ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

Nhóm 3

XGBOOST TRÊN LOAN APPROVAL CLASSIFICATION DATASET

Bài tập lớn Machine Learning
Lớp: 2425I_INT3405E_59

HÀ NỘI - 2024

MỤC LỤC

1. Thành viên

1. Phạm Ngọc Hải Dương - 23021513
2. Phạm Mai Anh - 23021469
3. Bùi Thế Kiệt – 23021592

2. Quá trình

- Tham khảo, đánh giá và chọn thuật toán.
- Test thuật toán và mô hình bằng thư viện.
- Code lại thuật toán và mô hình (không dùng thư viện)
- Train và test lại, chỉnh sửa và đánh giá hiệu quả.

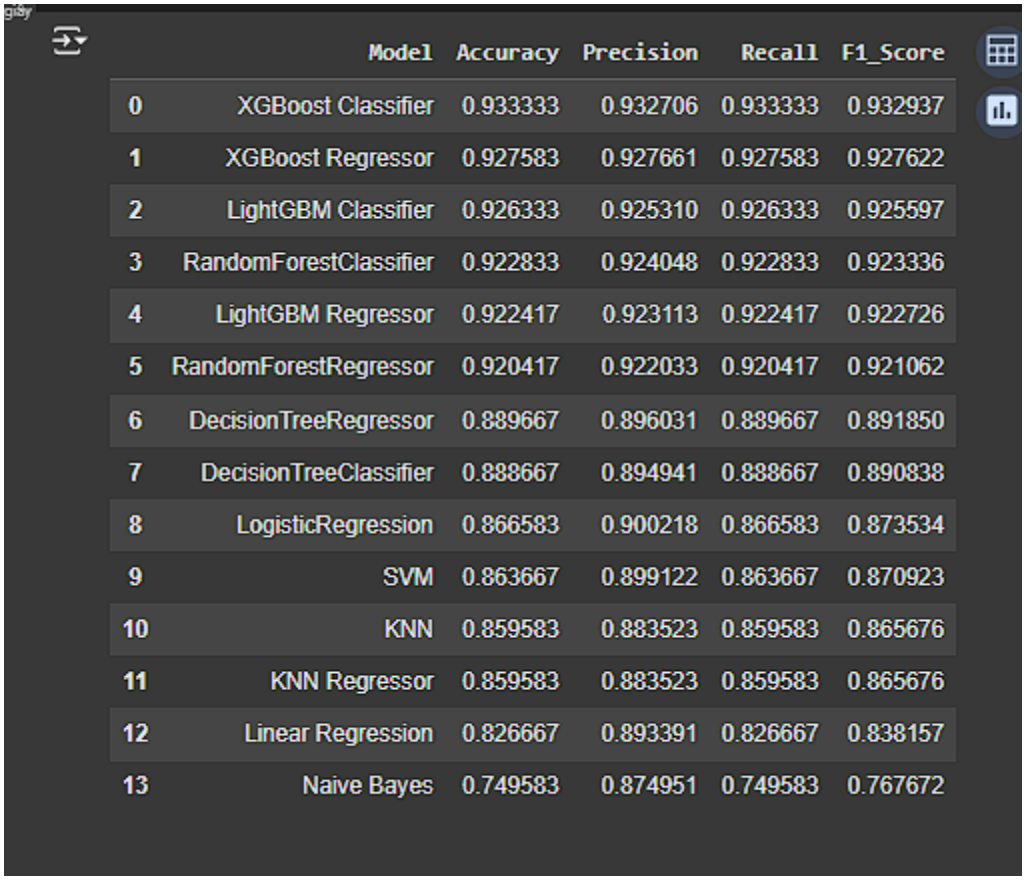
3. Chi tiết quá trình

3.1. Tham khảo, đánh giá và chọn thuật toán:

- Tham khảo qua tập dữ liệu Loan Approval Classification Dataset từ https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data/code, các model có trước như Lazy predict (https://www.kaggle.com/code/aruneembhowmick/lazypredict-for-loan-approval-classification),...
- Ưu tiên chọn các thuật toán có accuracy >= 90% như Decision Tree, XGB Classifier, Random Forest, …

3.2. Test thuật toán và mô hình bằng thư viện:

- Đánh giá, làm sạch, xử lý dữ liệu trước khi train model bằng thư viện
- Đánh giá hiệu quả thuật toán, model.



