gradient Boosting

Ta xét K cây quyết định và mô hình

$$\sum_{k-1}^{K} f_k$$

trong đó f_k là dự đoán từ cây quyết định thứ k. Mô hình là tập hợp gôm k cây quyết định.

Giá trị dự đoán khi thực hiện boosting K lần được tính như sau:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i)$$

trong đó x_i là vector đặc trưng của điểm dữ liệu thứ i.

Ta định nghĩa hàm mất mát giữa giá trị dự đoán $\hat{y}$ và giá trị thực y là: $l(\hat{y}, y)$. Ta sẽ cực tiểu hàm mục tiêu sau

$$\mathcal{L}(f_t) = \sum_{i=1}^{n} l\left(\hat{y}_i, y_i\right)$$

.

Ta có:

$$\begin{aligned}\min_{f_t} \mathcal{L}(f_t) &= \min_{f_t} \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t)}) \\ &= \min_{f_t} \sum_{i=1}^{n} l(y_i, \sum_{k=1}^{t} f_k(x_i)) \\ &= \min_{f_t} \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\end{aligned}$$

Regularization giúp kiểm soát tốt độ phức tạp của mô hình, từ đó tránh được trạng thái overfitting của mô hình. Ta định nghĩa hàm phạt

$$\Omega = \gamma T + \frac{1}{2}\lambda w_j^2$$

trong đó γ và λ là các siêu tham số, T là số lượng lá của cây và w_j^2 là "điểm" (score) trên lá thứ j của cây.

Kết hợp hàm mất mát và hàm phạt, ta được hàm mục tiêu của mô hình như sau

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} l(\hat{y}_i, y_i) + \sum_{k=1}^{K} \Theta(f_k) \tag{2.27}$$

Việc tìm cực tiểu $\mathcal{L}(\theta)$ 2.27 tương tự như ta cực tiểu tất cả $\mathcal{L}^{(t)}$ là hàm mục tiêu ở bước thứ t:

$$\min_{f_t} \mathcal{L}^{(t)} = \min_{f_t} \sum_{i=1}^{n} l(y_i, \hat{y}^{(t-1)} + f_t(x_i)) + \Theta(f_t) \tag{2.28}$$

Áp dụng phương pháp xấp xỉ bậc hai (Second-order approximation), ta chuyển $\mathcal{L}^{(t)}$ như sau [2]

$$\mathcal{L}^{(t)} \backsimeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Theta(f_t) \tag{2.29}$$

trong đó $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ và $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

Ta loại bỏ hằng số không ảnh hưởng đến quá trình tối ưu, khi đó hàm mục tiêu tại bước thứ t là

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{I} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{2.30}$$

Ta đinh nghĩa tập $I_j = \{i | q(x_i) = j\}$ là tập chứa các chỉ số của các điểm dữ liệu được gán cho lá (leaf) thứ j.

Khi đó hàm mục tiêu $\tilde{\mathcal{L}}^{(t)}$ (2.30) được viết lại như sau

$$\begin{aligned}\min_{f_t} \tilde{\mathcal{L}}^{(t)} &= \min_{f_t} \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \min_{f_t} \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \gamma T + \frac{1}{2}\lambda \|w\|^2 \\ &= \min_{f_t} \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T\end{aligned} \tag{2.31}$$

Vì tất cả các điểm dữ liệu trên cùng một lá sẽ được dự đoán cùng một giá trị, và công thức tính tổng các giá trị dự đoán trên các lá.

Ta thấy rằng hàm mục tiệu $\tilde{\mathcal{L}}^{(t)}$ là hàm bậc hai với biến w_j vì thế ta có thể dễ dàng tính được $\min_{f_t} \tilde{\mathcal{L}}^{(t)}$ đạt tại

$$w_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Thay w_j vào hàm mục tiêu $\tilde{\mathcal{L}}^{(t)}$ ta được:

$$\tilde{\mathcal{L}}^{(t)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{2.32}$$

Ta thường không thể liệt kê được tất cả các cấu trúc cây q có thể có. Thay vào đó, ta sử dụng một thuật toán tham lam bắt đầu từ một lá duy nhất và thêm tuần tự các nhánh vào cây. Giả sử I_L và I_R là tập hợp các nút bên trái và bên phải sau khi phân tách. Đặt $I = I_L \cup I_R$, thì hàm mất mát sau khi phân tách là

$$\mathcal{L}_{\text{split}} = \frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma \tag{2.33}$$

Công thức (2.33) được dùng để đánh giá các ứng viên cho việc phân tách.

