Xgboost

Pre-process:

B1: Visualize dữ liệu để xem xét.

B2: Những cột có dạng object(có chứa chữ) sẽ drop ,sử dụng one-hot encode để chuyển thành giá trị 0 hoặc 1 và đưa thành bảng chân trị với 0 là giá trị sai và 1 là giá trị đúng

B3: Vì tập kết quả giá xe là tập giá trị liên tục nên ta sẽ sử dụng mô hìnhregression để dự đoán. Vì vậy, sử dụng standard scaler sẽ phù hợp hơn cho các mô hình hồi quy. VÌ khi sử dụng Standard scaler, ta có thể căn giữa các feature column tại giá trị 0 với độ lệch chuẩn là 1. Từ đó các giá trị có phân phối chuẩn và các trọng số có thể dễ dàng học hơn

XGBoost là viết tắt của Extreme Gradient Boosting. Đây là thuật toán state-of-the-art nhằm giải quyết bài toán supervised learning cho độ chính xác khá cao bên cạnh mô hình Deep learning.

XGBoost là phiên bản cải tiến của Gradient Boosting. Ưu điểm vượt trội của nó được chứng minh ở các khía cạnh:

* Tốc độ xử lý
  + XGBoost thực hiện tinh toán song song nên tốc độ xử lý có thể tăng gấp 10 lần so với GBM. Ngoài ra, XGboost còn hỗ trợ tính toán trên Hadoop.
* Overfitting
  + XGBoost áp dụng cơ chế Regularization nên hạn chế đáng kể hiệ tượng Overfitting (GBM không có regularization).
* Sự linh hoạt
  + XGboost cho phép người dùng sử dụng hàm tối ưu và chỉ tiêu đánh giá của riêng họ, không hạn chế ở những hàm cung cấp sẵn.
* Xử lý missing value
  + XGBoost bao gồm cơ chế tự động xử lý missing value bên trong nó. Vì thế, có thể bỏ qua bước này khi chuẩn bị dữ liệu cho XGBoost.
* Tự động cắt tỉa
  + Tính năng tree pruning hộ trợ việc tự động bỏ qua những leaves, nodes không mang giá trị tích cực trong quá trình mở rộng tree.

Chính vì những ưu điểm đó mà hiệu năng của XGBoost tăng lên đáng kể so với các thuật toán ensemble learning khác. Nó được sử dụng ở hầu hết các cuộc thi trên Kaggle cũng như Hackathons.

Cụ thể:

Boosting: **Ý tưởng:** thay vì xây dựng một mô hình dự đoán (chẳng hạn descision tree) có độ chính xác tương đối, ta đi xây dựng nhiều mô hình dự đoán có độ chính xác kém hơn (weak learner) khi đi riêng lẻ nhưng lại cho độ chính xác cao khi kết hợp lại.

Ta có thể hình dung mỗi weak learner gồm học sinh yếu, khá, giỏi và thầy giáo. Trong đó, trọng số uy tín về kiến thức của thầy giáo sẽ là cao nhất và học sinh yếu sẽ là thấp nhất. Khi bạn đặt câu hỏi nào đó và cần những người này đưa ra kết luận, nếu nhiều người cùng có chung kết luận hoặc uy tín của những người đưa ra kết luận cao hơn tập thể thì ta có thể tin kết luận này là đúng.

Ví dụ trong thuật toán [AdaBoost](https://ongxuanhong.wordpress.com/2015/09/22/adaboost-hoi-gi-dap-nay/), mỗi lần huấn luyện weak learner, mô hình sẽ tính lại trọng số cho các điểm dữ liệu đã bị phân lớp sai, để những lượt huấn luyện tiếp theo những điểm dữ liệu này sẽ có cơ hội nhiều hơn được phân lớp đúng. Dưới đây là mô hình dự đoán tổng quát:

H(\mathbf{x}) = sign(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x}) + ... + \alpha_k h_k(\mathbf{x}))