	Model	Accuracy	Precision	Recall	F1_Score
0	XGBoost Classifier	0.933333	0.932706	0.933333	0.932937
1	XGBoost Regressor	0.927583	0.927661	0.927583	0.927622
2	LightGBM Classifier	0.926333	0.925310	0.926333	0.925597
3	RandomForestClassifier	0.922833	0.924048	0.922833	0.923336
4	LightGBM Regressor	0.922417	0.923113	0.922417	0.922726
5	RandomForestRegressor	0.920417	0.922033	0.920417	0.921062
6	DecisionTreeRegressor	0.889667	0.896031	0.889667	0.891850
7	DecisionTreeClassifier	0.888667	0.894941	0.888667	0.890838
8	LogisticRegression	0.866583	0.900218	0.866583	0.873534
9	SVM	0.863667	0.899122	0.863667	0.870923
10	KNN	0.859583	0.883523	0.859583	0.865676
11	KNN Regressor	0.859583	0.883523	0.859583	0.865676
12	Linear Regression	0.826667	0.893391	0.826667	0.838157
13	Naive Bayes	0.749583	0.874951	0.749583	0.767672

3.3. Code lại thuật toán và mô hình (không dùng thư viện)

3.4. Train và test lại, chỉnh sửa và đánh giá hiệu quả.

- Điều chỉnh tham số để được hiệu quả cao nhất. (93.53% cao nhất)
- Hiệu quả dao động quanh 93.1% - 93.5%

Thứ tự	Accuracy	F1 Score	Precision	Recall
1	92.43%	0.8213	0.953	0.7216
2	93.28%	0.8499	0.9212	0.7887
3	93.12%	0.8461	0.9168	0.7842
4	93.08%	0.8455	0.9147	0.786
5	93.08%	0.8429	0.931	0.7701
6	93.19%	0.8442	0.9409	0.7656
7	93.37%	0.8504	0.9316	0.7822
8	93.23%	0.8446	0.946	0.7628
9	93.24%	0.8458	0.9396	0.769
10	93.27%	0.8484	0.9288	0.7808
11	93.41%	0.8533	0.9204	0.7953
12	93.33%	0.8493	0.9326	0.7797
13	93.21%	0.8445	0.9379	0.768
14	93.18%	0.8481	0.9199	0.7867
15	93.33%	0.8502	0.9269	0.7853
16	93.29%	0.8501	0.9209	0.7894
17	93.21%	0.8409	0.9146	0.7922
18	93.33%	0.8508	0.9224	0.7894
19	93.28%	0.8486	0.9289	0.7811
20	93.24%	0.8484	0.9235	0.7846
21	93.16%	0.8466	0.9208	0.7835
22	93.28%	0.8492	0.9254	0.7846
23	93.47%	0.8533	0.9295	0.7887
24	93.29%	0.8508	0.9166	0.7939
25	93.29%	0.8497	0.9233	0.787
26	93.13%	0.8451	0.9262	0.777
27	93.53%	0.8554	0.9262	0.7946
28	93.17%	0.8463	0.9257	0.7794
29	93.27%	0.8486	0.9278	0.7818
30	93.36%	0.8517	0.9219	0.7915
31	93.13%	0.8454	0.9241	0.779
32	93.39%	0.8516	0.9282	0.7867
33	93.32%	0.849	0.9322	0.7794
34	93.03%	0.8449	0.9115	0.7873
35	93.12%	0.8461	0.9186	0.7842
36	93.20%	0.8483	0.9175	0.7887
37	93.25%	0.8487	0.9228	0.7856
38	93.36%	0.8516	0.9229	0.7905
39	93.42%	0.8515	0.9343	0.7822
40	93.37%	0.8542	0.9192	0.7946