2.3.1 Thuật toán tìm phân tách

Thuật toán Basic Exact Greedy

Để tối ưu cho mô hình cây có hiệu suất tốt là tìm điểm phân tách tốt nhất đê phân biệt rõ giữa các điểm dữ liệu. Điểm phân tách của XGBoost phụ thuộc vào điểm số của câu trúc cây.
Câu hỏi đặt ra làm sao có thể đánh giá tát cả các điểm phân tách có thể và tính điểm và chọn tính năng cung cấp điểm tốt nhất. Thuật toán Basic Exact Greedy đã giải quyết được vấn đề trên. Chi tiết thuật toán Basic Extract như sau:

Algorithm 5: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node

Input: d, feature dimension

$gain \leftarrow 0$;

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$;

for *k=1* ***to*** m **do**

  $G_L \leftarrow 0, H_L \leftarrow 0$;

  for j *in sorted* $(I, by\ x_{jk})$ **do**

    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$;

    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$;

    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$;

  end

end

Output: Split with max score

Thuật toán xấp xỉ tìm phân tách

Thuật toán Exact Greedy Algorithm (thuật toán 5) thường được sử dụng trong các mô hình cây khác nhau, tuy nhiên nó không phù hợp với mục đích của mô hình XGBoost. Một trong những ưu điểm của XGBoost là thời gian chạy của nó. Thuật toán xấp xỉ là một trong các phương pháp cho phép XGBoost thực hiện tính toán của nó với tốc độ nhanh.

Algorithm 6: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**

  Propose $S_k = \{s_{k1}, s_{k2}, ..., s_{kl}\}$ by percentiles on feature k ;

  Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ ***to*** m **do**

  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$;

  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$;

end

Follow same step as in previous section to find max score only among proposed splits.

1. for k =1 to m do: vòng lặp tất cả các thuộ tính và thực hiện những bước sau.

2. Chia khoảng S_k phân vị trên các thuộc tính Thuật toán xấp xỉ đang chia khoảng đặc trưng thành các phân vị. Ví dụ, nếu chúng ta có đặc trưng $\mathbf{x}$ với 100 mẫu và giá trị đặc trưng từ 1 đến 100. Phương pháp này sẽ đơn giản là chia phân phối của đặc trưng $\mathbf{x}$ thành n bucket. Giả sử chúng ta muốn chia đặc trưng này thành 10 bucket khác nhau, vì chúng ta sử dụng phân vị, mỗi khay sẽ có

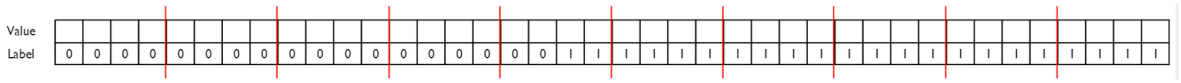
số mẫu tương đồng và trong ví dụ này, mỗi bucket chứa các giá trị từ $[1 \rightarrow 10]$, $[11 \rightarrow 20]$, và tiếp tục như vậy.

3. Đề xuất thực hiện trên toàn cục (Global) và cục bộ (Local): Sẽ được trình bày kỹ ở phần sau.

4. Tính G và H cho mỗi bucket

   G và H là hai giá trị mà mô hình cần tính để tìm điểm phân tách tốt.

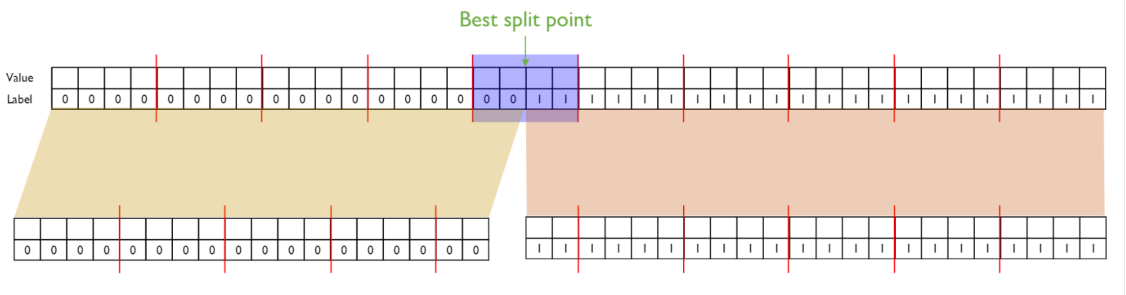
   Tương tự như thuật toán tham lam, chúng ta xử lý từng bucket một và thực hiện phân tách trên mỗi điểm dữ liệu. Ví dụ, giả sử mỗi bucket có 4 điểm dữ liệu khác nhau, do đó tính điểm phân tách sẽ được thực hiện 3 lần. Vì tính toán được thực hiện song song, nó có thể xác định điểm phân tách tốt nhất nhanh chóng.



