ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: Khai phá dữ liệu (CO3029)

Phân tích xác xuất vỡ nợ của khách vay tín dụng

Nhóm 4 - L01 - HK251

Giảng viên hướng dẫn : Đỗ Thành Thái
Sinh viên: Mai Đăng Dương - 2210612
Nguyễn Châu Hoài Phúc - 2212622
Nguyễn Bảo Phúc - 2212620

Thành phố Hồ Chí Minh, 9/2025

Mục lục

Chương 1

Tổng quan bài toán

Chương 2

Đặc tả bộ dữ liệu được sử dụng trong bài toán

2.1 Tổng quan bộ dữ liệu

Bộ dữ liệu *Home Credit Default Risk*[1] cung cấp hồ sơ khách hàng vay tiêu dùng và nhãn rủi ro vỡ nợ, nhằm hỗ trợ xây dựng mô hình dự đoán khả năng không trả nợ đúng hạn. Bộ dữ liệu chính gồm hai tệp:

- `application_train.csv`: 307511 quan sát, 122 biến (bao gồm biến mục tiêu `TARGET`).
- `application_test.csv`: 48744 quan sát, 121 biến (không có `TARGET`).

Trong `application_train`, tỷ lệ dương của `TARGET` (khách hàng *vỡ nợ*) là 8.073 (24825/307511), thể hiện hiện tượng *mất cân bằng lớp*. Tỷ lệ thiếu trung bình toàn bộ cột vào khoảng 24.396, trong đó một số biến căn hộ có tỷ lệ thiếu rất cao.

Biến mục tiêu: `TARGET` $\in \{0, 1\}$, với 1 = *vỡ nợ* và 0 = *không vỡ nợ*.

Kiểu biến: 106 biến số và 16 biến phân loại (object). Các nhóm biến chính:

- **Nhân khẩu/kinh tế:** `CODE_GENDER`, `CNT_CHILDREN`, `AMT_INCOME_TOTAL`, `NAME_EDUCATION_TYPE`, `NAME_INCOME_TYPE`, `NAME_FAMILY_STATUS`, `NAME_HOUSING_TYPE`, v.v.
- **Khoản vay:** `AMT_CREDIT`, `AMT_ANNUITY`, `AMT_GOODS_PRICE`, `NAME_CONTRACT_TYPE`.
- **Điểm/nguồn bên ngoài:** `EXT_SOURCE_1`, `EXT_SOURCE_2`, `EXT_SOURCE_3`.
- **Mốc thời gian tương đối (ngày):** `DAYS_*` như `DAYS_BIRTH`, `DAYS_EMPLOYED`, `DAYS_REGISTRATION` (giá trị *âm* biểu diễn số ngày trước thời điểm tham chiếu hiện tại).

Bộ dữ liệu Home Credit Default Risk là một điển hình trong các bài toán xếp hạng rủi ro tín dụng trong thực tế, nơi hiện tượng mất cân bằng lớp không phải là lỗi do việc thu thập mà là do bản chất của vấn đề. Trong tập dữ liệu này, các khoản vay "không vỡ nợ" chiếm tỷ lệ áp đảo lên đến khoảng 92%, trong khi các khoản vay "vỡ nợ" chỉ chiếm khoảng 8%. Sự chênh lệch lớn này phản ánh chính xác hiện tượng thực tế trong lĩnh vực tín dụng tiêu dùng: các quy trình thẩm định nghiêm ngặt đảm bảo rằng chỉ những khách hàng có rủi ro thấp mới được chấp thuận cho vay. Bên cạnh đó, điều kiện kinh tế thuận lợi tại thời điểm thu thập dữ liệu và thiên lệch trong việc lựa chọn mẫu (chỉ các hồ sơ đã được phê duyệt mới có mặt trong dataset) cũng góp phần tạo nên sự mất cân bằng này.

Hiện tượng mất cân bằng lớp dẫn đến những thách thức đáng kể trong việc đánh giá mô hình. Các chỉ số truyền thống như Accuracy trở nên không còn ý nghĩa khi một mô hình đơn giản chỉ cần luôn dự đoán "không vỡ nợ" đã có thể đạt độ chính xác lên đến 92%. Do đó, việc đánh giá hiệu suất thực sự của mô hình cần tập trung vào các chỉ số như ROC-AUC, PR-AUC, Recall và F1-score cho lớp thiểu số, đồng thời xem xét việc lựa chọn ngưỡng phân loại tối ưu phù hợp với mục tiêu kinh doanh cụ thể.

Một đặc điểm nổi bật khác của bộ dữ liệu này là tình trạng thiếu dữ liệu xuất hiện phổ biến với nhiều cơ chế khác nhau. Đối với nhóm thông tin nhân khẩu học và nghề nghiệp, các trường như loại hình công việc, nguồn thu nhập thường bị bỏ trống do không phải là thông tin bắt buộc trong quá trình đăng ký. Đối với dữ liệu từ các tổ chức

[1] Kaggle: `https://www.kaggle.com/competitions/home-credit-default-risk`

tín dụng bên ngoài, việc thiếu thông tin có thể do không truy vấn được hoặc nhà cung cấp dịch vụ không trả về kết quả. Đặc biệt, một số trường dữ liệu chứa các giá trị mã hóa đặc biệt, điển hình như DAYS_EMPLOYED với giá trị 365243, thực chất đại diện cho trạng thái "không xác định" thay vì một con số có ý nghĩa thực tế. Các biến tài chính quan trọng như AMT_ANNUITY và AMT_GOODS_PRICE cũng có tỷ lệ thiếu nhất định, gây ảnh hưởng đến việc tính toán các chỉ số tài chính then chốt.

Trong đánh giá mô hình, cần ưu tiên sử dụng các chỉ số như ROC-AUC và PR-AUC thay vì Accuracy, đồng thời bổ sung các chỉ số vận hành thực tế như Recall - tỷ lệ phát hiện đúng các khoản vay vỡ nợ trong nhóm k% hồ sơ có điểm rủi ro cao nhất. Cách tiếp cận này không chỉ đảm bảo đánh giá toàn diện hiệu suất mô hình mà còn bám sát vào quy trình sàng lọc và quản lý rủi ro trong thực tế kinh doanh của các tổ chức tín dụng.

2.1.1 Đặc điểm về bộ dữ liệu

Bộ dữ liệu được xây dựng từ thông tin hồ sơ tín dụng của khách hàng vay vốn tại một tổ chức tài chính. Mỗi dòng trong tập dữ liệu tương ứng với một khách hàng (hay một khoản vay), được mô tả bằng các đặc trưng cá nhân, tình trạng tài chính, tài sản sở hữu, thông tin hợp đồng và bối cảnh nộp hồ sơ. Toàn bộ các trường đầu vào đều là thông tin sẵn có tại thời điểm thẩm định nhằm tránh rò rỉ dữ liệu và phục vụ cho việc dự báo rủi ro sau này. Dữ liệu bao gồm các nhóm đặc trưng chính sau:

- **Thông tin định danh:** mã khách hàng SK_ID_CURR (chỉ dùng để phân biệt bản ghi, không đưa vào mô hình).
- **Thông tin cá nhân:** giới tính (CODE_GENDER), số con (CNT_CHILDREN), số thành viên gia đình (CNT_FAM_MEMBERS), tình trạng gia đình (NAME_FAMILY_STATUS), trình độ học vấn (NAME_EDUCATION_TYPE), loại hình thu nhập (NAME_INCOME_TYPE), loại hình cư trú/nhà ở (NAME_HOUSING_TYPE), nghề nghiệp (OCCUPATION_TYPE), loại hình tổ chức/cơ quan (ORGANIZATION_TYPE), nhóm đi cùng khi nộp hồ sơ (NAME_TYPE_SUITE).
- **Tài sản sở hữu:** cờ sở hữu ô tô (FLAG_OWN_CAR) và tuổi xe (OWN_CAR_AGE), cờ sở hữu nhà/đất (FLAG_OWN_REALTY), cùng các khối đặc trưng nhà ở/căn hộ được chuẩn hoá theo ba biến _AVG, _MEDI, _MODE như: APARTMENTS_*, LIVINGAREA_*, LANDAREA_*, BASEMENTAREA_*, YEARS_BUILD_*, YEARS_BEGINEXPLUATATION_*, COMMONAREA_*, ELEVATORS_*, ENTRANCES_*, FLOORSMAX_*, FLOORSMIN_*, LIVINGAPARTMENTS_*, NONLIVINGAPARTMENTS_*, NONLIVINGAREA_*.
- **Tài chính & thu nhập:** thu nhập hằng năm (AMT_INCOME_TOTAL), số tiền vay (AMT_CREDIT), giá trị hàng hoá/dịch vụ (AMT_GOODS_PRICE), niên kim trả góp (AMT_ANNUITY).
- **Điểm/nguồn rủi ro ngoại sinh:** các điểm tổng hợp từ nguồn ngoài (EXT_SOURCE_1, EXT_SOURCE_2, EXT_SOURCE_3) phản ánh rủi ro tín dụng tổng quát của khách hàng.
- **Bối cảnh & lịch sử hồ sơ theo thời gian:** bao gồm tuổi và thâm niên việc làm quy đổi theo ngày (DAYS_BIRTH, DAYS_EMPLOYED), thời điểm đăng ký hồ sơ (DAYS_REGISTRATION), thời điểm cấp/cập nhật giấy tờ (DAYS_ID_PUBLISH), thời điểm thay đổi số điện thoại (DAYS_LAST_PHONE_CHANGE), cùng bối cảnh xử lý hồ sơ (WEEKDAY_APPR_PROCESS_START, HOUR_APPR_PROCESS_START).
- **Đặc trưng vùng/khu vực & tính nhất quán nơi ở–làm việc:** mật độ dân số vùng (REGION_POPULATION_RELATIVE), xếp hạng vùng cư trú (REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY), và các cờ không khớp giữa nơi đăng ký–ở–làm việc như: REG_REGION_NOT_LIVE_REGION, REG_REGION_NOT_WORK_REGION, LIVE_REGION_NOT_WORK_REGION, REG_CITY_NOT_LIVE_CITY, REG_CITY_NOT_WORK_CITY, LIVE_CITY_NOT_WORK_CITY.
- **Liên hệ/thiết bị & trạng thái cung cấp:** FLAG_MOBIL, FLAG_EMP_PHONE, FLAG_WORK_PHONE, FLAG_CONT_MOBILE, FLAG_PHONE, FLAG_EMAIL — các cờ 0/1 phản ánh sự hiện diện của thông tin liên hệ.
- **Giấy tờ chứng minh:** nhóm cờ FLAG_DOCUMENT_2 … FLAG_DOCUMENT_21 thể hiện tình trạng có mặt của từng loại chứng từ.
- **Truy vấn tín dụng tại bureau:** biểu thị cường độ truy vấn tín dụng theo các cửa sổ thời gian khác nhau: AMT_REQ_CREDIT_BUREAU_HOUR, ...DAY, ...WEEK, ...MON, ...QRT, ...YEAR.