4. Báo cáo thuật toán XG BOOST và code model

4.1. Tổng quan:

- XGBoost là một thư viện tối ưu hóa phân tán cho thuật toán gradient boosting, được thiết kế để hiệu quả, linh hoạt và có thể chuyển giao dễ dàng. Nó triển khai các thuật toán học máy dưới khuôn khổ Gradient Boosting.

- Mô hình tái tạo một phần mô hình XGBoost trong thư viện. Dựa trên bài báo về XGBoost: https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf

- Mô hình không thể tái sử dụng tất cả các chức năng của bộ phân loại XGBoost trong thư viện xgb. Mô hình này sử dụng tree_method = 'exact'.

- Mặc dù không sao chép hoàn toàn tất cả các chức năng của thư viện XGBoost thực tế, mô hình này thể hiện các hoạt động cốt lõi của mô hình với các thành phần chính của XGBoost.

4.2. Chú thích:

4.2.1. Gradient Boosting:

Gradient boosting là một kỹ thuật học máy dựa trên phương pháp boosting trong không gian hàm số, trong đó mục tiêu là các pseudo-residual (phần dư giả) thay vì residual (phần dư) như trong boosting truyền thống. Nó tạo ra một mô hình dự đoán dưới dạng một tập hợp các mô hình dự đoán yếu, tức là các mô hình mà ít giả định về dữ liệu, thường là các cây quyết định đơn giản.

Thuật toán này xây dựng một mô hình cộng dồn theo từng giai đoạn, cho phép tối ưu hóa các hàm mất mát có thể phân biệt tùy ý. Trong mỗi giai đoạn, các cây hồi quy n_classes_ được điều chỉnh trên gradient âm của hàm mất mát, ví dụ như log loss nhị phân hoặc đa lớp. Phân loại nhị phân là một trường hợp đặc biệt, nơi chỉ có một cây hồi quy duy nhất được tạo ra.

4.2.2. Parameter:

- subsample: Tỷ lệ dữ liệu được sử dụng trong mỗi lần huấn luyện.

- learning_rate: Kích thước bước để điều chỉnh dự đoán trong mỗi lần lặp.

- base_score: Giá trị dự đoán ban đầu trước khi huấn luyện.

- max_depth: Độ sâu tối đa của mỗi cây quyết định.

- random_seed: Hạt giống cho việc tạo số ngẫu nhiên nhằm đảm bảo tính tái lập.

4.2.3. Fit:

- Phương thức fit huấn luyện mô hình trên dữ liệu đầu vào X và nhãn mục tiêu y.
- Ban đầu, nó thiết lập dự đoán của mô hình thành base_score cho tất cả các ví dụ.
- Gradient và Hessian (đạo hàm bậc nhất và bậc hai của hàm mất mát) được tính toán cho mỗi cây.
- Nếu subsample không phải là 1.0, một tập con ngẫu nhiên của dữ liệu sẽ được chọn.
- Một đối tượng TreeBooster (không được cung cấp trong mã) được tạo ra cho mỗi cây, và dự đoán sẽ được cập nhật tương ứng.
- Các cây đã huấn luyện được lưu trữ trong danh sách TreeBoosters để sử dụng sau này trong việc dự đoán.

4.2.4. Dự đoán

- Phương thức predict tính toán dự đoán cho đầu vào X dựa trên các cây đã huấn luyện.
- Nó tính toán dự đoán cho mỗi cây trong danh sách TreeBoosters và cộng dồn chúng lại.
- Dự đoán cuối cùng được lấy bằng cách cộng thêm base score và điều chỉnh với learning rate.
- Sau đó, nó chuyển đổi kết quả liên tục thành các dự đoán nhị phân (0 hoặc 1) dựa trên ngưỡng 0.5.