Hình 2.1: Biểu diễn Bucket

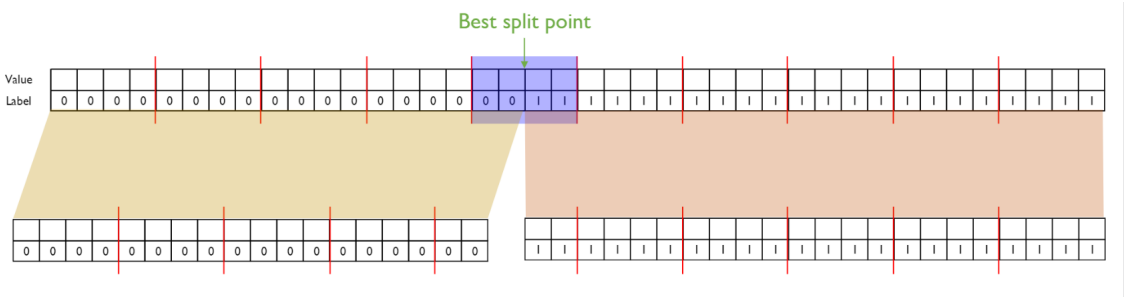
Ví dụ hình (2.1), giả sử ta có 40 điểm dữ liệu nếu ta dùng thuật toán Basic Exact Greedy Algorithm (nhớ ref) thì cần phải thực hiện 39 lần phân tách khác nhau theo thứ tự tuần tự. Tuy nhiên, với thuật toán xấp xỉ ta chia thành 10 bucket và thực hiện 30 lần phân tách được thực hiện song song.

Bucketing: Global Variant và Local Variant Bucketing có thể được thực hiện theo hai phương pháp khác nhau, một là phương pháp Global (trên cây) và hai là phương pháp Local (trên từng phân nhánh). Điều này có nghĩa là bucketing có thể được thực hiện một lần cho toàn bộ cây hoặc thực hiện bucketing mỗi khi có một phân nhánh mới.



Hình 2.2: Global Variant

Có thể thấy từ hình (2.2), với Global Variant, nút cha (parent node) sẽ chia thành hai nút con (child nodes) từ điểm phân tách tốt nhất. Những đường đỏ biểu thị các bucket, nó giữ nguyên các điểm bucket của nút cha.



Hình 2.3: Local Variant

Phương pháp Local variant thì thực hiện việc chia các bucket lại mỗi lần có một lần chia ở nút cha. Ở đây, hai nút con (trái và phải) sẽ có 10 bucket như hình (2.3).

Thuật toán Weighted Quantile Sketch

Thuật toán Weighted Quantile Sketch xuất phát từ thuật toán Sketch. Ý tưởng của thuật toán Sketch là khi xử lý các tập dữ liệu lớn, tính toán toàn bộ tập dữ liệu có thể trở nên chậm và không hiệu quả. Thay vào đó, thực hiện tính toán song song trên một tập con nhỏ hơn của dữ liệu và kết hợp các kết quả để thu được kết quả cuối cùng có thể có lợi. Điều này tương tự như sử dụng phân bố mẫu để xấp xỉ phân bố của tổng thể.

Thuật toán Weighted Quantile Sketch là một phương pháp được sử dụng để tính toán xấp xỉ phân bố của một tập dữ liệu lớn, đồng thời hỗ trợ việc tìm kiếm giá trị quantile. Khác với các phương pháp xấp xỉ thông thường, Weighted Quantile Sketch cho phép cân bằng trọng số giữa các mẫu dữ liệu, giúp đưa ra kết quả xấp xỉ chính xác hơn.

Thuật toán này hoạt động bằng cách chia tập dữ liệu ban đầu thành nhiều tập con, mỗi tập con được tính toán xấp xỉ histogram. Từ đó, các giá trị quantile có thể được tính toán thông qua việc ghép các histogram xấp xỉ và áp dụng phương pháp xấp xỉ dựa trên trọng số.

Khi sử dụng phương pháp normalized quantile, số lượng mẫu trong mỗi khoảng tương đương với nhau, tức là mỗi khoảng chứa số lượng mẫu bằng nhau. Tuy nhiên, khi sử dụng phương pháp weighted quantile, số lượng mẫu trong mỗi khoảng có thể khác nhau, nhưng tổng trọng số của các mẫu trong mỗi khoảng lại gần bằng nhau hoặc tương đương với nhau. Tức là, mỗi khoảng có tổng trọng số các mẫu trong đó gần bằng nhau hoặc tương đương nhau.

Ta có thể viết lại phương trình (2.30) như sau [1]

$$\sum_{i=1}^{n} \frac{1}{2} h_i \left(f_t \left(\mathbf{x}_i \right) - g_i / h_i \right)^2 + \Omega \left(f_t \right) + \text{constant} \tag{2.34}$$

là hàm mất mát bình phương của trọng số với các nhãn g_i/h_i và trọng số h_i.

Thuật toán Sparsity-Aware Split Finding

Trong thực tế, ma trận đầu vào **x** thường là ma trận thưa (sparse). Có nhiều nguyên nhân dẫn đến sự thưa này:

1. Giá trị của dữ liệu bị thiếu.
2. Số lượng phần tử có giá trị 0 nhiều.
3. Kết quả của kỹ thuật Feature Engineering (Mã hóa One-Hot).