- **Biến mục tiêu (Label):** `TARGET` – biến nhị phân cho biết khách hàng có phát sinh khó khăn khi trả nợ (1) hay không (0) trong cửa sổ theo dõi sau thời điểm ra quyết định.

Mục tiêu chính của bộ dữ liệu là cung cấp góc nhìn toàn diện về hồ sơ và hành vi khách hàng tại thời điểm thẩm định, từ đó dự đoán xác suất vỡ nợ để hỗ trợ phê duyệt tín dụng, định giá theo rủi ro, phân bổ hạn mức và kích hoạt can thiệp sớm. Đồng thời, bộ dữ liệu cũng bảo đảm tuân thủ nguyên tắc công bằng mô hình khi xem xét các biến nhân khẩu học hoặc đặc trưng vùng.

2.1.2 Mô tả các feature của bộ dữ liệu

Bảng 2.1: *Mô tả chi tiết các biến trong bộ dữ liệu*

Tên biến	**Kiểu dữ liệu**	**Kiểu biến**	**Ý nghĩa**
SK_ID_CURR	int64	đặc trưng	Mã khách hàng.
TARGET	int64	nhãn	1 = có khó khăn khi trả nợ; 0 = trả nợ bình thường.
NAME_CONTRACT_TYPE	object	phân loại	Loại hợp đồng tín dụng (vay tiêu dùng, tín dụng xoay vòng,...).
CODE_GENDER	object	phân loại	Giới tính khách hàng.
FLAG_OWN_CAR	object	phân loại	Cờ sở hữu ô tô (1 = Có, 0 = Không).
FLAG_OWN_REALTY	object	phân loại	Cờ sở hữu nhà/đất (1 = Có, 0 = Không).
CNT_CHILDREN	int64	đặc trưng	Số lượng con.
AMT_INCOME_TOTAL	float64	đặc trưng	Tổng thu nhập năm (đơn vị nội bộ).
AMT_CREDIT	float64	đặc trưng	Số tiền vay/tín dụng.
AMT_ANNUITY	float64	đặc trưng	Khoản trả góp định kỳ (niên kim).
AMT_GOODS_PRICE	float64	đặc trưng	Giá trị hàng hoá/dịch vụ được tài trợ.
NAME_TYPE_SUITE	object	phân loại	Nhóm/người đi cùng khi nộp hồ sơ.
NAME_INCOME_TYPE	object	phân loại	Loại hình thu nhập (làm công, tự doanh, hưu trí,...).
NAME_EDUCATION_TYPE	object	phân loại	Trình độ học vấn.
NAME_FAMILY_STATUS	object	phân loại	Tình trạng gia đình.
NAME_HOUSING_TYPE	object	phân loại	Loại hình nhà ở/cư trú.
REGION_POPULATION_RELATIVE	float64	đặc trưng	Mật độ dân số vùng (chuẩn hoá).
DAYS_BIRTH	int64	đặc trưng	Số ngày tính từ ngày sinh (âm; trị tuyệt đối lớn ⇒ tuổi cao).
DAYS_EMPLOYED	int64	đặc trưng	Số ngày làm việc/thâm niên (âm; trị tuyệt đối phản ánh thâm niên).
DAYS_REGISTRATION	float64	đặc trưng	Số ngày kể từ thời điểm đăng ký cư trú/hồ sơ.
DAYS_ID_PUBLISH	int64	đặc trưng	Số ngày kể từ khi cấp/cập nhật giấy tờ tuỳ thân.
OWN_CAR_AGE	float64	đặc trưng	Tuổi xe (năm) nếu có sở hữu ô tô.
FLAG_MOBIL	int64	đặc trưng	Có số điện thoại di động (1/0).

(tiếp trang sau)

Bảng 2.1 – tiếp theo từ trang trước

Tên biến	Kiểu dữ liệu	Kiểu biến	Ý nghĩa
FLAG_EMP_PHONE	int64	đặc trưng	Có số điện thoại công ty (1/0).
FLAG_WORK_PHONE	int64	đặc trưng	Có số điện thoại nơi làm việc (1/0).
FLAG_CONT_MOBILE	int64	đặc trưng	Liên hệ di động hoạt động/duy trì (1/0).
FLAG_PHONE	int64	đặc trưng	Có cung cấp số điện thoại (1/0).
FLAG_EMAIL	int64	đặc trưng	Có cung cấp email (1/0).
OCCUPATION_TYPE	object	phân loại	Nghề nghiệp/chức danh công việc.
CNT_FAM_MEMBERS	float64	đặc trưng	Số thành viên gia đình.
REGION_RATING_CLIENT	int64	đặc trưng	Xếp hạng rủi ro vùng cư trú (nội bộ).
REGION_RATING_CLIENT_W_CITY	int64	đặc trưng	Xếp hạng vùng cư trú (có xét yếu tố thành phố).
WEEKDAY_APPR_PROCESS_START	object	phân loại	Thứ trong tuần khi hệ thống bắt đầu xử lý hồ sơ.
HOUR_APPR_PROCESS_START	int64	đặc trưng	Giờ trong ngày khi hệ thống bắt đầu xử lý hồ sơ.
REG_REGION_NOT_LIVE_REGION	int64	đặc trưng	Vùng đăng ký khác vùng đang cư trú (cờ 1/0).
REG_REGION_NOT_WORK_REGION	int64	đặc trưng	Vùng đăng ký khác vùng làm việc (cờ 1/0).
LIVE_REGION_NOT_WORK_REGION	int64	đặc trưng	Vùng cư trú khác vùng làm việc (cờ 1/0).
REG_CITY_NOT_LIVE_CITY	int64	đặc trưng	Thành phố đăng ký khác thành phố cư trú (cờ 1/0).
REG_CITY_NOT_WORK_CITY	int64	đặc trưng	Thành phố đăng ký khác thành phố làm việc (cờ 1/0).
LIVE_CITY_NOT_WORK_CITY	int64	đặc trưng	Thành phố cư trú khác thành phố làm việc (cờ 1/0).
ORGANIZATION_TYPE	object	phân loại	Loại hình tổ chức/cơ quan nơi làm việc.
EXT_SOURCE_1	float64	đặc trưng	Điểm ngoại sinh tổng hợp 1 (rủi ro tín dụng).
EXT_SOURCE_2	float64	đặc trưng	Điểm ngoại sinh tổng hợp 2 (rủi ro tín dụng).
EXT_SOURCE_3	float64	đặc trưng	Điểm ngoại sinh tổng hợp 3 (rủi ro tín dụng).
APARTMENTS_AVG	float64	đặc trưng	Chỉ số diện tích căn hộ (trung bình, chuẩn hoá).
BASEMENTAREA_AVG	float64	đặc trưng	Chỉ số diện tích tầng hầm (trung bình, chuẩn hoá).
YEARS_BEGINEXPLUATATION_AVG	float64	đặc trưng	Năm bắt đầu đưa công trình vào sử dụng (trung bình, chuẩn hoá).
YEARS_BUILD_AVG	float64	đặc trưng	Năm xây dựng công trình (trung bình, chuẩn hoá).
COMMONAREA_AVG	float64	đặc trưng	Diện tích khu vực dùng chung (trung bình, chuẩn hoá).
ELEVATORS_AVG	float64	đặc trưng	Số thang máy (trung bình, chuẩn hoá).

(tiếp trang sau)

Bảng 2.1 – tiếp theo từ trang trước

Tên biến	Kiểu dữ liệu	Kiểu biến	Ý nghĩa
ENTRANCES_AVG	float64	đặc trưng	Số lối vào toà nhà (trung bình, chuẩn hoá).
FLOORSMAX_AVG	float64	đặc trưng	Số tầng tối đa (trung bình, chuẩn hoá).
FLOORSMIN_AVG	float64	đặc trưng	Số tầng tối thiểu (trung bình, chuẩn hoá).
LANDAREA_AVG	float64	đặc trưng	Diện tích đất (trung bình, chuẩn hoá).
LIVINGAPARTMENTS_AVG	float64	đặc trưng	Số căn hộ có người ở (trung bình, chuẩn hoá).
LIVINGAREA_AVG	float64	đặc trưng	Diện tích ở (trung bình, chuẩn hoá).
NONLIVINGAPARTMENTS_AVG	float64	đặc trưng	Số căn hộ không có người ở (trung bình, chuẩn hoá).
NONLIVINGAREA_AVG	float64	đặc trưng	Diện tích không dùng để ở (trung bình, chuẩn hoá).
APARTMENTS_MODE	float64	đặc trưng	Diện tích căn hộ (mode, chuẩn hoá).
BASEMENTAREA_MODE	float64	đặc trưng	Diện tích tầng hầm (mode, chuẩn hoá).
YEARS_BEGINEXPLUATATION_MODE	float64	đặc trưng	Năm bắt đầu sử dụng (mode, chuẩn hoá).
YEARS_BUILD_MODE	float64	đặc trưng	Năm xây dựng (mode, chuẩn hoá).
COMMONAREA_MODE	float64	đặc trưng	Khu vực dùng chung (mode, chuẩn hoá).
ELEVATORS_MODE	float64	đặc trưng	Số thang máy (mode, chuẩn hoá).
ENTRANCES_MODE	float64	đặc trưng	Số lối vào (mode, chuẩn hoá).
FLOORSMAX_MODE	float64	đặc trưng	Số tầng tối đa (mode, chuẩn hoá).
FLOORSMIN_MODE	float64	đặc trưng	Số tầng tối thiểu (mode, chuẩn hoá).
LANDAREA_MODE	float64	đặc trưng	Diện tích đất (mode, chuẩn hoá).
LIVINGAPARTMENTS_MODE	float64	đặc trưng	Số căn hộ có người ở (mode, chuẩn hoá).
LIVINGAREA_MODE	float64	đặc trưng	Diện tích ở (mode, chuẩn hoá).
NONLIVINGAPARTMENTS_MODE	float64	đặc trưng	Số căn hộ không người ở (mode, chuẩn hoá).
NONLIVINGAREA_MODE	float64	đặc trưng	Diện tích không dùng để ở (mode, chuẩn hoá).
APARTMENTS_MEDI	float64	đặc trưng	Diện tích căn hộ (median, chuẩn hoá).
BASEMENTAREA_MEDI	float64	đặc trưng	Diện tích tầng hầm (median, chuẩn hoá).
YEARS_BEGINEXPLUATATION_MEDI	float64	đặc trưng	Năm bắt đầu sử dụng (median, chuẩn hoá).
YEARS_BUILD_MEDI	float64	đặc trưng	Năm xây dựng (median, chuẩn hoá).

