**Lê Mạnh Hùng-1511008**

**Nguyễn Đức Quan-15110105**

**LightGBM: Cây quyết định tăng cường hiệu quả cao**

**Tóm tắt**

Gradient Decisioning Decision Tree (GBDT) là một thuật toán học máy phổ biến, và có một vài triển khai hiệu quả như XGBoost và pGBRT. Mặc dù nhiều tối ưu hóa kỹ thuật đã được áp dụng trong các triển khai này, hiệu quả và khả năng mở rộng vẫn không đạt yêu cầu khi thứ nguyên tính năng cao và kích thước dữ liệu lớn. Một lý do chính là đối với mỗi tính năng, họ cần phải quét tất cả các trường hợp dữ liệu để ước tính mức tăng thông tin của tất cả các điểm phân tách có thể, rất tốn thời gian. Để giải quyết vấn đề này, chúng tôi đề xuất hai kỹ thuật mới: Lấy mẫu một bên dựa trên nền (Gradient-based One-Side Sampling) (GOSS) và Gói tính năng độc quyền (Exclusive Feature Bundling) (EFB). Với GOSS, chúng tôi loại trừ một tỷ lệ đáng kể các phiên bản dữ liệu với các gradient nhỏ và chỉ sử dụng phần còn lại để ước tính mức tăng thông tin. Chúng tôi chứng minh rằng, vì các trường hợp dữ liệu với các gradient lớn hơn đóng một vai trò quan trọng hơn trong việc tính toán thông tin, GOSS có thể có được ước lượng khá chính xác về mức tăng thông tin với kích thước dữ liệu nhỏ hơn nhiều. Với EFB, chúng tôi gộp các tính năng loại trừ lẫn nhau (nghĩa là chúng hiếm khi lấy giá trị không đồng thời), để giảm số lượng đối tượng địa lý. Chúng tôi chứng minh rằng việc tìm kiếm gói tối ưu các tính năng độc quyền là NP-hard, nhưng thuật toán tham lam có thể đạt được tỷ lệ xấp xỉ khá tốt (và do đó có thể làm giảm hiệu quả số lượng các tính năng mà không làm tổn hại đến độ chính xác của việc xác định điểm phân tách). Chúng tôi gọi triển khai GBDT mới của chúng tôi với GOSS và EFB LightGBM. Thử nghiệm của chúng tôi trên nhiều bộ dữ liệu công cộng cho thấy, LightGBM tăng tốc quá trình đào tạo GBDT thông thường lên tới hơn 20 lần trong khi đạt được độ chính xác gần như giống nhau.

**1 Introduction**

Cây quyết định tăng cường độ dốc (GBDT) [1] là một thuật toán học máy được sử dụng rộng rãi, do hiệu quả, độ chính xác và khả năng diễn giải của nó. GBDT đạt được các màn trình diễn nghệ thuật tiên tiến trong nhiều nhiệm vụ học máy, chẳng hạn như phân loại nhiều lớp [2], nhấp vào dự đoán [3] và học cách xếp hạng [4]. Trong những năm gần đây, với sự xuất hiện của dữ liệu lớn (cả về số lượng tính năng và số lượng cá thể), GBDT đang đối mặt với những thách thức mới, đặc biệt là trong sự cân bằng giữa tính chính xác và hiệu quả. Việc triển khai thông thường của GBDT cần, đối với mọi tính năng, quét tất cả các cá thể dữ liệu để ước tính mức tăng thông tin của tất cả các điểm phân tách có thể có. Do đó, phức tạp tính toán của chúng sẽ tỷ lệ thuận với cả số lượng đối tượng và số lượng phiên bản. Điều này làm cho việc triển khai này tốn rất nhiều thời gian khi xử lý dữ liệu lớn.

Để giải quyết thách thức này, một ý tưởng đơn giản là giảm số lượng phiên bản dữ liệu và số lượng đối tượng địa lý. Tuy nhiên, điều này hóa ra là rất không tầm thường. Ví dụ, không rõ làm thế nào để thực hiện lấy mẫu dữ liệu cho GBDT. Trong khi có một số công trình lấy mẫu dữ liệu theo trọng lượng của chúng để tăng tốc quá trình đào tạo thúc đẩy [5, 6, 7], chúng không thể được áp dụng trực tiếp cho GBDT.

Hội nghị lần thứ 31 về các hệ thống xử lý thông tin thần kinh (NIPS 2017), Long Beach, CA, USA. vì không có trọng lượng mẫu nào trong GBDT cả. Trong bài báo này, chúng tôi đề xuất hai kỹ thuật mới cho mục tiêu này, như được trình bày bên dưới.

Lấy mẫu một bên dựa trên nền (GOSS). Mặc dù không có trọng số riêng cho dữ liệu trong GBDT, chúng tôi nhận thấy rằng các phiên bản dữ liệu với các gradient khác nhau đóng vai trò khác nhau trong việc tính toán tăng thông tin. Đặc biệt, theo định nghĩa về tăng thông tin, những trường hợp có độ dốc lớn hơn1 (tức là các trường hợp chưa được đào tạo) sẽ đóng góp nhiều hơn vào việc đạt được thông tin. Do đó, khi lấy mẫu các trường hợp dữ liệu, để duy trì tính chính xác của việc ước lượng thu thập thông tin, chúng ta nên giữ các trường hợp đó với các gradient lớn hơn (ví dụ: lớn hơn ngưỡng được xác định trước hoặc trong số các phần trăm trên cùng) và chỉ ngẫu nhiên thả những trường hợp với gradient nhỏ. Chúng tôi chứng minh rằng việc điều trị như vậy có thể dẫn đến ước lượng độ chính xác cao hơn so với lấy mẫu ngẫu nhiên thống nhất, với cùng tỷ lệ lấy mẫu mục tiêu, đặc biệt khi giá trị của thông tin đạt được có phạm vi rộng.