XGBoost có thể xử lý dữ liệu thưa (sparse data) một cách hiệu quả, đồng thời đem lại khả năng mở rộng cho thuật toán, giúp tăng tốc độ thời gian chạy. Thuật toán 7 giúp hiểu rõ hơn về cách mà thuật toán xử lý dữ liệu thưa.

Algorithm 7: Sparsity-aware Split Finding

Input: I, instance set of current node

Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$

Input: d, feature dimension

Also applies to the approximate setting, only collect statistics of non-missing entries into buckets;

$gain \leftarrow 0$;

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} hi$;

for $k = 1$ ***to*** m **do**

  // enumerate missing value goto right;

  $G_L \leftarrow 0, H_L \leftarrow 0$;

  for j *in sorted* (I_k, *ascent order by* x_{jk}) **do**

    $G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$;

    $G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$;

    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$;

  end

  // enumerate missing value goto left;

  $G_R \leftarrow 0, H_R \leftarrow 0$;

  for j *in sorted* (I, *descent order by*x_{jk}) **do**

    $G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$;

    $G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$;

    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$;

  end

end

Output: Split and default directions with max gain

1. Đầu vào của thuật này là I là tập hợp các điểm dữ liệu đang được xử lý tại nút hiện tại trên cây. I_k là tập các điểm dữ liệu không bị thiếu.

2. Khởi tại $G = 0$ và $H = 0$ tham số cho việc phân tách nhánh.

3. Với mỗi đặc trưng $k \longrightarrow m$, thực hiện các bước sau:

Value	1.3		1.1	0.2		1.9	0.5		1.5	1.8
Class	1	0	1	0	0	1	0	0	1	1

Hình 2.4: Ví dụ bộ dữ liệu thưa

Giả sử chúng ta có tập dữ liệu sau đây với màu cam đại diện cho dữ liệu thưa.

4. Sắp xếp những dữ liệu bị thiếu sang bên phải

Value	0.2	0.5	0.8	1.1	1.3	1.5	1.9			
Class	0	0	1	1	1	1	1	0	0	0

Hình 2.5: Ví dụ sắp xếp những dữ liệu bị thiếu sang bên phải

Trong tập dữ liệu, sắp xếp các điểm dữ liệu bị thiếu sang bên phải và thực hiện phân tách (quy trình tương tự các thuật toán trước) Điểm phân tách tốt nhất được tìm như hình (2.6)

Value	0.2	0.5	0.8	1.1	1.3	1.5	1.9			
Class	0	0	1	1	1	1	1	0	0	0

Hình 2.6: Điểm phân tách tốt nhất khi các điểm dữ liệu thiếu được sắp xếp sang phải

5. Sắp xếp những điểm dữ liệu thiệu sang trái Cách tiếp cận tương tự nhưng lần này sắp xếp dữ liệu thiếu sang bên trái và với quá trình tính điểm phân tách tương tự sẽ cho kết quả:

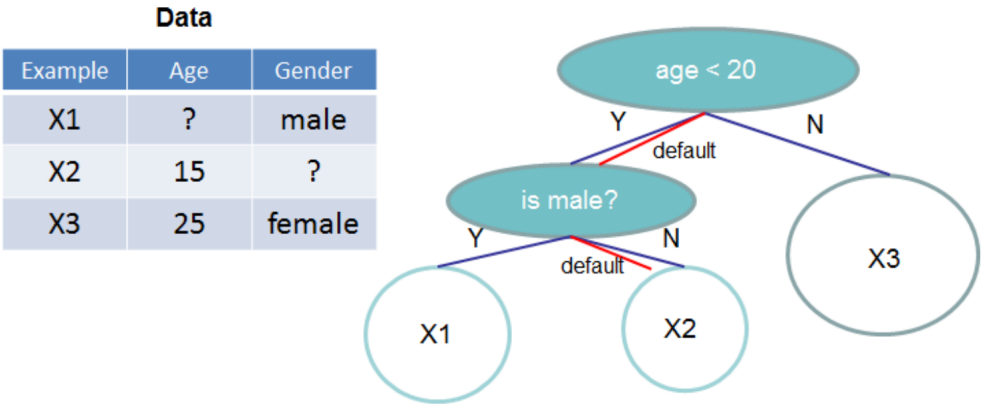
			0.2	0.5	0.8	1.1	1.3	1.5	1.9
0	0	0	0	0	1	1	1	1	1

Best split, default direction = left

Hình 2.7: Điểm phân tách tốt nhất khi các điểm dữ liệu thiếu được sắp xếp sang trái

Có thể thấy từ hình (2.7), khi dữ liệu thưa được sắp xếp sang bên trái, chúng ta có được sự phân tách rõ ràng hơn giữa hai lớp (0 và 1) và do đó việc sắp sang bên trái được chọn cho toàn bộ phân chia đặc trưng.