(tiếp trang sau)

Bảng 2.1 – tiếp theo từ trang trước

Tên biến	Kiểu dữ liệu	Kiểu biến	Ý nghĩa
COMMONAREA_MEDI	float64	đặc trưng	Khu vực dùng chung (median, chuẩn hoá).
ELEVATORS_MEDI	float64	đặc trưng	Số thang máy (median, chuẩn hoá).
ENTRANCES_MEDI	float64	đặc trưng	Số lối vào (median, chuẩn hoá).
FLOORSMAX_MEDI	float64	đặc trưng	Số tầng tối đa (median, chuẩn hoá).
FLOORSMIN_MEDI	float64	đặc trưng	Số tầng tối thiểu (median, chuẩn hoá).
LANDAREA_MEDI	float64	đặc trưng	Diện tích đất (median, chuẩn hoá).
LIVINGAPARTMENTS_MEDI	float64	đặc trưng	Số căn hộ có người ở (median, chuẩn hoá).
LIVINGAREA_MEDI	float64	đặc trưng	Diện tích ở (median, chuẩn hoá).
NONLIVINGAPARTMENTS_MEDI	float64	đặc trưng	Số căn hộ không người ở (median, chuẩn hoá).
NONLIVINGAREA_MEDI	float64	đặc trưng	Diện tích không dùng để ở (median, chuẩn hoá).
FONDKAPREMONT_MODE	object	phân loại	Loại quỹ sửa chữa toà nhà (biến phân loại).
HOUSETYPE_MODE	object	phân loại	Loại hình nhà ở (biến phân loại).
TOTALAREA_MODE	float64	đặc trưng	Tổng diện tích căn hộ/nhà ở (mode, chuẩn hoá).
WALLSMATERIAL_MODE	object	phân loại	Vật liệu tường (biến phân loại).
EMERGENCYSTATE_MODE	object	phân loại	Trạng thái khẩn cấp (nếu có).
OBS_30_CNT_SOCIAL_CIRCLE	float64	đặc trưng	Chỉ số liên quan mốc 30 ngày (nguồn xã hội).
DEF_30_CNT_SOCIAL_CIRCLE	float64	đặc trưng	Chỉ số liên quan mốc 30 ngày (nguồn xã hội).
OBS_60_CNT_SOCIAL_CIRCLE	float64	đặc trưng	Chỉ số liên quan mốc 60 ngày (nguồn xã hội).
DEF_60_CNT_SOCIAL_CIRCLE	float64	đặc trưng	Chỉ số liên quan mốc 60 ngày (nguồn xã hội).
DAYS_LAST_PHONE_CHANGE	float64	đặc trưng	Số ngày kể từ lần đổi số điện thoại gần nhất.
FLAG_DOCUMENT_2	int64	đặc trưng	Cờ có giấy tờ #2 (1/0).
FLAG_DOCUMENT_3	int64	đặc trưng	Cờ có giấy tờ #3 (1/0).
FLAG_DOCUMENT_4	int64	đặc trưng	Cờ có giấy tờ #4 (1/0).
FLAG_DOCUMENT_5	int64	đặc trưng	Cờ có giấy tờ #5 (1/0).
FLAG_DOCUMENT_6	int64	đặc trưng	Cờ có giấy tờ #6 (1/0).
FLAG_DOCUMENT_7	int64	đặc trưng	Cờ có giấy tờ #7 (1/0).
FLAG_DOCUMENT_8	int64	đặc trưng	Cờ có giấy tờ #8 (1/0).
FLAG_DOCUMENT_9	int64	đặc trưng	Cờ có giấy tờ #9 (1/0).
FLAG_DOCUMENT_10	int64	đặc trưng	Cờ có giấy tờ #10 (1/0).
FLAG_DOCUMENT_11	int64	đặc trưng	Cờ có giấy tờ #11 (1/0).
FLAG_DOCUMENT_12	int64	đặc trưng	Cờ có giấy tờ #12 (1/0).
FLAG_DOCUMENT_13	int64	đặc trưng	Cờ có giấy tờ #13 (1/0).
FLAG_DOCUMENT_14	int64	đặc trưng	Cờ có giấy tờ #14 (1/0).
FLAG_DOCUMENT_15	int64	đặc trưng	Cờ có giấy tờ #15 (1/0).

(tiếp trang sau)

Bảng 2.1 – tiếp theo từ trang trước

Tên biến	Kiểu dữ liệu	Kiểu biến	Ý nghĩa
FLAG_DOCUMENT_16	int64	đặc trưng	Cờ có giấy tờ #16 (1/0).
FLAG_DOCUMENT_17	int64	đặc trưng	Cờ có giấy tờ #17 (1/0).
FLAG_DOCUMENT_18	int64	đặc trưng	Cờ có giấy tờ #18 (1/0).
FLAG_DOCUMENT_19	int64	đặc trưng	Cờ có giấy tờ #19 (1/0).
FLAG_DOCUMENT_20	int64	đặc trưng	Cờ có giấy tờ #20 (1/0).
FLAG_DOCUMENT_21	int64	đặc trưng	Cờ có giấy tờ #21 (1/0).
AMT_REQ_CREDIT_BUREAU_HOUR	float64	đặc trưng	Số lần truy vấn tín dụng trong 1 giờ gần nhất.
AMT_REQ_CREDIT_BUREAU_DAY	float64	đặc trưng	Số lần truy vấn tín dụng trong 1 ngày gần nhất.
AMT_REQ_CREDIT_BUREAU_WEEK	float64	đặc trưng	Số lần truy vấn tín dụng trong 1 tuần gần nhất.
AMT_REQ_CREDIT_BUREAU_MON	float64	đặc trưng	Số lần truy vấn tín dụng trong 1 tháng gần nhất.
AMT_REQ_CREDIT_BUREAU_QRT	float64	đặc trưng	Số lần truy vấn tín dụng trong 1 quý gần nhất.
AMT_REQ_CREDIT_BUREAU_YEAR	float64	đặc trưng	Số lần truy vấn tín dụng trong 1 năm gần nhất.

Chương 3

Cơ sở lý thuyết

3.1 Tiền xử lý dữ liệu

3.1.1 Xử lý dữ liệu khuyết với Phương pháp gán đa biến bằng phương trình chuỗi (MICE - Multivariate imputation by chained equations) [?]

3.1.1.1 Sơ lược về MICE

Thiếu khuyết dữ liệu là vấn đề phổ biến thường xảy ra khi thực hiện các bài toán với dữ liệu. Khi gặp trường hợp thiếu khuyết nếu thực hiện bỏ hoàn toàn hoặc chỉ thay thế bằng một giá trị trung bình có thể dẫn đến mất thông tin, giảm độ chính xác và bias trong phân tích. Nhằm giải quyết vấn đề trên, **MICE** là phương án xử lý dữ liệu thiếu khuyết đặc biệt hữu ích cho quy trình gán lớn.

Phương pháp gán đa biến bằng phương trình chuỗi (MICE), còn được gọi là **"đặc tả có điều kiện đầy đủ"(FCS)** hoặc **"gán nhiều hồi quy tuần tự"** đã xuất hiện trong tài liệu thống kê như một phương pháp nguyên tắc để giải quyết dữ liệu bị thiếu. Tạo ra nhiều quy gán, trái ngược với các quy gán đơn lẻ, giải thích cho sự không chắc chắn thống kê trong các quy gán. Ngoài ra, phương pháp phương trình chuỗi rất linh hoạt và có thể xử lý các biến thuộc các loại khác nhau cũng như độ phức tạp như giới hạn hoặc mẫu bỏ qua khảo sát.

*** Quy trình hiện thực MICE:**

1. **Một phép gán đơn giản**, chẳng hạn như gán giá trị trung bình **(mean)** hay trung vị **(median)**, được thực hiện cho mọi giá trị bị thiếu trong tập dữ liệu. Những quy gán trung bình này có thể được coi là "người giữ chỗ".

2. **"Người giữ chỗ"** có nghĩa là gán cho một biến **("var")** được đặt trở lại thiếu.

3. **Các giá trị quan sát** được từ biến **"var"** trong **Bước 2** được hồi quy trên các biến khác trong mô hình gán, có thể bao gồm hoặc không bao gồm tất cả các biến trong tập dữ liệu. Nói cách khác, **"var"** là biến phụ thuộc trong mô hình hồi quy và tất cả các biến khác là biến độc lập trong mô hình hồi quy. Các mô hình hồi quy này hoạt động theo cùng một giả định mà người ta sẽ thực hiện khi thực hiện các mô hình **hồi quy tuyến tính**, **logistic** hoặc **Poison** bên ngoài bối cảnh gán dữ liệu bị thiếu.