Gói tính năng độc quyền (EFB). Thông thường trong các ứng dụng thực tế, mặc dù có một số lượng lớn các tính năng, không gian tính năng khá thưa thớt, cung cấp cho chúng tôi khả năng thiết kế một phương pháp gần như không mất mát để giảm số lượng các tính năng hiệu quả. Cụ thể, trong một không gian tính năng thưa thớt, nhiều tính năng (gần như) độc quyền, tức là, chúng hiếm khi lấy các giá trị không đồng bộ. Ví dụ bao gồm các tính năng một nóng (ví dụ: biểu diễn một từ nóng trong khai thác văn bản). Chúng tôi có thể đóng gói các tính năng độc quyền một cách an toàn. Để đạt được mục tiêu này, chúng tôi thiết kế một thuật toán hiệu quả bằng cách giảm vấn đề bó tối ưu cho vấn đề tô màu đồ thị (bằng cách lấy các đối tượng như đỉnh và thêm cạnh cho mỗi hai đối tượng nếu chúng không loại trừ lẫn nhau) và giải quyết nó bằng thuật toán tham lam tỷ lệ xấp xỉ không đổi.

Chúng tôi gọi thuật toán GBDT mới với GOSS và EFB LightGBM2. Thử nghiệm của chúng tôi trên nhiều bộ dữ liệu công cộng cho thấy rằng LightGBM có thể đẩy nhanh quá trình đào tạo lên tới hơn 20 lần trong khi đạt được độ chính xác gần như giống nhau.

Việc còn lại của bài viết này được tổ chức như sau. Lúc đầu, chúng tôi xem xét các thuật toán GBDT và công việc liên quan trong Sec. 2. Sau đó, chúng tôi giới thiệu chi tiết về GOSS ở Sec. 3 và EFB ở Sec. 4. Các thí nghiệm của chúng tôi về LightGBM trên các tập dữ liệu công khai được trình bày trong phần Tài liệu. 5. Cuối cùng, chúng tôi kết luận bài viết trong phần Tài liệu. 6.

**2 Sơ bộ**

**2.1 GBDT và phân tích độ phức tạp**

GBDT là một mô hình tập hợp các cây quyết định, được huấn luyện theo trình tự [1]. Trong mỗi lần lặp lại, GBDT học các cây quyết định bằng cách lắp các gradient âm (còn được gọi là các lỗi còn lại).

Chi phí chính trong GBDT nằm trong việc học các cây quyết định, và phần tốn nhiều thời gian nhất trong việc học một cây quyết định là tìm ra các điểm phân chia tốt nhất. Một trong những thuật toán phổ biến nhất để tìm các điểm phân tách là thuật toán được sắp xếp trước [8, 9], liệt kê tất cả các điểm phân tách có thể có trên các giá trị tính năng được sắp xếp trước. Thuật toán này rất đơn giản và có thể tìm ra các điểm phân tách tối ưu, tuy nhiên, nó không hiệu quả trong cả tốc độ đào tạo và mức tiêu thụ bộ nhớ. Một thuật toán phổ biến khác là thuật toán dựa trên biểu đồ [10, 11, 12], như được hiển thị trong Alg. 13. Thay vì tìm các điểm phân tách trên các giá trị tính năng được sắp xếp, thuật toán dựa trên biểu đồ sẽ gộp các giá trị tính năng liên tục vào các thùng riêng biệt và sử dụng các thùng này để xây dựng các biểu đồ tính năng trong quá trình đào tạo. Vì thuật toán dựa trên biểu đồ hiệu quả hơn trong cả mức tiêu thụ bộ nhớ và tốc độ đào tạo, chúng tôi sẽ phát triển công việc của mình trên cơ sở.

Như được hiển thị trong Alg. 1, thuật toán dựa trên biểu đồ tìm các điểm chia tách tốt nhất dựa trên biểu đồ tính năng. Nó chi phí O (#data × #feature) cho việc xây dựng biểu đồ và O (#bin × #feature) cho việc tìm điểm phân tách. Vì #bin thường nhỏ hơn nhiều so với #data, nên việc xây dựng biểu đồ sẽ thống trị độ phức tạp tính toán. Nếu chúng ta có thể giảm #data hoặc #feature, chúng ta sẽ có thể tăng tốc đáng kể việc đào tạo GBDT.

**2.2 Công việc liên quan**

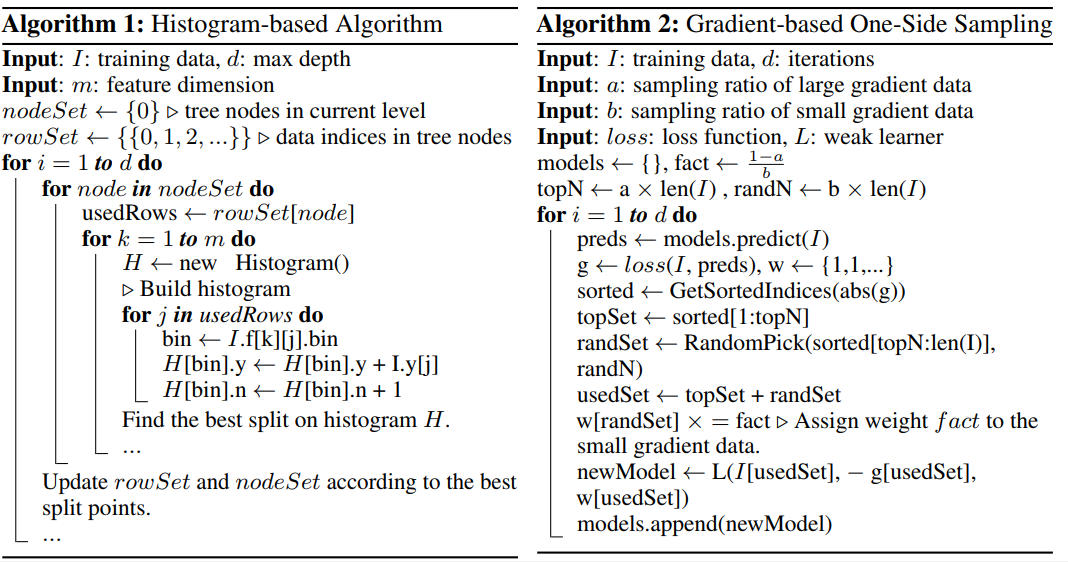
Đã có khá nhiều triển khai GBDT trong tài liệu, bao gồm XGBoost [13], pGBRT [14], scikit-learn [15], và gbm in R [16] 4. Scikit-learning và gbm trong R triển khai thuật toán đã định trước và pGBRT triển khai thuật toán dựa trên biểu đồ. XGBoost hỗ trợ cả thuật toán được sắp xếp trước và thuật toán dựa trên biểu đồ. Như được thể hiện trong [13], XGBoost hoạt động tốt hơn các công cụ khác. Vì vậy, chúng tôi sử dụng XGBoost làm đường cơ sở trong phần thử nghiệm.