Gradient boosting: Cả AdaBoost và Gradient Boosting đều xây dựng thuật toán nhằm giải quyết bài toán tối ưu sau :

min*cn*​=1:*N*,*wn*​=1:*N*​*L*(*y*,*n*=1∑*N*​*cn*​*wn*​))

Trong đó :

* *L* : giá trị loss function
* *y* : label
* *cn*​ : confidence score của weak learner thứ n (hay còn gọi là trọng số)
* *wn*​ : weak learner thứ n

Thoạt nhìn, công thức trên có vẻ khá giống với Bagging, thế nhưng cách tính ra các giá trị confidence score kia lại làm nên sự khác biệt về hướng giải quyết của Boosting. Thay vì cố gằng quét tìm tất cả các giá trị c\_n, w\_n*cn*​,*wn*​ để tìm nghiệm tối ưu toàn cục - một công việc tốn nhiều thời gian và tài nguyên, chúng ta sẽ cố gắng tìm các giá trị nghiệm cục bộ sau khi thêm mỗi một mô hình mới vào chuỗi mô hình với mong muốn dần đi đến nghiệm toàn cục.

min*cn*​,*wn*​​*L*(*y*,*Wn*−1​+*cn*​*wn*​))

với  *Wn*−1​=∑*n*=1*N*−1​*cn*​*wn*​

vấn đề tối ưu:

min*cn*​,*wn*​​*L*(*y*,*Wn*−1​+*cn*​*wn*​))

nhắc lại về gradient decent:

*θn*​=*θn*−1​−*η*∂*θ*∂​*L*(*θn*−1​)

Phía trên là công thức cập nhật tham số mô hình theo hướng giảm của đạo hàm (Gradient Descent). Công thức này được sử dụng không gian tham số, tuy nhiên, để liên hệ với bài toán chúng ta đang xét, mình chuyển công thức sang góc nhìn của không gian hàm số.

Khá đơn giản thôi, nếu chúng ta coi chuỗi các model boosting là một hàm số *W*, thì mỗi hàm learner có thể coi là một tham số *w*. Đến đây, để cực tiểu hóa hàm loss  *L*(*y*,*W*), chúng ta áp dụng Gradient Descent

*Wn*​=*Wn*−1​−*η*∂*w*∂​*L*(*Wn*−1​)

Đến đây, ta có thể thấy mối quan hệ liên quan sau

*cn*​*wn*​≈−*η*∂*w*∂​*L*(*Wn*−1​)

với *wn*​ là model được thêm vào tiếp theo. Khi đó, model mới cần học để fit để vào giá trị  -*η*∂*w*∂​*L*(*Wn*−1​)

XGBoost:

XGBoost và Gradient boosting đều dựa trên cùng ý tưởng đó là boosting thông qua gradient descent trong không gian hàm số. Tuy nhiên, điều làm nên hiệu suất ấn tượng và khả năng tính toán của XGBoost nằm ở ba yếu tố:

* Engineering để tránh overfitting như: sub-sampling row, column, column per split levels, áp dụng regularized L1 và L2.
* Khả năng tận dụng tài nguyên hệ thống: tính toán song song trên CPU/GPU, tính toán phân tán trên nhiều server, tính toán khi tài nguyên bị giới hạn, cache optimization để tăng tốc training.
* Và cuối cùng là khả năng xử lý missing data value, tiếp tục training bằng mô hình đã được build trước đó để tiết kiệm thời gian.

Sử dụng gridsearchCV:

Eta: 0.1

Max\_depth: 6

N\_estimators: 400

eta: Tương tự learning\_rate

max\_depth: Độ sâu tối đa của decision tree

n\_estimators: số lượng các cây sử dụng trong model

|  |  |  |  |
| --- | --- | --- | --- |
|  | R2 | MAE | MSE |
| Original | 0.9079843438287456 | 3142.4137067769834 | 24488849.408993956 |
| Tuned | 0.9595187869082665 | 1926.4253627916905 | 10773583.241658695 |