4. **Các giá trị bị thiếu** cho **"var"** sau đó được thay thế bằng các dự đoán (gán) từ **mô hình hồi quy**. Khi **"var"** sau đó được sử dụng như một biến độc lập trong các mô hình hồi quy cho các biến khác, cả hai giá trị quan sát được và các giá trị được gán này sẽ được sử dụng.

5. Sau đó, các **bước 2–4 được lặp lại** cho mỗi biến bị thiếu dữ liệu. Chu kỳ qua mỗi biến tạo thành một **vòng lặp** hoặc **"chu kỳ"**. Vào cuối một chu kỳ, tất cả các giá trị bị thiếu đã được thay thế bằng các dự đoán từ hồi quy phản ánh các **mối quan hệ quan sát** được trong dữ liệu.

6. Các **bước 2–4 được lặp lại** trong một số chu kỳ, với các quy gán được cập nhật ở mỗi chu kỳ.

3.1.1.2 Hiện thực MICE trong Python

MICE trong Python được hiện thực thông qua `sklearn.experimental.IterativeImputer`. Các tham số thiết lập MICE:

Tham số	**Giá trị mặc định / Kiểu dữ liệu**	**Ý nghĩa / Giải thích**
estimator	BayesianRidge()	Mô hình hồi quy được dùng để dự đoán giá trị bị khuyết. Có thể thay bằng mô hình khác (LinearRegression, RandomForestRegressor, v.v.).
missing_values	np.nan	Giá trị được xem là missing trong dữ liệu (thường là `np.nan`).
sample_posterior	False	Nếu True, mẫu ngẫu nhiên được rút từ phân phối hậu nghiệm (posterior) của mô hình thay vì giá trị dự đoán trung bình. Giúp phản ánh bất định (uncertainty) khi multiple imputation.
max_iter	10	Số vòng lặp tối đa qua tất cả các biến cần được impute. Giá trị cao hơn có thể cho kết quả ổn định hơn nhưng tốn thời gian hơn.
tol	1e-3	Ngưỡng hội tụ (tolerance). Nếu thay đổi trung bình giữa hai vòng nhỏ hơn `tol`, quá trình dừng sớm.
imputation_order	'ascending'	Thứ tự các biến được impute trong mỗi vòng: `'ascending'` (từ ít missing đến nhiều missing), `'descending'`, `'roman'`, `'arabic'`, hoặc `'random'`.
skip_complete	False	Nếu True, bỏ qua các biến không có missing values để tiết kiệm thời gian.
min_value	-np.inf	Giới hạn nhỏ nhất của giá trị được impute. Dùng để tránh sinh ra giá trị âm hoặc vượt giới hạn.
max_value	np.inf	Giới hạn lớn nhất của giá trị được impute. Ví dụ, đặt `max_value=1` nếu biến là tỷ lệ phần trăm.
verbose	0	Mức độ hiển thị thông tin khi chạy: 0 = im lặng, 1 = thông tin cơ bản, 2 = chi tiết.
random_state	None	Hạt giống (seed) để tái lập kết quả ngẫu nhiên trong quá trình imputation.
add_indicator	False	Nếu True, thêm cột chỉ báo (binary indicator) cho biết giá trị nào bị khuyết trong dữ liệu gốc. Hữu ích cho mô hình học máy muốn tận dụng thông tin "missingness".
initial_strategy	'mean'	Cách khởi tạo giá trị ban đầu cho các biến khuyết: `'mean'`, `'median'`, `'most_frequent'`, hoặc `'constant'`.
fill_value	None	Chỉ dùng nếu `initial_strategy='constant'`. Giá trị được dùng để thay thế ban đầu cho missing.
n_nearest_features	None	Số lượng biến (features) gần nhất được sử dụng làm predictor khi dự đoán giá trị khuyết. Nếu None, dùng toàn bộ biến còn lại.
initial_imputer	None	Cho phép truyền một imputer khác để khởi tạo giá trị ban đầu thay vì dùng `initial_strategy`.
keep_empty_features	False	Nếu True, vẫn giữ các cột có toàn bộ giá trị bị missing trong dữ liệu đầu vào. Mặc định bỏ qua vì không thể impute.

Bảng 3.1: *Các tham số của `IterativeImputer` trong scikit-learn và ý nghĩa chi tiết.*

3.1.2 Trọng số của bằng chứng (WOE - Weight Of Envidence) và Giá trị thông tin (IV - Information Value) [?]

Mô hình hồi quy logistic là một trong những kỹ thuật thống kê được sử dụng phổ biến nhất để giải bài toán phân loại nhị phân. Đây là một kỹ thuật được chấp nhận trong hầu hết các lĩnh vực. Hai khái niệm này - **trọng số bằng chứng (WOE)** và **giá trị thông tin (IV)** phát triển từ cùng một kỹ thuật hồi quy logistic. Hai thuật ngữ này đã tồn tại trong **thế giới chấm điểm tín dụng** trong hơn 4-5 thập kỷ. Hai thông số đã được sử dụng như một tiêu chuẩn để **sàng lọc các biến số** trong các dự án **mô hình hóa rủi ro tín dụng** như **xác suất vỡ nợ**. Chúng giúp khám phá dữ liệu và các biến màn hình. Chúng cũng được sử dụng trong **dự án phân tích tiếp thị** như mô hình tiêu hao khách hàng, mô hình phản hồi chiến dịch, v.v.

3.1.2.1 Trọng số của bằng chứng (WOE - Weight Of Envidence)

Trọng lượng của bằng chứng (WOE) cho biết **khả năng dự đoán** của một biến độc lập liên quan đến biến phụ thuộc. Vì thông số này được phát triển từ **thế giới chấm điểm tín dụng**, nó thường được mô tả như **một thước đo sự tách biệt** giữa khách hàng tốt và xấu. **"Khách hàng xấu"** đề cập đến những khách hàng không trả được khoản vay và **"Khách hàng tốt"** là khách hàng đã trả lại khoản vay.

$$WOE = ln(\frac{Dist_{Goods}}{Dist_{Bads}}) \tag{3.1}$$

*** Trong đó:**

- $Dist_{Goods}$ là tỉ lệ ***"Khách hàng tốt"*** trong 1 nhóm cụ thể.
- $Dist_{Bads}$ là tỉ lệ ***"Khách hàng xấu"*** trong 1 nhóm cụ thể.
- $Dist_{Goods} > Dist_{Bads}$ dẫn đến giá trị WOE dương.
- $Dist_{Goods} < Dist_{Bads}$ dẫn đến giá trị WOE âm.

Trọng lượng của bằng chứng (WOE) giúp chuyển đổi một biến độc lập liên tục thành một tập hợp các nhóm hoặc thùng dựa trên sự tương đồng của phân phối biến phụ thuộc, tức là số lượng quan sát "diễn ra sự kiện" và "không diễn ra sự kiện".

- **Đối với các biến độc lập liên tục (numeric):** Đầu tiên, **tạo các thùng (danh mục / nhóm)** cho một biến độc lập liên tục, sau đó kết hợp các danh mục có giá trị WOE tương tự và thay thế các danh mục bằng các giá trị WOE. Sử dụng các giá trị WOE thay vì các giá trị đầu vào trong mô hình.
- **Đối với các biến độc lập phân loại (categrical):** Kết hợp các danh mục có WOE tương tự và sau đó tạo các danh mục mới của một biến độc lập với các giá trị WOE liên tục. Nói cách khác, hãy sử dụng giá trị WOE thay vì danh mục thô trong mô hình. Biến biến đổi sẽ là một biến liên tục với các giá trị WOE giống như bất kỳ biến liên tục nào.

Việc kết hợp các **"danh mục"** có WOE tương tự sẽ cho tỉ lệ "diễn ra sự kiện" và "không diễn ra sự kiện" gần như bằng nhau.

*** Ứng dụng của Trọng số bằng chứng:**

- Thống số này có thể xử lý các giá trị ngoại biên (outliers).
- Nó có thể được sử dụng để xử lý dữ liệu khuyết bằng cách gom nhóm dữ liệu khuyết thành 1 danh mục tách biệt.
- Việc biến đổi WOE xử lý biến phân loại giúp cho việc chuyển đổi biến giả là không cần thiết.
- Chuyển đổi WOE giúp xây dựng mối quan hệ tuyến tính chặt chẽ với tỷ lệ cược log. Nếu không, không dễ dàng để hoàn thành mối quan hệ tuyến tính bằng cách sử dụng các phương pháp biến đổi khác như log, căn bậc hai, v.v. Tóm lại, nếu không sử dụng phép biến đổi WOE, có thể phải thử một số phương pháp biến đổi để đạt được điều này.

3.1.2.2 Giá trị thông tin (IV - Information Value)

Giá trị thông tin (IV) là một trong những kỹ thuật hữu ích nhất để chọn các biến quan trọng trong mô hình dự đoán. Thông số giúp xếp hạng các biến trên cơ sở tầm quan trọng của chúng. IV được tính theo công thức sau:

$$IV = \sum(Dist_{Goods} - Dist_{Bads}) * WOE \tag{3.2}$$

Theo **Siddiqi (2006)**, theo quy ước, các giá trị của thống kê IV trong chấm điểm tín dụng có thể được hiểu như sau.