Để giảm kích thước của dữ liệu đào tạo, một cách tiếp cận chung là giảm mẫu các cá thể dữ liệu. Ví dụ: trong [5], các phiên bản dữ liệu được lọc nếu trọng số của chúng nhỏ hơn ngưỡng cố định. SGB [20] sử dụng một tập con ngẫu nhiên để huấn luyện những người học yếu trong mỗi lần lặp lại. Trong [6], tỷ lệ lấy mẫu được điều chỉnh động trong tiến trình đào tạo. Tuy nhiên, tất cả các tác phẩm này ngoại trừ SGB [20] đều dựa trên AdaBoost [21] và không thể áp dụng trực tiếp cho GBDT vì không có trọng số riêng cho các phiên bản dữ liệu trong GBDT. Mặc dù SGB có thể được áp dụng cho GBDT, nhưng nó thường gây tổn thương chính xác và do đó nó không phải là một lựa chọn mong muốn.

Tương tự, để giảm số lượng đối tượng địa lý, việc lọc các đối tượng địa lý yếu là tự nhiên [22, 23, 7, 24]. Điều này thường được thực hiện bằng cách phân tích thành phần nguyên tắc hoặc theo đuổi chiếu. Tuy nhiên, các phương pháp này dựa vào giả định rằng các tính năng chứa dự phòng đáng kể, có thể không phải lúc nào cũng đúng trong thực tế (các tính năng thường được thiết kế với những đóng góp độc đáo của chúng và loại bỏ bất kỳ ảnh hưởng nào có thể ảnh hưởng đến độ chính xác của đào tạo ở một mức độ nào đó).

Các bộ dữ liệu quy mô lớn được sử dụng trong các ứng dụng thực thường khá thưa thớt. GBDT với thuật toán được sắp xếp trước có thể giảm chi phí đào tạo bằng cách bỏ qua các tính năng có giá trị bằng không [13]. Tuy nhiên, GBDT với thuật toán dựa trên biểu đồ không có giải pháp tối ưu hóa hiệu quả. Lý do là thuật toán dựa trên biểu đồ cần lấy các giá trị bin tính năng (tham khảo Alg. 1) cho mỗi cá thể dữ liệu bất kể giá trị đối tượng là 0 hay không. Nó được đánh giá cao rằng GBDT với thuật toán dựa trên biểu đồ có hiệu quả có thể tận dụng tài sản thưa thớt như vậy.

Để giải quyết các hạn chế của các tác phẩm trước đó, chúng tôi đề xuất hai kỹ thuật mới được gọi là Gradientbased One-Side Sampling (GOSS) và Gói tính năng độc quyền (EFB). Chi tiết sẽ được giới thiệu trong phần tiếp theo.



**3 Lấy mẫu một bên dựa trên Gradient**

Trong phần này, chúng tôi đề xuất một phương pháp lấy mẫu mới cho GBDT có thể đạt được sự cân bằng tốt giữa việc giảm số lượng các cá thể dữ liệu và giữ độ chính xác cho các cây quyết định đã học.

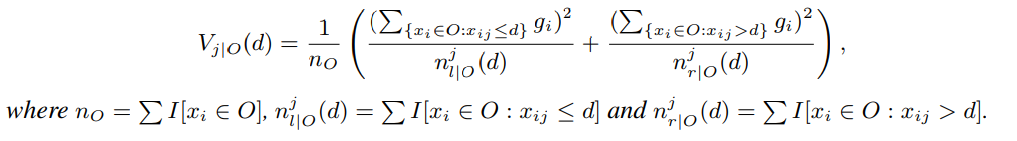
**3.1 Mô tả thuật toán**

Trong AdaBoost, trọng lượng mẫu phục vụ như một chỉ báo tốt cho tầm quan trọng của các cá thể dữ liệu. Tuy nhiên, trong GBDT, không có trọng số mẫu gốc và do đó các phương pháp lấy mẫu được đề xuất cho AdaBoost không thể được áp dụng trực tiếp. May mắn thay, chúng ta nhận thấy rằng gradient cho mỗi cá thể dữ liệu trong GBDT cung cấp cho chúng ta thông tin hữu ích để lấy mẫu dữ liệu. Tức là, nếu một cá thể được liên kết với một gradient nhỏ, lỗi huấn luyện cho cá thể này là nhỏ và nó đã được đào tạo tốt. Một ý tưởng đơn giản là loại bỏ các cá thể dữ liệu đó bằng các gradient nhỏ. Tuy nhiên, việc phân phối dữ liệu sẽ được thay đổi bằng cách làm như vậy, điều này sẽ làm tổn thương tính chính xác của mô hình đã học. Để tránh vấn đề này, chúng tôi đề xuất một phương pháp mới được gọi là Lấy mẫu một bên dựa trên Gradient (GOSS).

GOSS giữ tất cả các cá thể với các gradient lớn và thực hiện lấy mẫu ngẫu nhiên trên các cá thể với các gradient nhỏ. Để bù đắp ảnh hưởng đến việc phân phối dữ liệu, khi tính toán độ lợi thông tin, GOSS giới thiệu một hệ số không đổi cho các cá thể dữ liệu với các gradient nhỏ (xem Alg. 2). Cụ thể, GOSS trước hết sắp xếp các cá thể dữ liệu theo giá trị tuyệt đối của các gradient của chúng và chọn các trường hợp x 100% trên cùng. Sau đó, nó lấy mẫu ngẫu nhiên các trường hợp b x 100% từ phần còn lại của dữ liệu. Sau đó, GOSS khuếch đại dữ liệu được lấy mẫu với các gradient nhỏ bằng một hằng số (1-a)/b khi tính toán tăng thông tin. Bằng cách đó, chúng tôi tập trung hơn vào các trường hợp được đào tạo mà không thay đổi phân phối dữ liệu gốc nhiều.