Quyết định hướng mặc định sẽ được thực hiện với tất cả các đặc trưng và mỗi đặc trưng sẽ có hướng riêng của nó. Trong ví dụ (2.8), đặc trưng tuổi có hướng sang trái là hướng mặc định của nó và đặc trưng giới tính nam có hướng sang phải là hướng mặc định của nó.



Hình 2.8: Một ví dụ về thuật toán Sparse Aware

Phân tách đầu tiên sẽ xảy ra với thuộc tính tuổi và vì chúng ta biết X1 bị thiếu trên cột tuổi nó sẽ di chuyển đến nút bên trái. X2 không thiếu trên cột tuổi và có tuổi nhỏ hơn 20, sẽ kết thúc trong nút bên trái. Một lần nữa, giới tính được chọn là tiêu chí để phân chia. X1 không thiếu trên đặc tính này và do đó kết thúc ở nút con bên trái. Trong khi đó, X2 bị thiếu trên cột giới tính, sẽ kết quả theo hướng phải (hướng mặc định).

2.3.2 Thiết kế hệ thống trong XGBoost

XGBoost là một thuật toán đạt được hiệu suất và độ chính xác cao đối các bài toán có lượng đặc trưng lớn. Để đạt hiệu suất cao thì XGBoost có các tính năng Column Block for Parallel Learning , Cache-aware Access, Blocks for Out-of-core Computation:

1. Column Block for Parallel Learning [1]:

   Column Block là một trong những tính năng quan trọng của XGBoost, nó giúp tăng tốc độ huấn luyện mô hình bằng cách sử dụng kiến trúc dữ liệu dạng block. Thay vì lưu trữ dữ liệu liên tục trong bộ nhớ, Column Block phân chia dữ liệu thành các khối (block) dựa trên cột (column) của ma trận dữ liệu đầu vào. Mỗi block chỉ chứa một số lượng nhỏ các dòng và tập trung vào các cột khác nhau. Điều này cho phép XGBoost sử dụng các luồng CPU khác nhau để huấn luyện mô hình trên các khối dữ liệu độc lập cùng một lúc. Điều này giúp tăng tốc độ huấn luyện mô hình đáng kể bằng cách sử dụng tối đa các lõi CPU có sẵn.

2. Cache-aware Access [1]:

   Cache-aware Access là một tính năng trong XGBoost giúp tối ưu hóa quá trình truy cập dữ liệu từ bộ nhớ trong việc huấn luyện mô hình. Khi XGBoost đọc dữ liệu đầu vào, nó sẽ tạo ra các khối

(block) và lưu chúng vào bộ nhớ cache của CPU để truy cập nhanh hơn trong quá trình huấn luyện. Bên cạnh đó, Cache-aware Access cũng giúp giảm thiểu việc truy cập dữ liệu từ bộ nhớ chính, giảm thiểu tình trạng xung đột dữ liệu và tăng tốc độ huấn luyện mô hình.

3. Blocks for Out-of-core Computation [1]:
   Với kích thước dữ liệu lớn, việc lưu trữ toàn bộ dữ liệu trong bộ nhớ chính để huấn luyện mô hình có thể gặp phải rào cản về khả năng bộ nhớ. Để giải quyết vấn đề này, XGBoost có tính năng Blocks for Out-of-core Computation. Điều này cho phép XGBoost tải các khối dữ liệu từ đĩa cứng vào bộ nhớ cache, thay vì tải toàn bộ dữ liệu vào bộ nhớ chính, giúp giảm bớt áp lực lên bộ nhớ và giúp huấn luyện mô hình có thể thực hiện được trên các máy tính có bộ nhớ nhỏ hơn.

2.4 So sánh Adaptive Boosting, Gradient Boosting và Extreme Gradient Boosting

Bảng 2.1, so sánh ưu điểm và nhược điểm của ba thuật toán Adaptive Boosting, Gradient Boosting và Extreme Gradient Boosting như sau:

	Ưu điểm	**Nhược điểm**
AdaBoost	- AdaBoost tăng độ chính xác của các mô hình học máy yếu. - AdaBoost không bị ảnh hưởng bởi hiện tượng overfitting vì nó thực hiện mỗi mô hình theo trình tự và có trọng số được gán cho chúng.	- Để huấn luyện AdaBoost, cần dữ liệu chất lượng vì nó rất nhạy cảm với dữ liệu nhiễu và các điểm dữ liệu ngoại lai (outliers).
GBM	- GBM thường có độ chính xác dự đoán cao. - Không yêu cầu tiền xử lý dữ liệu - thường hoạt động tốt với các giá trị phân loại. - Có thể xử lý dữ liệu bị mất	- GBM cải thiện để giảm thiểu các lỗi, nhưng điều này có thể dẫn đến quá chú trọng vào các giá trị ngoại lệ và gây ra hiện tượng overffit. - Tính toán tốn kém - thường yêu cầu nhiều cay quyết định vì thế có thể gây tốn thời gian và bộ nhớ.
XGB	- XGB dùng regularization L1 (Lasso Regression) và L2 (Ridge Regression) để không xảy ra hiện tượng overfitting. - XGB xử lý, tính toán song song vì vậy nó nhanh hơn so với GB. - XGB có khả năng xử lý các giá trị bị thiếu.	- XGB rất nhạy cảm với dữ liệu ngoại lệ.