Giá trị thông tin (IV)	Khả năng dự đoán biến đổi
Dưới 0,02	Không hữu ích cho việc lập mô hình dự đoán
0,02 đến 0,1	Khả năng dự đoán yếu
0,1 đến 0,3	Công suất dự đoán trung bình
0,3 đến 0,5	Khả năng dự đoán mạnh mẽ
> 0,5	Khả năng dự đoán đáng ngờ, cần kiểm tra lại

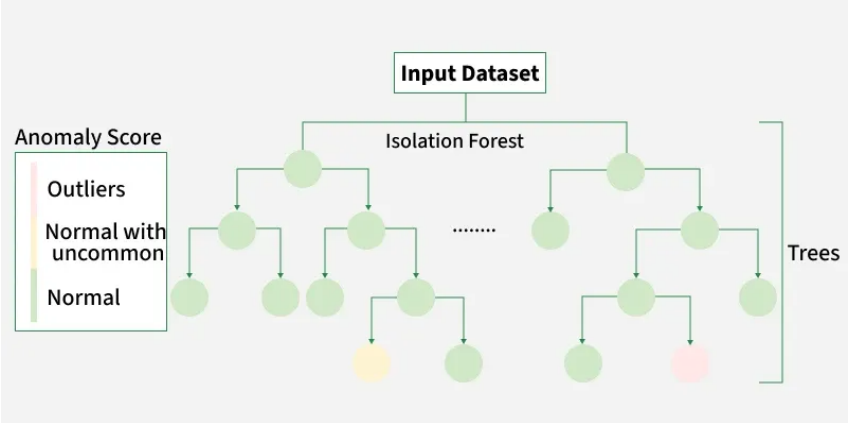
Bảng 3.2: *Thang đánh giá mức độ thông tin (Information Value - IV)*

*** Những điểm quan trọng:**

- Giá trị thông tin tăng khi các thùng / nhóm tăng lên đối với một biến độc lập. Hãy cẩn thận khi có hơn 20 thùng vì một số nhóm có thể có rất ít quan sát "diễn ra sự kiện" và "không diễn ra sự kiện".
- Giá trị thông tin không phải là một phương pháp lựa chọn tính năng (biến) tối ưu khi bạn đang xây dựng một mô hình phân loại khác với hồi quy logistic nhị phân vì tỷ lệ cược log có điều kiện (mà chúng tôi dự đoán trong mô hình hồi quy logistic) có liên quan nhiều đến việc tính toán trọng số của bằng chứng. Nói cách khác, thông số này được thiết kế chủ yếu cho mô hình hồi quy logistic nhị phân. Cũng hãy nghĩ theo cách này - **Rừng ngẫu nhiên (Random Forest)** có thể phát hiện mối quan hệ phi tuyến tính rất tốt, vì vậy việc chọn các biến thông qua **Giá trị thông tin** và sử dụng chúng trong mô hình rừng ngẫu nhiên có thể không tạo ra mô hình dự đoán chính xác và mạnh mẽ nhất.

3.1.3 Xử lý dữ liệu ngoại biên (outliers) với Rừng cô lập (Isolation Forest) [?]

Isolation Forest là một thuật toán hiệu quả và đơn giản được sử dụng để **phát hiện bất thường**, khiến nó trở thành lựa chọn phổ biến trong các ngành như an ninh mạng, tài chính và chăm sóc sức khỏe. Thuật toán xác định **các ngoại lệ** trong các bộ dữ liệu lớn bằng cách **cô lập** chúng thông qua **phân vùng nhị phân** đòi hỏi chi phí tính toán tối thiểu. Khả năng nhanh chóng tìm ra **điểm bất thường** này rất quan trọng trong các ứng dụng mà việc phát hiện các mẫu bất thường là chìa khóa để bảo vệ khỏi rủi ro hoặc xác định thông tin chi tiết ẩn.



Hình 3.1: *Rừng cô lập - Isolation Forest*

Trong sơ đồ, "Tập dữ liệu đầu vào" (Input Dataset) ở trên cùng. Tập dữ liệu này sau đó được chia thành hai nhánh, được gắn nhãn "Bình thường với không phổ biến" (Normal with uncommon) và "Ngoại lệ" (Outliers).

Nhánh "Bình thường với không phổ biến" lại tách ra cho đến khi đạt đến nhãn "Bình thường" (). Điều này cho thấy rằng các điểm dữ liệu được phân loại là bình thường có thể có một số đặc điểm bất thường. Nhánh "Ngoại lệ" đạt đến nhãn "Ngoại lệ" nhanh hơn, cho thấy rằng các ngoại lệ có thể được xác định tương đối dễ dàng bằng cách sử dụng Rừng cô lập.

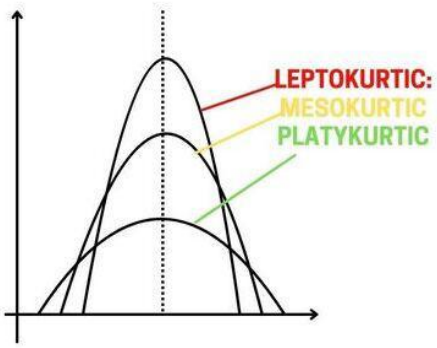
*** Cách thức hoạt động của thuật toán Isolation Forest:**

1. **Phân vùng ngẫu nhiên - Random Partitioning:** Thuật toán bắt đầu với việc chọn đặc trưng ngẫu nhiên từ bộ dữ liệu. Sau đó, chia dữ liệu theo một giá trị ngẫu nhiên trong phạm vi của đối tượng địa lý đó, chia thành hai phần. Quá trình này được lặp lại đệ quy giúp tạo cây nhị phân trong đó mỗi nhánh đại diện cho một sự phân tách trong dữ liệu.
2. **Con đường cách ly - Isolation Path:** Đề cập đến số lượng phân tách cần thiết để cô lập một điểm dữ liệu. Các dị thể thường có đường dẫn ngắn hơn vì chúng ở xa hầu hết dữ liệu, đòi hỏi ít phân tách hơn để tách chúng.
3. **Quần thể cây cối - Ensemble of Trees:** Thay vì dựa vào một cái cây, nó xây dựng một quần thể cây. Mỗi cây được tạo độc lập với các phân tách ngẫu nhiên giúp dẫn đến các đường dẫn cách ly đa dạng cho từng điểm dữ liệu trên nhiều cây. Điều này đảm bảo tính mạnh mẽ và độ tin cậy trong kết quả.
4. **Chấm điểm bất thường - Anomaly Scoring:** Điểm bất thường cho mỗi điểm dữ liệu được tính bằng cách tính trung bình độ dài đường dẫn trên tất cả các cây. Đường đi ngắn hơn (ít phân tách hơn) cho thấy điểm có nhiều khả năng là một điểm bất thường.
5. **Phân loại - Classification:** Để phân loại các điểm dữ liệu là bình thường hoặc bất thường, thuật toán đặt ngưỡng cho điểm bất thường. Các điểm trên ngưỡng được phân loại là dị thường trong khi các điểm bên dưới được coi là bình thường.

3.1.4 Phân tích độ nhọn (Kurtosis) và độ lệch (Skewness)

3.1.4.1 Độ nhọn (Kurtosis) [?]

Kurtosis là một thước đo thống kê mô tả hình dạng của phân phối dữ liệu, đặc biệt là mức độ nặng hay nhẹ của đuôi. Thông số cho chúng ta biết liệu một tập dữ liệu có nhiều điểm ngoại lệ hơn phân phối chuẩn hay hầu hết các điểm dữ liệu gần với mức trung bình hơn. Thông số này cho biết một bức tranh rõ nét hơn về mức độ lan rộng hoặc đỉnh của dữ liệu thực sự vượt ra ngoài giá trị trung bình và phương sai.



Hình 3.2: *Độ nhọn - Kurtosis*

Kurtosis (độ nhọn) định lượng mức độ mà các điểm dữ liệu tập trung ở phần đuôi hoặc đỉnh của một phân phối. Công thức của nó thường được xác định dựa trên moment chuẩn hóa bậc bốn (fourth standardized moment):

$$Kurtosis = \frac{\mathbf{E}[(X-\mu)^4]}{\sigma^4} \tag{3.3}$$

Trong đó, μ là giá trị trung bình và σ là độ lệch chuẩn. Phân phố chuẩn có giá trị độ nhọn là 3, tức là mesokurtic.

*** Các loại độ nhọn:**

1. **Leptokurtic:** Phân bố có đuôi rộng và kurtosis dương được gọi là phân bố leptokurtic. Dữ liệu có các ngoại lệ cực đoan, đạt đỉnh mạnh ở mức trung bình.
2. **Mesokurtic:** Khi kurtosis dư thừa bằng không hoặc gần bằng không được gọi là phân bố mesokurtic. Phân bố giống như đường cong bình thường, xác suất tiêu chuẩn của các sự kiện cực đoan.
3. **PlatyKurtic:** Khi kurtosis dư thừa âm tính được gọi là phân bố platykurtic. Dữ liệu trải đều hơn, đỉnh phẳng hơn và đuôi nhẹ hơn.

*** Một số ứng dụng của phân tích độ nhọn:**

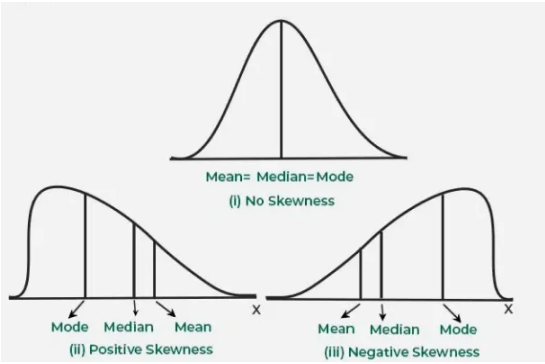
1. **Tài chính:** Kurtosis giúp đo lường rủi ro bằng cách phát hiện những đuôi nặng nề trong lợi nhuận cổ phiếu, cho thấy khả năng thua lỗ hoặc lợi nhuận cực lớn.
2. **Kiểm soát chất lượng:** Xác định khả năng xảy ra lỗi hoặc ngoại lệ trong quy trình sản xuất.
3. **Thống kê và phân tích dữ liệu:** Được sử dụng cùng với giá trị trung bình và phương sai để mô tả các đặc điểm phân phối đầy đủ hơn.
4. **Machine Learning:** Giúp phát hiện sự bất thường trong bộ dữ liệu và thông báo các quyết định tiền xử lý.
5. **Nghiên cứu và Khoa học Xã hội:** Đo lường mức độ cực đoan của các câu trả lời khảo sát hoặc thực nghiệm.