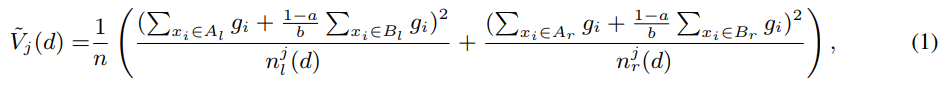
**3.2 Phân tích lý thuyết**

GBDT sử dụng cây quyết định để tìm hiểu một hàm từ không gian đầu vào X s đến không gian gradient G [1]. Giả sử chúng ta có một tập huấn luyện với n i.i.d. các cá thể {x1, · · ·, xn}, trong đó mỗi xi là một vectơ có kích thước s trong không gian X s. Trong mỗi lần lặp của tăng cường gradient, các gradient âm của hàm mất đối với đầu ra của mô hình được ký hiệu là {g1, · · ·, gn}. Mô hình cây quyết định chia tách từng nút ở tính năng thông tin nhất (với mức tăng thông tin lớn nhất). Đối với GBDT, mức tăng thông tin thường được đo bằng phương sai sau khi tách, được xác định như sau.

Định nghĩa 3.1 Cho O là tập dữ liệu huấn luyện trên một nút cố định của cây quyết định. Độ lợi chênh lệch của tính năng tách j tại điểm d cho nút này được định nghĩa là

Đối với tính năng j, thuật toán cây quyết định chọn d ∗ j = argmaxdVj (d) và tính mức tăng lớn nhất Vj (d ∗ j). 5 Sau đó, dữ liệu được chia theo tính năng j ∗ tại điểm dj∗ vào các nút con trái và phải.

Trong phương pháp GOSS được đề xuất của chúng tôi, trước tiên, chúng tôi xếp hạng các trường hợp đào tạo theo giá trị tuyệt đối của các gradient của chúng theo thứ tự giảm dần; thứ hai, chúng tôi giữ các trường hợp x 100% trên cùng với các gradient lớn hơn và nhận được một tập hợp con cá thể A; sau đó, đối với tập hợp còn lại Ac bao gồm (1 - a) × 100% các trường hợp có độ dốc nhỏ hơn, chúng tôi tiếp tục lấy mẫu ngẫu nhiên một tập con B với kích thước b × | Ac |; cuối cùng, chúng tôi chia các trường hợp theo mức chênh lệch ước tính V˜j (d) trên tập con A ∪ B, tức là,



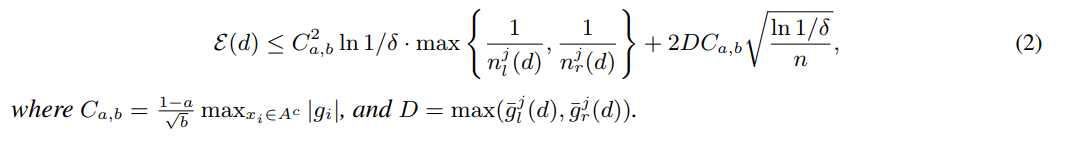
trong đó Al = {xi ∈ A: xij ≤ d}, Ar = {xi ∈ A: xij> d}, Bl = {xi ∈ B: xij ≤ d}, Br = {xi ∈ B: xij> d} và hệ số (1 – a)/b được sử dụng để bình thường hóa tổng của các gradient trên B trở lại thành giá trị của Ac.

Như vậy, trong GOSS, chúng tôi sử dụng ước lượng V˜j (d) trên một tập hợp con nhỏ hơn, thay vì Vj (d) chính xác trên tất cả các trường hợp để xác định điểm phân tách, và chi phí tính toán có thể giảm đáng kể. Quan trọng hơn, định lý sau chỉ ra rằng GOSS sẽ không mất đi độ chính xác đào tạo nhiều và sẽ làm tốt hơn việc lấy mẫu ngẫu nhiên. Do không gian hạn chế, chúng tôi để lại bằng chứng về định lý cho các tài liệu bổ sung.

**Định lý 3.2** *Chúng tôi biểu thị lỗi xấp xỉ trong GOSS là E(d) = |V˜j (d) − Vj (d)|*

*and g¯ j l P (d) =*

*Với xác suất ít nhất 1 - δ, chúng ta có*



Theo định lý, chúng ta có các cuộc thảo luận sau đây: (1) Tỷ lệ xấp xỉ tiệm cận của GOSS là .Nếu sự phân chia không quá cân bằng

Và , lỗi xấp xỉ sẽ bị chi phối bởi thuật ngữ thứ hai của Ineq. (2) giảm xuống 0 trong O (√n) bằng n → ∞. Điều đó có nghĩa là khi số lượng dữ liệu lớn, xấp xỉ là khá chính xác. (2) Lấy mẫu ngẫu nhiên là một trường hợp đặc biệt của GOSS với a = 0. Trong nhiều trường hợp, GOSS có thể làm tốt hơn việc lấy mẫu ngẫu nhiên, trong điều kiện C0, β> Ca, β − a, tương đương với

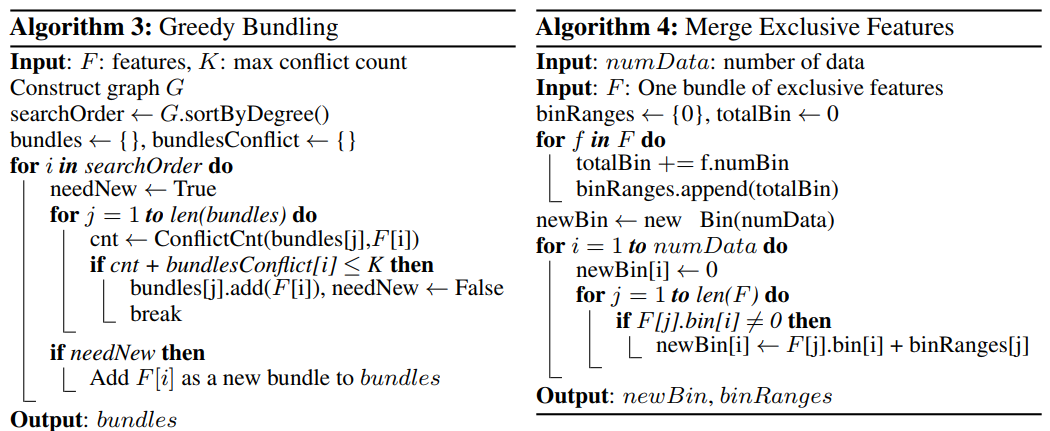
(αa /√β)> (1− a) /(√β – a) với αa = maxxi∈A∪Ac | gi | / maxxi∈Ac | gi |.