4.3. Chi tiết TreeBooster Code

4.3.1. Đặc điểm:

- Xây dựng một cây với giới hạn độ sâu tối đa (max_depth).
- Tìm điểm chia tốt nhất (ngưỡng) cho mỗi đặc trưng để tối đa hóa "gain".
- Dừng quá trình chia khi gain không còn cải thiện nữa hoặc khi các tiêu chí dừng khác được đáp ứng.
- Chia dữ liệu đệ quy thành các nút con trái và phải cho đến khi cây đạt độ sâu tối đa hoặc các nút lá trở nên thuần nhất (đồng nhất).
- Dự đoán đầu ra dựa trên cấu trúc của cây.

4.3.2. Phương thức:

- **__init__**(self, X, g, h, params, max_depth, idxs=None) (Khởi tạo):

    - Khởi tạo đối tượng TreeBooster bằng cách nhận dữ liệu (X), gradient (g), Hessian (h), và các siêu tham số khác như max_depth, min_child_weight, reg_lambda, và gamma.

    - Bộ tạo (constructor) thiết lập các giá trị ban đầu, bao gồm giá trị dự đoán của nút gốc, và nếu max_depth lớn hơn không, nó sẽ tiếp tục chèn các nút con (tức là chia dữ liệu đệ quy thành các cây con).

- **_maybe_insert_child_nodes**(self) (Chia đệ quy):

    - Phương thức này cố gắng tìm đặc trưng và ngưỡng tốt nhất để chia dữ liệu.

    - Nó lặp qua tất cả các đặc trưng (feature_idx) và gọi phương thức _find_better_split() để đánh giá các điểm chia tiềm năng cho mỗi đặc trưng.

    - Nếu một điểm chia tốt được tìm thấy, dữ liệu sẽ được chia thành hai nhóm, và các nút con (cây con trái và phải) sẽ được tạo ra đệ quy.

- **is_leaf(self)** (Kiểm tra nút lá):

    - Thuộc tính này kiểm tra xem nút hiện tại có phải là nút lá hay không (tức là cây đã ngừng chia).

    - Nếu best split gain (best_score_so_far) bằng không, điều này chỉ ra rằng nút là nút lá và không thực hiện chia thêm.

- **_find_better_split**(self, feature_idx) (Tìm cách chia tốt nhất):

    - Phương thức này đánh giá tất cả các điểm chia tiềm năng cho một đặc trưng cụ thể (feature_idx) và tính toán gain cho mỗi điểm chia.

    - Phương thức sử dụng giá trị gradient và Hessian (g và h) để xác định điểm chia tốt nhất cho dữ liệu, dựa trên công thức từ bài báo XGBoost (phương trình 7).

    - Nó sắp xếp dữ liệu dựa trên giá trị của đặc trưng và kiểm tra từng cặp giá trị liên tiếp để tính toán gain cho việc chia.

    - Nếu gain được cải thiện, nó cập nhật điểm chia tốt nhất đã tìm thấy và thiết lập ngưỡng cho việc chia.

- **predict**(self, X) (Dự đoán cho đa dòng):

  - Phương thức này nhận dữ liệu đầu vào X và dự đoán đầu ra bằng cách lặp qua từng hàng và truyền nó qua cây một cách đệ quy.

  - Nó gọi phương thức _predict_row() cho từng hàng dữ liệu để nhận các dự đoán từ cây.

- **_predict_row**(self, row) (Dự đoán đơn dòng):

  - Phương thức đệ quy này dự đoán giá trị cho một hàng dữ liệu đơn lẻ.

  - Nếu nút hiện tại là nút lá, nó trả về giá trị được lưu trữ tại nút đó.