Bảng 2.1: So sánh ba thuật toán Adboost, GBM và XGB

2.5 Kết quả thực nghiệm

Trong phần này, tác giả sẽ xây dựng một số mô hình phân loại như Decision Trees, Logistic Regression, K-Nearest Neighbors và tiến hành so sánh các mô hình đó với mô hình AdaBoost, Gradient Boosting và XGBoost. Các mô hình sẽ được cài đặt bằng ngôn ngữ Python và sử dụng bộ dữ liệu Churn để huấn luyện các mô hình.

2.5.1 Bộ dữ liệu sử dụng

Trong bài báo cào này, nhóm tác giả sẽ sử dụng bộ dữ liệu Churn gồm thông tin của 7043 khách hàng. Dữ liệu bào gồm các thông tin:

- Khách hàng đã rời mạng trong vòng 1 tháng qua - cột được gọi là "Churn".
- Các dịch vụ mà mỗi khách hàng đã đăng ký- điện thoại, internet, bảo mật trực tuyến, sao lưu trực tuyến, bảo vệ thiết bị, hỗ trợ kỹ thuật và xem phim - truyền hình trực tuyến.
- Thông tin tài khoản của khách hàng - thời gian họ đã trở thành khách hàng, hợp đồng, phương thức thanh toán, thanh toán trực tuyến, chi phí hàng tháng và tổng chi phí.
- Thông tin cá nhân của khách hàng: giới tính, độ tuổi, tình trạng hôn nhân.

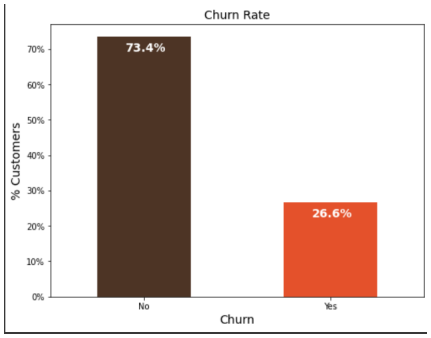
Thông tin bộ dữ liệu Churn như sau:

Shape: (7043, 21)

	customerID	gender	SeniorCitizen	Partner	Dependents	...	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
0	7590-VHVEG	Female	0	Yes	No	...	Yes	Electronic check	29.85	29.85	No
1	5575-GNVDE	Male	0	No	No	...	No	Mailed check	56.95	1889.5	No
2	3668-QPYBK	Male	0	No	No	...	Yes	Mailed check	53.85	108.15	Yes
3	7795-CFOCW	Male	0	No	No	...	No	Bank transfer (automatic)	42.30	1840.75	No
4	9237-HQITU	Female	0	No	No	...	Yes	Electronic check	70.70	151.65	Yes
5	9305-CDSKC	Female	0	No	No	...	Yes	Electronic check	99.65	820.5	Yes
6	1452-KIOVK	Male	0	No	Yes	...	Yes	Credit card (automatic)	89.10	1949.4	No
7	6713-OKOMC	Female	0	No	No	...	No	Mailed check	29.75	301.9	No

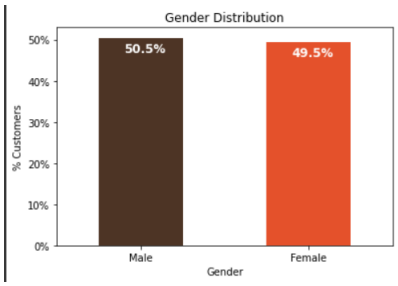
Hình 2.9: Thông tin về bộ dữ liệu Churn

Tiến hành khai phá dữ liệu bằng một vài bước đơn giản, ta thấy 26.6% khách hàng rời mạng



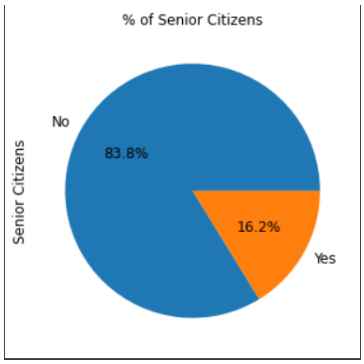
Hình 2.10: Tỷ lệ Churn

Khoảng một nửa khách hàng trong bộ dữ liệu là nam trong khi nửa còn lại là nữ.