Tiếp theo, chúng tôi phân tích hiệu suất tổng quát hóa trong GOSS. Chúng tôi xem xét lỗi tổng quát trong GOSS E GOSS gen (d) = |V˜j (d) − V∗(d)|, đó là khoảng cách giữa chênh lệch phương sai được tính toán bởi các cá thể đào tạo được lấy mẫu trong GOSS và mức chênh lệch phương sai thực sự cho phân phối cơ bản. Chúng ta có E GOSS gen (d) ≤ |V˜j (d) − Vj (d)| + |Vj (d) − V∗(d)| ∆= EGOSS(d) + Egen(d)

Do đó, lỗi tổng quát với GOSS sẽ gần bằng cách tính toán bằng cách sử dụng các trường hợp dữ liệu đầy đủ nếu xấp xỉ GOSS là chính xác. Mặt khác, lấy mẫu sẽ làm tăng sự đa dạng của những người học cơ bản, có khả năng giúp cải thiện hiệu suất tổng quát hóa [24].

**4 Gói tính năng độc quyền**

Trong phần này, chúng tôi đề xuất một phương pháp mới để giảm hiệu quả số lượng các tính năng.



Dữ liệu chiều cao thường rất thưa thớt. Sự thưa thớt của không gian tính năng cung cấp cho chúng ta khả năng thiết kế một cách tiếp cận gần như mất mát để giảm số lượng các tính năng. Cụ thể, trong không gian tính năng thưa thớt, nhiều tính năng được loại trừ lẫn nhau, tức là, chúng không bao giờ lấy giá trị không đồng thời. Chúng tôi có thể gói các tính năng độc quyền một cách an toàn vào một tính năng duy nhất (mà chúng tôi gọi là một gói tính năng độc quyền). Bằng thuật toán quét tính năng được thiết kế cẩn thận, chúng tôi có thể tạo các biểu đồ tính năng giống nhau từ các gói tính năng như các từ tính năng riêng lẻ. Theo cách này, độ phức tạp của việc thay đổi biểu đồ histogram từ O (#data × #feature) thành O (#data × #bundle), trong khi #bundle << #feature. Sau đó, chúng tôi có thể tăng tốc đáng kể việc đào tạo GBDT mà không làm ảnh hưởng đến độ chính xác. Trong phần tiếp theo, chúng tôi sẽ chỉ ra cách làm thế nào để đạt được điều này một cách chi tiết. Có hai vấn đề cần giải quyết. Điều đầu tiên là xác định các tính năng nào nên được nhóm lại với nhau. Thứ hai là làm thế nào để xây dựng các gói.

**Định lý 4.1** Vấn đề về tính năng phân vùng thành một số gói nhỏ nhất là NP-hard. Bằng chứng: Chúng tôi sẽ giảm vấn đề tô màu đồ thị [25] cho vấn đề của chúng tôi. Vì vấn đề tô màu đồ thị là NP-hard, chúng ta có thể suy ra kết luận của mình.

Cho bất kỳ trường hợp nào G = (V, E) của vấn đề tô màu đồ thị. Chúng tôi xây dựng một ví dụ về vấn đề của chúng tôi như sau. Lấy từng hàng của ma trận tỷ lệ của G như một đối tượng địa lý, và lấy một ví dụ về vấn đề của chúng ta với | V | Tính năng, đặc điểm. Thật dễ dàng để thấy rằng một loạt các tính năng độc quyền trong vấn đề của chúng ta tương ứng với một tập các đỉnh có cùng màu và ngược lại.

Đối với vấn đề đầu tiên, chúng tôi chứng minh trong Định lý 4.1 rằng nó là NP-Hard để tìm chiến lược bó tối ưu, cho thấy rằng không thể tìm ra giải pháp chính xác trong thời gian đa thức. Để tìm ra thuật toán xấp xỉ tốt, trước tiên chúng tôi giảm vấn đề bó tối ưu xuống vấn đề tô màu đồ thị bằng cách lấy các đối tượng như đỉnh và thêm cạnh cho mỗi hai đối tượng nếu chúng không loại trừ lẫn nhau, sau đó chúng tôi sử dụng thuật toán tham lam có thể sản xuất hợp lý kết quả tốt

 (với một tỷ lệ xấp xỉ liên tục) cho màu đồ thị để sản xuất các bó. Hơn nữa, chúng tôi nhận thấy rằng thường có một vài tính năng, mặc dù không phải là 100% loại trừ lẫn nhau, cũng hiếm khi có các giá trị không đồng thời. Nếu thuật toán của chúng tôi có thể cho phép một phần nhỏ các xung đột, chúng tôi có thể nhận được một số gói tính năng thậm chí còn nhỏ hơn và cải thiện hơn nữa hiệu quả tính toán. Bằng cách tính toán đơn giản, ngẫu nhiên gây ô nhiễm một phần nhỏ các giá trị tính năng sẽ ảnh hưởng đến độ chính xác của đào tạo tối đa O ([(1 - γ) n] −2/3) (xem Đề xuất 2.1 trong tài liệu bổ sung), trong đó γ là cực đại tỷ lệ xung đột trong mỗi gói. Vì vậy, nếu chúng ta chọn một relatively tương đối nhỏ, chúng ta sẽ có thể đạt được sự cân bằng tốt giữa độ chính xác và hiệu quả.