  - Nếu không, nó kiểm tra giá trị của đặc trưng tương ứng với điểm chia và chọn nút con trái hoặc phải để tiếp tục dự đoán.

4.3.3. Tham số quan trọng và ý tưởng:

- **min_child_weight:**

  - Đây là ngưỡng cho tổng tối thiểu của các giá trị Hessian (đạo hàm bậc hai) trong một nút con để tiếp tục chia. Nếu tổng này nhỏ hơn min_child_weight, không có sự chia thêm nào xảy ra tại nút đó.

- **reg_lambda:**

  - Tham số này là hạng mục điều chỉnh (regularization L2) để kiểm soát hiện tượng overfitting. Giá trị cao của reg_lambda giảm tác động của mỗi điểm chia, làm cho cây trở nên ít phức tạp hơn.

- **gamma:**

  - Tham số này điều khiển mức giảm thiểu mất mát tối thiểu cần thiết để thực hiện một phép chia (split) tiếp theo. Nếu lợi ích từ một phép chia tiềm năng nhỏ hơn giá trị gamma, phép chia đó sẽ không được thực hiện.

- **best_score_so_far:**

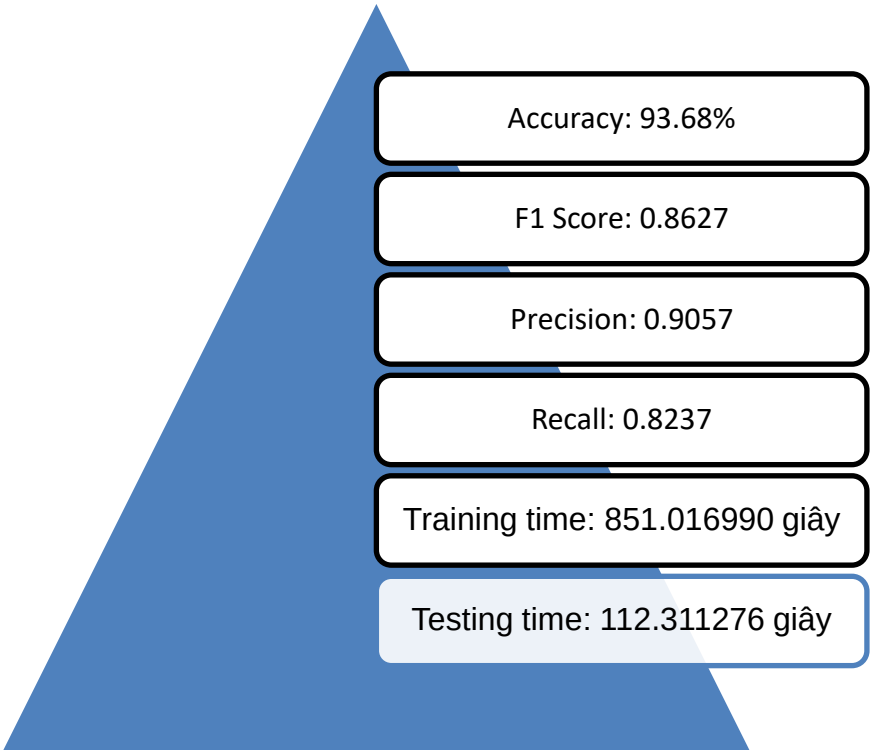
  - Biến này theo dõi gain tốt nhất được tìm thấy trong quá trình tìm kiếm điểm chia tối ưu. Nếu gain từ một điểm chia tốt hơn giá trị này, điểm chia đó sẽ được chọn.

- **threshold:**

  - Ngưỡng (threshold) là giá trị được sử dụng để phân tách dữ liệu cho một đặc trưng nhất định. Nó là điểm giữa giữa hai giá trị liên tiếp trong danh sách các giá trị của đặc trưng đã được sắp xếp.

5. Kết quả

Kết quả cuối cùng:



References

[1] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. https://doi.org/10.48550/arXiv.1603.02754