*** Hạn chế của phân tích độ nhọn:**

1. **Nhạy cảm với các giá trị ngoại lệ:** Cực kỳ bị ảnh hưởng bởi một vài giá trị cực trị trong dữ liệu.
2. **Gây hiểu lầm cho các mẫu nhỏ:** Bộ dữ liệu nhỏ có thể tạo ra các giá trị kurtosis không chính xác.
3. **Không hiển thị hướng:** Kurtosis cho biết trọng lượng đuôi nhưng không cho biết dữ liệu bị lệch sang trái hay phải.
4. **Bối cảnh nhu cầu:** Nên được giải thích với các thước đo thống kê khác để có những hiểu biết sâu sắc có ý nghĩa.

3.1.4.2 Độ lệch (Skewness) [?]

Độ lệch là một thước đo thống kê quan trọng cho thấy dữ liệu được trải rộng như thế nào trong tập dữ liệu. Giá trị này cho chúng ta biết liệu các điểm dữ liệu bị lệch sang trái (độ lệch âm) hay sang phải (độ lệch dương) liên quan đến giá trị trung bình. Điều này rất quan trọng vì nó giúp chúng ta hiểu hình dạng của phân phối dữ liệu, điều này rất quan trọng để phân tích dữ liệu chính xác và giúp xác định các ngoại lệ và tìm ra các phương pháp thống kê tốt nhất để sử dụng để phân tích.



Hình 3.3: *Độ lệch (Skewness)*

Độ lệch mô tả hướng và mức độ bất đối xứng trong phân phối của tập dữ liệu. Các dạng hình độ lệch:

1. **Độ lệch tích cực (Right Skewness):** Trong phân phối lệch dương, đuôi phải dài hơn đuôi trái, có nghĩa là hầu hết các điểm dữ liệu nằm ở bên trái với một vài giá trị lớn kéo phân phối sang phải.
2. **Độ lệch tiêu cực (Left Skewness):** Trong phân phối lệch âm, đuôi trái dài hơn, có nghĩa là hầu hết các điểm dữ liệu nằm ở bên phải với một vài giá trị nhỏ hơn kéo phân phối sang trái.
3. **Độ lệch bằng không (Symmetrical Distribution)**: Độ lệch bằng không cho thấy một phân phối đối xứng hoàn hảo trong đó giá trị trung bình, trung vị và chế độ bằng nhau. Trong phân phối đối xứng, các điểm dữ liệu được phân bố đều xung quanh điểm trung tâm.

Giá trị độ lệch được đo lường bằng các kĩ thuật khác nhau để định lượng mức độ bất đối xứng trong phân phối của tập dữ liệu. Dưới đây là ba phương pháp phổ biến để đo độ lệch:

1. **Phép đo lường của Karl Pearson**

   Phép đo lường của Karl Pearson sử dụng giá trị trung bình, trung bình và độ lệch chuẩn của dữ liệu đã cho để đo lường sự bất đối xứng của phân phối. Nó cung cấp một số không thứ nguyên giúp định lượng mức độ sai lệch của dữ liệu.

   *** Công thức:**

   (a) Trung bình và trung vị: $S_k = \frac{3*\hat{X}-M}{\sigma}$

   (b) Trung bình và Mode: $S_k = \frac{\hat{X}-Mode}{\sigma}$

   Trong đó:

   - S_k là hệ số độ lệch của *Karl Pearson*.
   - $\hat{X}$ là giá trị trung bình của tập dữ liệu.
   - M là trung vị của tập dữ liệu.
   - σ là độ lệch chuẩn của tập dữ liệu.

2. **Phép đo lường của Bowley**

   Hệ số độ lệch của Bowley là một phương pháp khác để tính độ lệch dựa trên tứ phân vị (Q_1, Q_2, Q_3). Không giống như Phép đo lường của Karl Pearson, nó không dựa vào giá trị trung bình hoặc độ lệch chuẩn, điều này làm cho nó hữu ích cho dữ liệu có thể không tuân theo phân phối chuẩn. Nó được tính bằng cách sử dụng tứ phân vị đầu tiên (Q_1), tứ phân vị thứ hai (Q_2 hoặc trung bình) và tứ phân vị thứ ba (Q_3).

   *** Công thức:** $B = \frac{Q_3+Q_1-2Q_2}{Q_3-Q_1}$

3. **Phép đo lường của Kelly**

   Phép đo lường của Kelly tính toán độ lệch bằng cách so sánh các phân vị nhất định trong dữ liệu, thường là phân vị thứ 10, 50 (trung bình) và 90. Biện pháp này hữu ích khi xử lý các bộ dữ liệu không được phân phối bình thường hoặc khi các biện pháp độ lệch khác có thể không hiệu quả.

   *** Công thức:** $SKL = \frac{P_{90}+P_{10}-2P_{50}}{P_{90}-P_{10}}$

   Trong đó:

   - P_{90} = Phân vị thứ 90
   - P_{50} = Phân vị thứ 50 (Trung bình)
   - P_{10} = Phân vị thứ 10

Việc diễn giải độ lệch (skewness) bao gồm việc hiểu **hướng lệch** (trái hoặc phải) và **mức độ lệch** (độ mạnh của sự bất đối xứng) trong phân phối dữ liệu:

1. **Hướng của Độ lệch (Direction of Skewness)**

   - **Độ lệch âm (Negative Skewness / Left-Skewed):** Khi skewness có giá trị âm, phân phối bị lệch về bên trái. Trong phân phối lệch trái:
     - Đuôi bên trái (phía các giá trị nhỏ) dài hơn và có thể chứa các ngoại biên (outliers).
     - Phần lớn các điểm dữ liệu tập trung về phía bên phải.
     - Giá trị trung bình (mean) nhỏ hơn giá trị trung vị (median).
   - **Độ lệch dương (Positive Skewness / Right-Skewed):** Khi skewness có giá trị dương, phân phối bị lệch về bên phải. Trong phân phối lệch phải:
     - Đuôi bên phải (phía các giá trị lớn) dài hơn và có thể chứa các ngoại biên.
     - Phần lớn các điểm dữ liệu tập trung về phía bên trái.
     - Giá trị trung bình (mean) lớn hơn giá trị trung vị (median).
   - **Độ lệch bằng không (Zero Skewness / Symmetric):** Khi giá trị skewness gần bằng 0, phân phối được xem là đối xứng, tức dữ liệu được phân bố đồng đều hai bên giá trị trung bình. Khi đó, phân phối không có độ lệch rõ rệt.

2. **Mức độ của Độ lệch (Magnitude of Skewness)**

   Mức độ (độ lớn) của skewness phản ánh mức độ nghiêm trọng của sự lệch trong phân phối:

   - $-0.5 \leq$ Skewness ≤ 0.5: Phân phối gần như đối xứng.
   - Skewness < -1: Phân phối lệch trái mạnh (strong negative skew) với đuôi dài ở bên trái.
   - Skewness > 1: Phân phối lệch phải mạnh (strong positive skew) với đuôi dài ở bên phải.

Khi làm việc với dữ liệu bị lệch (skewed data), điều quan trọng là phải hiểu cách xử lý độ lệch một cách hiệu quả. Dữ liệu có độ lệch cao có thể ảnh hưởng đến độ chính xác của các phân tích thống kê và kết quả dự đoán. Tuỳ theo bản chất của dữ liệu và loại phân tích cần thực hiện, ta có thể áp dụng các phương pháp khác nhau để giảm hoặc khắc phục độ lệch. Dưới đây là một số cách phổ biến:

1. **Biến đổi dữ liệu (Data Transformation)**

   - **Log Transformation:** Thường được sử dụng cho dữ liệu lệch phải, giúp nén các giá trị lớn và làm cho phân phối trở nên đối xứng hơn.
   - **Square Root / Cube Root:** Giúp giảm độ lệch dương, đặc biệt hữu ích đối với dữ liệu đếm (*count data*).
   - **Box–Cox Transformation:** Là một phương pháp linh hoạt, có thể xử lý cả độ lệch dương và âm, nhưng chỉ áp dụng được cho dữ liệu có giá trị dương.
   - **Yeo–Johnson Transformation:** Là phần mở rộng của Box–Cox, cho phép biến đổi cả dữ liệu có giá trị âm lẫn dương. Phương pháp này đặc biệt hữu ích khi tập dữ liệu chứa giá trị bằng không hoặc âm, giúp phân phối trở nên gần chuẩn hơn mà không cần loại bỏ dữ liệu.

2. **Loại bỏ giá trị ngoại biên (Removing Outliers)**

   Các ngoại biên (*outliers*) có thể là nguyên nhân gây ra độ lệch. Việc loại bỏ chúng có thể giúp phân phối dữ liệu trở nên đối xứng hơn:

   - **Z-score:** Xác định và loại bỏ các điểm dữ liệu có giá trị z vượt quá ± 3.
   - **IQR Method:** Loại bỏ các điểm dữ liệu nằm ngoài khoảng $1.5 \times$ IQR (interquartile range) tính từ tứ phân vị thứ nhất và thứ ba.

3. **Kiểm định phi tham số (Non-Parametric Tests)**

   Khi các phép biến đổi không mang lại hiệu quả, có thể sử dụng các kiểm định phi tham số như *Mann–Whitney U Test* hoặc *Kruskal–Wallis Test*. Các phương pháp này không giả định phân phối chuẩn và tập trung vào so sánh trung vị (*median*) thay vì trung bình (*mean*).

4. **Mô hình Học máy (Machine Learning Models)**

   Một số mô hình có khả năng xử lý dữ liệu lệch tốt hơn:

   - **Tree-based Models:** Các mô hình cây quyết định (*Decision Trees*) và rừng ngẫu nhiên (*Random Forests*) ít nhạy cảm hơn với độ lệch dữ liệu.
   - **Generalized Linear Models (GLM):** Sử dụng các hàm liên kết (link functions) thích hợp để mô hình hóa dữ liệu có độ lệch.

*** Sự khác biệt giữa Độ phân tán (Dispersion) và Độ lệch (Skewness):**

Mặc dù độ phân tán và độ lệch có thể trông tương tự nhau, nhưng chúng đo lường những khía cạnh hoàn toàn khác nhau của phân phối dữ liệu.

Độ phân tán (Dispersion) phản ánh mức độ mà các điểm dữ liệu *phân tán xung quanh giá trị trung tâm* (trung bình hoặc trung vị). Nó giúp ta hiểu mức độ biến thiên của dữ liệu — tức dữ liệu có đồng nhất hay không.