Dựa trên các cuộc thảo luận ở trên, chúng tôi thiết kế một thuật toán cho gói tính năng độc quyền như được hiển thị trong Alg. 3. Đầu tiên, chúng tôi xây dựng biểu đồ có các cạnh có trọng số, có trọng số tương ứng với tổng số xung đột giữa các đối tượng địa lý. Thứ hai, chúng tôi sắp xếp các tính năng theo độ của chúng trong biểu đồ theo thứ tự giảm dần. Cuối cùng, chúng tôi kiểm tra từng tính năng trong danh sách có thứ tự và gán nó cho một gói hiện có với một xung đột nhỏ (được kiểm soát bởi γ) hoặc tạo một gói mới. Độ phức tạp thời gian của Alg. 3 là O (# feature2) và nó được xử lý chỉ một lần trước khi đào tạo. Độ phức tạp này có thể chấp nhận được khi số lượng đối tượng địa lý không lớn lắm nhưng vẫn có thể bị ảnh hưởng nếu có hàng triệu đối tượng địa lý. Để cải thiện hiệu quả hơn, chúng tôi đề xuất chiến lược đặt hàng hiệu quả hơn mà không cần xây dựng biểu đồ: đặt hàng bằng số giá trị không đồng bộ, tương tự như thứ tự theo độ vì nhiều giá trị khác không thường dẫn đến xác suất xung đột cao hơn. Vì chúng tôi chỉ thay đổi các chiến lược đặt hàng trong Alg. 3, các chi tiết của thuật toán mới được bỏ qua để tránh trùng lặp.

Đối với các vấn đề thứ hai, chúng ta cần một cách hợp nhất các tính năng trong cùng một gói để giảm độ phức tạp của đào tạo tương ứng. Điều quan trọng là đảm bảo rằng các giá trị của các tính năng ban đầu có thể được xác định từ các gói tính năng. Vì thuật toán dựa trên biểu đồ lưu trữ các thùng riêng biệt thay vì các giá trị liên tục của các tính năng, chúng ta có thể xây dựng một gói tính năng bằng cách cho phép các tính năng độc quyền nằm trong các thùng khác nhau. Điều này có thể được thực hiện bằng cách thêm bù đắp cho các giá trị ban đầu của các tính năng. Ví dụ: giả sử chúng tôi có hai tính năng trong một gói tính năng. Ban đầu, đối tượng A lấy giá trị từ [0, 10] và đối tượng B lấy giá trị [0, 20). Sau đó, chúng tôi thêm giá trị bù trừ là 10 vào các giá trị của đối tượng địa lý B sao cho đối tượng địa lý được tinh chỉnh lấy giá trị từ [10, 30]. Sau đó, có thể hợp nhất các tính năng A và B, và sử dụng một gói tính năng có dải ô [0, 30] để thay thế các tính năng gốc A và B. Thuật toán chi tiết được hiển thị trong Alg. 4.

Thuật toán EFB có thể kết hợp nhiều tính năng độc quyền với ít tính năng dày đặc hơn nhiều, có thể tránh được việc tính toán không cần thiết đối với các giá trị tính năng bằng không. Trên thực tế, chúng tôi cũng có thể tối ưu hóa thuật toán dựa trên biểu đồ cơ bản để bỏ qua các giá trị tính năng bằng không bằng cách sử dụng bảng cho mỗi đối tượng để ghi lại dữ liệu với các giá trị khác. Bằng cách quét dữ liệu trong bảng này, chi phí xây dựng biểu đồ cho một đối tượng địa lý sẽ thay đổi từ O (#data) thành O (#non\_zero\_data). Tuy nhiên, phương pháp này cần thêm bộ nhớ và chi phí tính toán để duy trì các bảng tính năng này trong toàn bộ quá trình tăng trưởng của cây. Chúng tôi thực hiện tối ưu hóa này trong LightGBM như một chức năng cơ bản. Lưu ý, tối ưu hóa này không xung đột với EFB vì chúng tôi vẫn có thể sử dụng nó khi các gói bị thưa thớt.

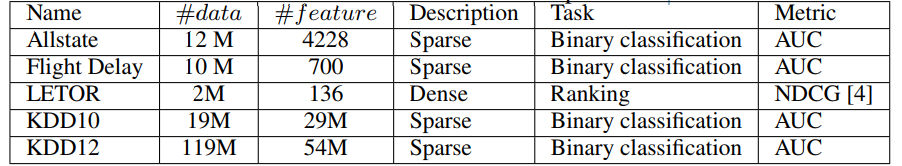
**5 thử nghiệm**

Trong phần này, chúng tôi báo cáo kết quả thử nghiệm về thuật toán LightGBM được đề xuất của chúng tôi. Chúng tôi sử dụng năm bộ dữ liệu khác nhau, tất cả đều có sẵn công khai. Chi tiết về các bộ dữ liệu này được liệt kê trong Bảng 1. Trong số đó, bộ dữ liệu Microsoft Learning to Rank (LETOR) [26] chứa 30 nghìn truy vấn tìm kiếm trên web. Các tính năng được sử dụng trong tập dữ liệu này chủ yếu là các tính năng số lớn. Bộ dữ liệu Allstate Insurance Claim [27] và Flight Delay [28] đều chứa rất nhiều tính năng mã hóa nóng một lần. Và hai tập dữ liệu cuối cùng là từ KDD CUP 2010 và KDD CUP 2012. Chúng tôi trực tiếp sử dụng các tính năng được sử dụng bởi giải pháp chiến thắng từ NTU [29, 30, 31], chứa cả hai đặc điểm dày đặc và thưa thớt, và hai bộ dữ liệu này rất lớn . Các bộ dữ liệu này lớn, bao gồm cả các tính năng thưa thớt và dày đặc và bao gồm nhiều tác vụ trong thế giới thực. Vì vậy, chúng tôi có thể sử dụng chúng để kiểm tra kỹ thuật toán của chúng tôi. Môi trường thử nghiệm của chúng tôi là một máy chủ Linux với hai CPU E5-2670 v3 (trong tổng số 24 lõi) và bộ nhớ 256GB. Tất cả các thử nghiệm chạy với đa luồng và số lượng chuỗi được cố định đến 16.