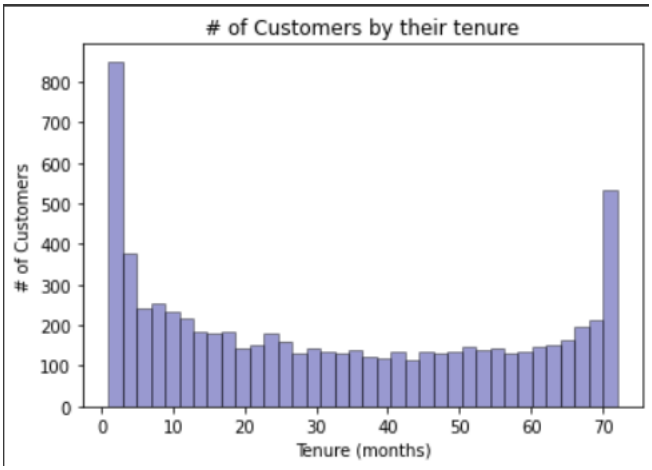
Hình 2.11: Tỷ lệ giới tính

Chỉ có 16% khách hàng là người cao tuổi, hầu hết khách hàng của công ty là những người trẻ tuổi.



Hình 2.12: Biểu đồ tỷ lệ khách hàng cao tuổi của công ty viễn thông

Nhìn biểu đồ 2.13, chúng ta có thể thấy rằng rất nhiều khách hàng đã làm việc với công ty viễn thông chỉ trong một tháng, trong khi khá nhiều khách hàng đã sử dụng trong khoảng 72 tháng. Điều này có thể do khách hàng có các hợp đồng khác nhau và dựa trên hợp đồng đó, việc ở lại hoặc rời khỏi công ty viễn thông sẽ trở nên dễ dàng hơn hoặc khó khăn hơn cho khách hàng.



Hình 2.13: Biểu đồ tần số thời gian sử dụng của khách hàng

2.5.2 Kết quả chay mô hình

Kết quả thực nghiệm và code cài đặt mô hình được up load tại link `https://colab.research.google.com/drive/1u0seXfUKX2A3Tpm8j-2OcBGjRHEbm2if?usp=sharing`. Trong bài báo cáo này, tác giả cài đặt và chạy ba mô hình chính là AdaBoost, Gradient Boosting và XGBoost và tiến hành so sánh với các mô hình Decision Trees, Logistic Regression, K-Nearest Neighbors.

Hầu hết các thuật toán học máy đều có nhiều tham số do người dùng định nghĩa ảnh hưởng đến hiệu suất phân loại. Trong Bảng 2.2, danh sách các tham số chính trên các mô hình. Trong bài báo cáo này, tác giải chỉ tập trung vào và điều chỉnh các tham số quan trọng, được liệt kê dưới đây. Mặc dù các giá trị mặc định thường được sử dụng cho các tham số do người dùng định nghĩa, nhưng cần phải thực hiện thử nghiệm kinh nghiệm để xác định các giá trị tối ưu cho các tham số này và đảm bảo rằng kết quả phân loại tốt nhất có thể đã được đạt được. Để làm được điều này, 'GridSearchCV' được thiết lập để tạo ra một bộ điều chỉnh siêu tham số lưới trên tập huấn luyện.

Thuật toán	Tham số	Miêu tả
DT	- max_depth = [1, 5, 10, 20]	-max_depth: Giới hạn độ sâu tối đa của cây
LR	- solvers = ['newton-cg', 'lbfgs', 'liblinear'] - penalty = ['l2', 'l1'] - c_values = [100, 10, 1.0, 0.1, 0.01]	- solvers: Thuật toán tối ưu hóa được sử dụng để tìm ra các tham số tối ưu cho mô hình. - penalty: Hàm phạt. - c_values: Giá trị nghịch đảo của hệ số điều chỉnh.
kNN	- n_neighbors = range(1, 21, 2) - weights = ['uniform', 'distance'] - metric = ['euclidean', 'manhattan', 'minkowski']	- n_neighbors: Số lượng láng giềng gần nhất để sử dụng. - weights: Phương pháp tính trọng số giữa các điểm láng giềng. - metric: Hàm tính khoảng cách khoảng cách để tính khoảng cách giữa các điểm dữ liệu.
AdaBoost	- n_estimators = [50, 100, 200] - learning_rate = [0.1, 0.5, 1.0]	- n_estimators: Số lượng cây quyết định trong mô hình. - learning_rate: Hệ số điều chỉnh tốc độ học của thuật toán.
GBM	- learning_rate = [0.025, 0.05, 0.1, 0.2, 0.3] - max_depth = [2, 3, 5, 7, 10] - min_samples_split = [2, 5, 10, 20] - max_features = ['sqrt', 'log2'] -subsample = [0.15, 0.5, 0.75, 1.0]	- learning_rate: Hệ số điều chỉnh tốc độ học của thuật toán. - max_depth: Độ sâu tối đa của các cây quyết định trong mô hình. - min_samples_split: Số lượng mẫu tối thiểu cần có trong một nốt (node) để tiếp tục phân tách. - max_features: Số lượng đặc trưng được chọn ngẫu nhiên để xây dựng cây quyết định. - subsample: Tỷ lệ mẫu được sử dụng để xây dựng một cây quyết định
XGB	- learning_rate = [0.025, 0.05, 0.1, 0.2, 0.3] - max_depth = [2, 3, 5, 7, 10] - min_samples_split = [2, 5, 10, 20] - max_features = ['sqrt', 'log2'] - subsample = [0.15, 0.5, 0.75, 1.0]	- learning_rate: Hệ số điều chỉnh tốc độ học của thuật toán. - max_depth: Độ sâu tối đa của các cây quyết định trong mô hình. - min_samples_split: Số lượng mẫu tối thiểu cần có trong một nốt (node) để tiếp tục phân tách. - max_features: Số lượng đặc trưng được chọn ngẫu nhiên để xây dựng cây quyết định. - subsample: Tỷ lệ mẫu được sử dụng để xây dựng một cây quyết định