Ngược lại, **Độ lệch (Skewness)** mô tả *hình dạng của phân phối* và hướng mà dữ liệu bị kéo dài (trái hoặc phải). Trong khi độ phân tán tập trung vào "mức độ lan rộng" của dữ liệu, thì độ lệch tập trung vào "mức độ bất đối xứng" của phân phối.

Dispersion	**Skewness**
Đo lường mức độ phân tán của dữ liệu quanh giá trị trung tâm (*mean, median*).	Đo lường hình dạng của phân phối và hướng lệch (trái hoặc phải).
Bao gồm các chỉ số: phương sai (*variance*), độ lệch chuẩn (*standard deviation*), phạm vi (*range*), và khoảng tứ phân vị (*interquartile range, IQR*).	Bao gồm các thước đo như hệ số lệch Pearson (*Pearson's coefficient of skewness*), moment skewness, và biểu đồ Q–Q (*Q–Q plots*).
Độ phân tán ảnh hưởng đến cách diễn giải giá trị trung bình nhưng không liên quan trực tiếp đến độ lệch.	Độ lệch thể hiện mối quan hệ giữa giá trị trung bình và trung vị.
Độ phân tán cao cho thấy các điểm dữ liệu phân tán rộng quanh giá trị trung tâm.	**Skewness dương:** đuôi bên phải dài hơn. **Skewness âm:** đuôi bên trái dài hơn. **Skewness bằng không:** phân phối đối xứng.
Giúp hiểu rõ mức độ biến thiên của dữ liệu.	Giúp nhận diện hình dạng và mức độ bất đối xứng của phân phối dữ liệu.
Ví dụ: điểm kiểm tra (*test scores*), biến động giá cổ phiếu, độ tuổi trong mẫu khảo sát.	Ví dụ: phân phối thu nhập (thường lệch phải), phân phối điểm thi (có thể lệch trái hoặc phải).

*** Kết luận:**

Bằng cách nắm vững khái niệm về *độ lệch (skewness)* và hiểu rõ cách đo lường nó, chúng ta có thể dễ dàng đánh giá hình dạng của phân phối dữ liệu, đưa ra các quyết định phân tích chính xác hơn, và lựa chọn các kỹ thuật xử lý dữ liệu phù hợp. Điều này đặc biệt quan trọng trong các quy trình *tiền xử lý dữ liệu (data preprocessing)* và *phân tích thống kê (statistical analysis)* trong học máy (Machine Learning), giúp cải thiện hiệu suất và độ tin cậy của mô hình.

3.1.5 Xử lý dữ liệu mất cân bằng trong bài toán phân loại [?]

3.1.5.1 Sơ lược về mất cân bằng dữ liệu (Imbalanced Data)

Một thành phần quan trọng trong các bài toán phân loại (classification) của học máy (machine learning) là xử lý **dữ liệu mất cân bằng (imbalanced data)**, được đặc trưng bởi **phân bố lớp bị lệch (skewed class distribution)** — trong đó một **lớp chiếm ưu thế (majority class)** có số lượng vượt trội so với các **lớp thiểu số (minority classes)**.

Thách thức mà sự mất cân bằng này gây ra là mô hình (model) có thể thể hiện **hiệu suất kém (poor performance)** do **thiên vị (bias)** về phía **lớp đa số (majority class)**. Trong các tình huống dữ liệu không đồng đều, mô hình thường có xu hướng **ưu tiên độ chính xác tổng thể (accuracy)** hơn là **khả năng nhận diện chính xác các mẫu thuộc lớp thiểu số (minority class recognition)**.

Vấn đề này có thể được giải quyết bằng cách áp dụng các **chiến lược chuyên biệt (specialized strategies)** như:

- **Tái lấy mẫu (Resampling)** — bao gồm **tăng mẫu lớp thiểu số (oversampling)** hoặc **giảm mẫu lớp đa số (undersampling)** hoặc sử dụng **kết hợp cả hai cách**;
- Sử dụng các **chỉ số đánh giá (evaluation metrics)** khác như **F1-score**, **precision**, và **recall**;
- Triển khai các **thuật toán nâng cao (advanced algorithms)** được thiết kế để hoạt động hiệu quả với **tập dữ liệu mất cân bằng (imbalanced datasets)**.

Dữ liệu mất cân bằng (Imbalanced data) đề cập đến các tập dữ liệu trong đó **phân bố của các quan sát (observations)** trong **nhãn mục tiêu (target class)** là **không đồng đều**. Nói cách khác, một **nhãn lớp (class label)** có số lượng quan sát cao hơn đáng kể, trong khi các nhãn khác có số lượng thấp hơn rõ rệt.

Khi một lớp có số lượng mẫu vượt trội so với các lớp khác trong một bài toán phân loại, dữ liệu được xem là **mất cân bằng (imbalanced)**. Các mô hình học máy (machine learning models) có thể trở nên **thiên lệch (biased)** trong quá trình dự đoán, có xu hướng **ưu tiên lớp đa số (majority class)**. Các kỹ thuật như **tăng mẫu lớp thiểu số (oversampling)** hoặc **giảm mẫu lớp đa số (undersampling)** được sử dụng trong **tái lấy mẫu (resampling)** để khắc phục vấn đề này.

Hơn nữa, có thể đánh giá hiệu suất mô hình (model performance) một cách chính xác hơn bằng cách thay thế thước đo **độ chính xác (accuracy)** bằng các **chỉ số đánh giá khác (assessment measures)** như **precision**, **recall** hoặc **F1-score**. Ngoài ra, để cải thiện hơn nữa việc xử lý **tập dữ liệu mất cân bằng (imbalanced datasets)** nhằm đạt được **kết quả dự đoán đáng tin cậy và công bằng hơn (reliable and equitable predictions)**, có thể áp dụng các kỹ thuật chuyên biệt (specialized techniques) như **phương pháp tập hợp (ensemble approaches)** hoặc **tạo dữ liệu tổng hợp (synthetic data generation)**.

3.1.5.2 Các cách xử lý dữ liệu mất cân bằng trong phân loại

Việc giải quyết **dữ liệu mất cân bằng (imbalanced data)** trong các bài toán **phân loại (classification)** là rất quan trọng để đảm bảo **hiệu suất mô hình công bằng (fair model performance)**. Các kỹ thuật phổ biến bao gồm **tái lấy mẫu (resampling)** — như **tăng mẫu (oversampling)** hoặc **giảm mẫu (undersampling)**, **tạo dữ liệu tổng hợp (synthetic data generation)**, **thuật toán chuyên biệt (specialized algorithms)**, và **chỉ số đánh giá thay thế (alternative evaluation metrics)**. Việc áp dụng các chiến lược này giúp mô hình đưa ra dự đoán chính xác hơn và ít thiên lệch hơn (more accurate and unbiased predictions) cho tất cả các lớp.

a) **Chỉ số đánh giá khác (Different Evaluation Metric)**

Độ chính xác của bộ phân loại (**classifier accuracy**) được tính bằng cách chia tổng số dự đoán đúng cho tổng số dự đoán — phù hợp với các lớp cân bằng, nhưng kém hiệu quả khi dữ liệu bị mất cân bằng. **Precision** đo độ chính xác của mô hình khi dự đoán một lớp cụ thể, trong khi **recall** đánh giá khả năng nhận diện đúng các mẫu của lớp đó.

Trong các tập dữ liệu mất cân bằng, **F1-score** trở thành thước đo được ưu tiên vì nó cân bằng giữa precision và recall, mang lại cái nhìn toàn diện hơn về hiệu suất mô hình. F1-score được biểu diễn như sau:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Khi mô hình dự đoán sai lớp thiểu số (tăng số lượng **false positives**), cả precision và F1-score đều giảm. Tương tự, nếu mô hình không nhận diện tốt lớp thiểu số (tăng **false negatives**), recall và F1-score cũng giảm. F1-score chỉ tăng khi mô hình cải thiện cả số lượng và độ chính xác của dự đoán.

Tóm lại, **F1-score** là một thống kê tổng hợp quan trọng, phản ánh **mối đánh đổi (trade-off)** giữa precision và recall — yếu tố then chốt trong việc đánh giá hiệu suất mô hình trên các tập dữ liệu mất cân bằng.

b) **Tái lấy mẫu (Resampling: Undersampling và Oversampling)**

Phương pháp này điều chỉnh sự cân bằng giữa lớp thiểu số và lớp đa số bằng cách **tăng mẫu (upsampling)** hoặc **giảm mẫu (downsampling)**. Với các tập dữ liệu mất cân bằng, **oversampling** được sử dụng để tăng số lượng mẫu của lớp thiểu số bằng cách lặp lại ngẫu nhiên các mẫu hiện có. Ngược lại, **undersampling** loại bỏ ngẫu nhiên các mẫu từ lớp đa số để làm cân bằng dữ liệu.

Cách tiếp cận này giúp tạo ra một **tập dữ liệu cân bằng (balanced dataset)**, đảm bảo cả hai lớp đều có mức đại diện tương đương, từ đó mô hình **học công bằng hơn (learns fairly)** trong quá trình huấn luyện.

c) **BalancedBaggingClassifier**

Khi làm việc với tập dữ liệu mất cân bằng, các bộ phân loại truyền thống thường có xu hướng ưu tiên **lớp đa số (majority class)** và bỏ qua **lớp thiểu số (minority class)** do tần suất xuất hiện thấp. **BalancedBaggingClassifier** — một mở rộng của các bộ phân loại trong **Scikit-learn** — giải quyết vấn đề này bằng cách thực hiện cân bằng dữ liệu trong quá trình huấn luyện (balancing during training).

Bộ phân loại này cung cấp các tham số như:

- `sampling_strategy`: xác định loại tái lấy mẫu (ví dụ: `'majority'` để chỉ lấy mẫu lại lớp đa số, `'all'` để tái lấy mẫu tất cả các lớp);
- `replacement`: quy định việc lấy mẫu có **thay thế (with replacement)** hay không.