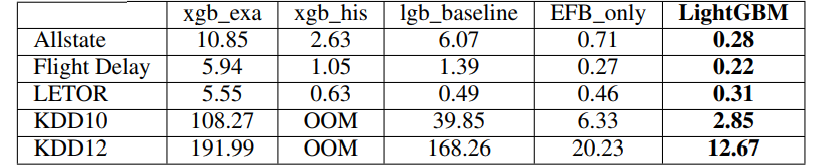
**5.1 So sánh tổng thể**

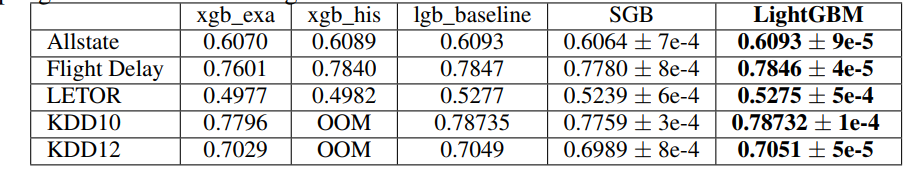
Chúng tôi trình bày các so sánh tổng thể trong tiểu mục này. XGBoost [13] và LightGBM không có GOSS và EFB (được gọi là lgb\_baselline) được sử dụng làm đường cơ sở. Đối với XGBoost, chúng tôi đã sử dụng hai phiên bản, xgb\_exa (thuật toán được sắp xếp trước) và xgb\_his (thuật toán dựa trên biểu đồ). Đối với xgb\_his, lgb\_baseline và LightGBM, chúng tôi đã sử dụng chiến lược tăng trưởng cây xanh lá cây [32]. Đối với xgb\_exa, vì nó chỉ hỗ trợ chiến lược tăng trưởng lớp khôn ngoan, chúng tôi đã điều chỉnh các tham số cho xgb\_exa để cho nó phát triển các cây tương tự như

Bảng 1: Các tập dữ liệu được sử dụng trong các thử nghiệm

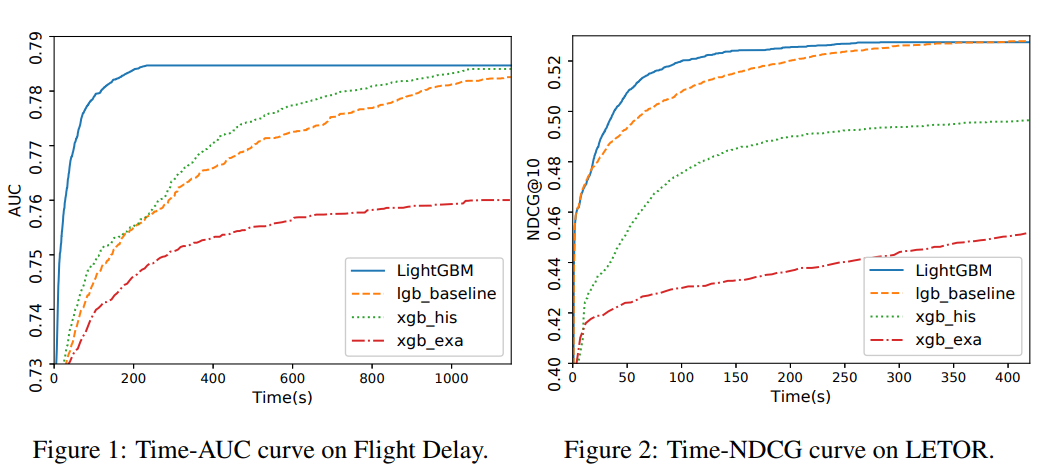


Bảng 2: So sánh tổng chi phí đào tạo tổng thể. LightGBM là lgb\_baseline với GOSS và EFB. EFB\_only là lgb\_baseline với EFB. Các giá trị trong bảng là chi phí thời gian trung bình (giây) để huấn luyện một lần lặp.

Bảng 3: So sánh độ chính xác tổng thể trên các bộ dữ liệu thử nghiệm. Sử dụng AUC cho nhiệm vụ phân loại và NDCG @ 10 để xếp hạng nhiệm vụ. SGB là lgb\_baseline với Stochastic Gradient Boosting, và tỷ lệ lấy mẫu của nó giống với LightGBM

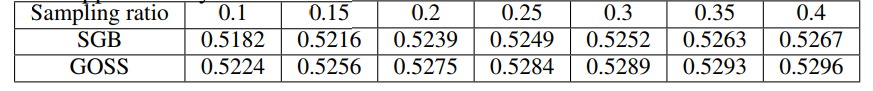


phương pháp. Và chúng tôi cũng điều chỉnh các thông số cho tất cả các tập dữ liệu theo hướng cân bằng tốt hơn giữa tốc độ và độ chính xác. Chúng tôi thiết lập một = 0,05, b = 0,05 cho Allstate, KDD10 và KDD12, và thiết lập một = 0,1, b = 0,1 cho Delay chuyến bay và LETOR. Chúng tôi đặt γ = 0 trong EFB. Tất cả các thuật toán được chạy cho các lần lặp cố định và chúng tôi nhận được kết quả chính xác từ lần lặp lại với điểm số tốt nhất.6



Thời gian đào tạo và độ chính xác kiểm tra được tóm tắt trong Bảng 2 và Bảng 3 tương ứng. Từ những kết quả này, chúng ta có thể thấy rằng LightGBM là nhanh nhất trong khi vẫn duy trì độ chính xác gần giống như đường cơ sở. Xgb\_exa dựa trên thuật toán được sắp xếp trước, nó khá chậm so với các thuật toán cơ sở histogram. Bằng cách so sánh với lgb\_baseline, LightGBM tăng tốc lần lượt là 21x, 6x, 1.6x, 14x và 13x trên bộ dữ liệu Allstate, Flight Delay, LETOR, KDD10 và KDD12. Vì xgb\_his tiêu thụ khá nhiều bộ nhớ, nó không thể chạy thành công trên bộ dữ liệu KDD10 và KDD12 do bộ nhớ ngoài. Trên các bộ dữ liệu còn lại, LightGBM là tất cả nhanh hơn, lên đến 9x tốc độ đạt được trên bộ dữ liệu Allstate. Tốc độ được tính dựa trên thời gian đào tạo cho mỗi lần lặp vì tất cả các thuật toán hội tụ sau số lần lặp tương tự nhau. Để chứng minh quá trình đào tạo tổng thể, chúng tôi cũng hiển thị các đường cong đào tạo dựa trên thời gian đồng hồ treo tường trên Delay chuyến bay và LETOR trong hình. 1

Bảng 4: So sánh chính xác về số liệu LETOR cho GOSS và SGB theo các tỷ lệ lấy mẫu khác nhau. Chúng tôi đảm bảo tất cả các thử nghiệm đều đạt đến điểm hội tụ bằng cách sử dụng các lần lặp lớn với điểm dừng sớm. Độ lệch chuẩn trên các cài đặt khác nhau là nhỏ. Các thiết lập của a và b cho GOSS có thể được tìm thấy trong các tài liệu bổ sung.



và Hình 2, tương ứng. Để tiết kiệm không gian, chúng tôi đặt các đường cong đào tạo còn lại của các bộ dữ liệu khác trong tài liệu bổ sung.