Bảng 2.2: Tham số Cấu hình của thuật toán.

Ta sử dụng accuracy, precision, recall và f1-score là độ đo dể đánh giá. Kết quả được tổng hợp trong bảng dưới đây:

	F1-Score	Recall	Precision	Accuracy
DT	0.7856	0.8234	0.7723	0.7936
LR	0.7949	0.8231	0.7742	0.7933
kNN	0.8302	0.8642	0.7911	0.8078
AdaBoost	0.8351	0.8535	0.8176	0.8315
GBM	0.8469	0.8502	0.8436	0.8463
XGB	0.8472	0.8502	0.8442	0.8466

Bảng 2.3: Đánh giá các mô hình

Nhìn vào bảng 2.3, ta có thể thấy các thuật toán AdaBoost, Gradient Boosting và XGBoosting phân loại tốt hơn những mô hình còn lại và XGBoosting , Gradient Boosting cho kết quả vượt trội.

Bây giờ, ta sẽ đi so sánh thời gian chạy của hai mô hình XGBoosting và Gradient Boosting:

	Time (s)
GBM	6.0379
XGB	5.8478

Bảng 2.4: So sánh thời gian chạy của hai mô hình XGBoosting và Gradient Boosting

Từ bảng 2.4, ta có thể thấy thời gian chạy của thuật toán XGBoost nhanh hơn thuật toán Gradient Boosting. Vì thế, XGBoost là một thuật toán phân loại mạnh mẽ và có hiệu suất tốt trong việc phân loại dữ liệu và được sử dụng rộng rãi trong các cuộc thi và thử thách về dự đoán và phân loại.

Bảng 2.5, tổng hợp kết quả của độ chệch và phương sai của thuật toán AdaBoost, GBM, XGB và DT. Ta có thể thấy là ác thuật toán ensemble learning như Adaboost, XGB và GBM có độ chệch thấp hơn và phương sai thấp hơn so với cây quyết định đơn lẻ.

	Bias	Variance
DT	0.176	0.128
AdaBoost	0.165	0.037
GBM	0.151	0.048
XGB	0.156	0.051

Bảng 2.5: So sánh độ chệch và phương sai của một số thuật toán

Chương 3

Kết luận

Đồ án đã đạt được mục tiêu đề ra

Đồ án đã tìm hiểu kỹ thuật Boosting trong học máy.

Kết quả của đồ án

1. Giới thiệu bài toán phân loại và một số độ đo đánh giá mô hình phân loại.
2. Giới thiệu về Ensemble learning và trình bày tổng quan ba kỹ thuật chính trong Ensemble Learning đó là Bagging, Boosting và Stacking
3. Hiểu được ba thuật toán trong Boosing là Adaptive Boosting, Gradient Boosting và Extreme Gradient Boosting. So sánh ưu nhược điểm của ba thuật toán.
4. Tiến hành cài đặt và chạy ba mô hình và so sánh chúng với một số mô hình phân loại khác.

Kỹ năng đạt được

1. Cải thiện việc đọc hiểu và diễn đạt lại các tài liệu chuyên ngành
2. Nâng cao kỹ năng tìm kiếm, khai thác các tài liệu chuyên ngành liên quan tới chủ đề cần tìm hiểu.
3. Cải thiện kỹ năng viết báo cáo khoa học, viết bài báo cáo chuyên ngành và kỹ năng trình bày một báo cáo khoa học.

Tài liệu tham khảo

[1] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 2016, pp. 785–794.

[2] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)". In: *The annals of statistics* 28.2 (2000), pp. 337–407.

[3] Jerome H Friedman. "Greedy function approximation: a gradient boosting machine". In: *Annals of statistics* (2001), pp. 1189–1232.

[4] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.