Trên tất cả các bộ dữ liệu, LightGBM có thể đạt được độ chính xác kiểm tra gần giống như các đường cơ sở. Điều này cho thấy cả GOSS và EFB sẽ không làm tổn thương độ chính xác trong khi mang lại tốc độ đáng kể. Nó phù hợp với phân tích lý thuyết của chúng tôi ở Sec. 3.2 và Sec. 4. LightGBM đạt được tỷ lệ tăng tốc khá khác nhau trên các bộ dữ liệu này. Tốc độ tăng tổng thể đến từ sự kết hợp giữa GOSS và EFB, chúng tôi sẽ phá vỡ sự đóng góp và thảo luận về hiệu quả của GOSS và EFB một cách riêng biệt trong các phần tiếp theo.

**5.2 Phân tích về GOSS**

Đầu tiên, chúng tôi nghiên cứu khả năng tăng tốc của GOSS. Từ so sánh LightGBM và EFB\_only (LightGBM không có GOSS) trong Bảng 2, chúng ta có thể thấy rằng GOSS có thể mang lại tốc độ gần gấp 2 lần bằng cách sử dụng dữ liệu 10% - 20%. GOSS có thể học cây bằng cách chỉ sử dụng dữ liệu được lấy mẫu. Tuy nhiên, nó vẫn giữ lại một số tính toán trên tập dữ liệu đầy đủ, chẳng hạn như tiến hành các dự đoán và tính toán các gradient. Do đó, chúng ta có thể thấy rằng tốc độ tổng thể không tương quan tuyến tính với tỷ lệ dữ liệu được lấy mẫu. Tuy nhiên, tốc độ tăng lên của GOSS vẫn rất quan trọng và kỹ thuật này được áp dụng rộng rãi cho các bộ dữ liệu khác nhau.

Thứ hai, chúng tôi đánh giá tính chính xác của GOSS bằng cách so sánh với Stochastic Gradient Boosting (SGB) [20]. Không mất tính tổng quát, chúng tôi sử dụng tập dữ liệu LETOR cho thử nghiệm. Chúng tôi điều chỉnh tỷ lệ lấy mẫu bằng cách chọn a và b khác nhau trong GOSS và sử dụng cùng tỷ lệ lấy mẫu tổng thể cho SGB. Chúng tôi chạy các cài đặt này cho đến khi hội tụ bằng cách dừng sớm. Các kết quả được thể hiện trong Bảng 4. Chúng ta có thể thấy độ chính xác của GOSS luôn tốt hơn SGB khi sử dụng cùng một tỷ lệ lấy mẫu. Những kết quả này phù hợp với các cuộc thảo luận của chúng tôi ở Sec. 3.2. Tất cả các thí nghiệm cho thấy rằng GOSS là một phương pháp lấy mẫu hiệu quả hơn so với lấy mẫu ngẫu nhiên.

**5.3 Phân tích về EFB**

Chúng tôi kiểm tra sự đóng góp của EFB để tăng tốc bằng cách so sánh lgb\_baseline với EFB\_only. Các kết quả được thể hiện trong Bảng 2. Ở đây chúng tôi không cho phép xung đột trong quá trình tìm kiếm bó (tức là, γ = 0) .7 Chúng tôi nhận thấy rằng EFB có thể giúp tăng tốc đáng kể trên các bộ dữ liệu quy mô lớn này.

Xin lưu ý lgb\_baseline đã được tối ưu hóa cho các tính năng thưa thớt, và EFB vẫn có thể tăng tốc độ đào tạo theo một yếu tố lớn. Đó là bởi vì EFB kết hợp nhiều tính năng thưa thớt (cả tính năng mã hóa một nóng và các tính năng độc quyền hoàn toàn) vào các tính năng ít hơn nhiều. Tối ưu hóa tính năng thưa thớt cơ bản được bao gồm trong quá trình đóng gói. Tuy nhiên, EFB không có chi phí bổ sung về việc duy trì bảng dữ liệu không phải cho mỗi tính năng trong quá trình học tập cây. Hơn thế nữa, vì nhiều tính năng được tách biệt trước đó được nhóm lại với nhau, nó có thể tăng vị trí không gian và cải thiện đáng kể tỷ lệ truy cập bộ nhớ cache. Do đó, sự cải thiện tổng thể về hiệu quả là đáng kể. Với phân tích trên, EFB là một thuật toán rất hiệu quả để tận dụng tài sản thưa thớt trong thuật toán dựa trên biểu đồ, và nó có thể mang lại một tốc độ đáng kể cho quá trình đào tạo GBDT.

**6 Kết luận**

Trong bài báo này, chúng tôi đã đề xuất một thuật toán GBDT mới gọi là LightGBM, chứa hai kỹ thuật mới: Lấy mẫu một bên dựa trên Gradient và Gói tính năng độc quyền để xử lý số lượng lớn các phiên bản dữ liệu và số lượng lớn các tính năng tương ứng. Chúng tôi đã thực hiện cả hai nghiên cứu phân tích lý thuyết và thực nghiệm về hai kỹ thuật này. Kết quả thử nghiệm phù hợp với lý thuyết và cho thấy rằng với sự trợ giúp của GOSS và EFB, LightGBM có thể vượt trội hơn đáng kể so với tốc độ tính toán và tiêu thụ bộ nhớ của XGBoost và SGB. Đối với công việc trong tương lai, chúng tôi sẽ nghiên cứu lựa chọn tối ưu của a và b trong Lấy mẫu một bên dựa trên Gradient và tiếp tục cải thiện hiệu suất của tính năng Gói độc quyền để xử lý nhiều tính năng bất kể chúng có thưa thớt hay không.