BalancedBaggingClassifier đảm bảo **xử lý cân bằng và công bằng hơn giữa các lớp (equitable class treatment)**, đặc biệt hữu ích khi làm việc với **tập dữ liệu mất cân bằng (imbalanced datasets)**.

d) **SMOTH**

Kỹ thuật Tăng mẫu Lớp thiểu số Tổng hợp (Synthetic Minority Oversampling Technique - SMOTE) giải quyết vấn đề **tập dữ liệu mất cân bằng (imbalanced datasets)** bằng cách **tạo ra các mẫu tổng hợp (synthetic instances)** cho **lớp thiểu số (minority class)**.

Không giống như việc chỉ đơn giản **sao chép lại các bản ghi hiện có (duplicating records)**, SMOTE giúp **tăng tính đa dạng (enhance diversity)** bằng cách **tạo ra các mẫu nhân tạo (artificial instances)** mới.

Nói một cách đơn giản, SMOTE sẽ:

- Xem xét các mẫu thuộc lớp thiểu số;
- Chọn ngẫu nhiên một **láng giềng gần nhất (nearest neighbor)** bằng thuật toán **k-láng giềng gần nhất (k-nearest neighbors)**;
- Tạo ra một **mẫu tổng hợp (synthetic instance)** mới được đặt ngẫu nhiên trong **không gian đặc trưng (feature space)** giữa hai điểm này.

Kỹ thuật này giúp mô hình học máy có thêm dữ liệu đa dạng, cải thiện khả năng **nhận diện lớp thiểu số (minority class recognition)** mà không làm mất cân bằng tự nhiên của tập dữ liệu.

e) **Dịch chuyển ngưỡng - Threshold Moving**

Trong các **bộ phân loại (classifiers)**, các **dự đoán (predictions)** thường được biểu diễn dưới dạng xác suất thuộc về một lớp (probabilities of class membership). Ngưỡng mặc định (**default threshold**) để gán một dự đoán vào một lớp cụ thể thường được đặt là **0.5**.

Tuy nhiên, trong bối cảnh của các **bài toán có lớp mất cân bằng (imbalanced class problems)**, ngưỡng mặc định này có thể không mang lại kết quả tối ưu. Để **cải thiện hiệu suất của bộ phân loại (enhance classifier performance)**, việc **điều chỉnh ngưỡng (adjusting the threshold)** là cần thiết nhằm đạt được sự **phân biệt hiệu quả (efficient discrimination)** giữa hai lớp.

Các kỹ thuật như:

- **Đường cong ROC (ROC Curves)** và **đường cong Precision–Recall (Precision–Recall Curves)** — được sử dụng để xác định **ngưỡng tối ưu (optimal threshold)**;
- **Tìm kiếm theo lưới (Grid Search)** hoặc khám phá trong một dải giá trị cụ thể (exploration within a range of values) — có thể được áp dụng để tìm ra **ngưỡng phù hợp nhất (most suitable threshold)** cho bộ phân loại.

Việc tinh chỉnh ngưỡng này giúp mô hình đạt được sự cân bằng tốt hơn giữa **độ chính xác (precision)** và **độ bao phủ (recall)**, đặc biệt trong các tập dữ liệu có sự chênh lệch lớn giữa các lớp.

3.2 Gradient Boosting

3.2.1 Sơ lược về giải thuật Gradient Boosting [?]

Gradient Boosting là một thuật toán tăng cường và ở đây mỗi mô hình mới được đào tạo để giảm thiểu hàm tổn thất, chẳng hạn như sai số bình phương trung bình hoặc entropy chéo của mô hình trước đó bằng cách sử dụng **Gradient Descent** (giảm độ dốc). Trong mỗi lần lặp, thuật toán tính toán gradient của hàm mất liên quan đến các dự đoán và sau đó đào tạo một mô hình yếu mới để giảm thiểu gradient này. Các dự đoán của mô hình mới sau đó được thêm vào tổng hợp (dự đoán tất cả các mô hình) và quá trình được lặp lại cho đến khi đáp ứng tiêu chí dừng.

Trong đó, **Gradient Descent** là xương sống của quá trình học tập cho các thuật toán khác nhau, bao gồm hồi quy tuyến tính, hồi quy logistic, máy vectơ hỗ trợ và mạng nơ-ron đóng vai trò như một kỹ thuật tối ưu hóa cơ bản để giảm thiểu hàm chi phí của mô hình bằng cách **điều chỉnh lặp đi lặp lại các tham số mô hình để giảm sự khác biệt giữa giá trị dự đoán và giá trị thực tế, cải thiện hiệu suất của mô hình.**

a) Độ co ngót và độ phức tạp của mô hình (Shrinkage & Model Complexity)

Một tính năng chính của Gradient Boosting là thu nhỏ quy mô đóng góp của mỗi mô hình mới bằng cách sử dụng **tốc độ học tập - learning_rate** (được ký hiệu là η).

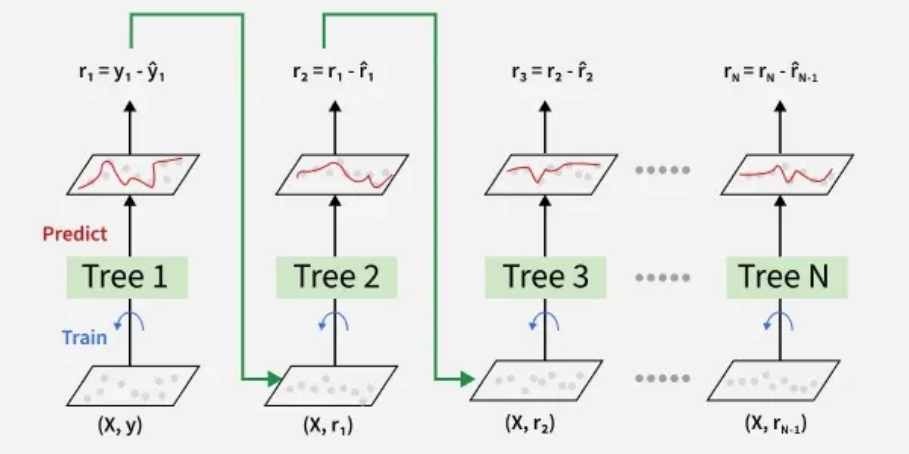
- **Tỷ lệ học tập nhỏ hơn:** có nghĩa là sự đóng góp của mỗi cây nhỏ hơn, điều này làm giảm nguy cơ quá khớp nhưng cần nhiều cây hơn để đạt được hiệu suất tương tự.
- **Tỷ lệ học tập lớn hơn:** có nghĩa là mỗi cây có tác động đáng kể hơn nhưng điều này có thể dẫn đến quá khớp.

Có một sự đánh đổi giữa tốc độ học tập và số lượng estimators (cây), tốc độ học nhỏ hơn thường có nghĩa là cần nhiều cây hơn để đạt được hiệu suất tối ưu.

b) Cơ chế hoạt động của Gradient Boosting

Cơ chế học máy của giải thuật **Gradient Boosting** là quy trình học tuần tự thông qua một **quần thể (ensemble)** bao gồm nhiều cây với **mỗi cây sẽ sửa lỗi cho cây trước đó**. Trong **lần lặp (iteration)** đầu tiên, **Cây 1** được đào tạo trên dữ liệu gốc *x* và các nhãn hiệu (label) *y* thực sự. Nó đưa ra dự đoán được sử dụng để tính toán lỗi.

Trong lần lặp thứ hai, **Cây 2** được đào tạo bằng ma trận tính năng *x* và các lỗi từ Cây 1 dưới dạng nhãn. Điều này có nghĩa là Cây 2 được đào tạo để **dự đoán sai số** của Cây 1. Quá trình này tiếp tục đối với tất cả các cây trong quần thể. Mỗi cây tiếp theo được đào tạo để dự đoán sai số của cây trước đó.



Hình 3.4: *Cây tăng cường độ dốc*

Sau khi mỗi cây được huấn luyện, các dự đoán của nó được thu nhỏ bằng cách nhân chúng với tốc độ học η nằm trong khoảng từ 0 đến 1. Điều này ngăn chặn tình trạng quá khớp bằng cách đảm bảo mỗi cây có tác động nhỏ hơn đến mô hình cuối cùng.

Khi tất cả các cây được đào tạo, các dự đoán được thực hiện bằng cách tổng đóng góp của tất cả các cây. Dự đoán cuối cùng được đưa ra theo công thức:

$$y_{pred} = y_1 + \eta * r_1 + \eta * r_2 * ... * \eta * r_N$$

Trong đó, $r_1, r_2, ..., r_N$ là sai số được dự đoán ở mỗi cây.

Gradient Boosting là một kỹ thuật máy học hiệu quả và được sử dụng rộng rãi cho cả các vấn đề phân loại và hồi quy. Nó xây dựng các mô hình tuần tự tập trung vào việc sửa lỗi do các mô hình trước đó gây ra, dẫn đến cải thiện hiệu suất. Giải thuật bao gồm các thông số:

- **n_estimators:** Số lượng cây (ước tính) sẽ được xây dựng. Giá trị cao hơn thường cải thiện hiệu suất mô hình nhưng làm tăng thời gian tính toán.
- **learning_rate:** Tốc độ học của mô hình.
- **random_state:** Đảm bảo khả năng tái tạo của kết quả. Đặt giá trị cố định cho random_state đảm bảo rằng bạn nhận được kết quả giống nhau mỗi khi chạy mô hình.
- **max_features: Tham số này giới hạn số lượng yếu tố (features) mà mỗi cây có thể sử dụng để tách. Nó giúp ngăn chặn quá khớp bằng cách hạn chế độ phức tạp của từng cây và thúc đẩy sự đa dạng trong mô hình.**

3.2.2 Cơ sở toán học cho Gradient Boosting [?]

Khi thực hiện khảo sát trên 1 tập dữ liệu $\mathscr{D} = \{(x_k, y_k)\}_{k=1..n}$ với x_k là vector ngẫu nhiên của m features và $y \in R$ là giá trị đích (nhãn) - y có thể mang giá trị định lượng biến thiên liên tục trên tập số thực hoặc giá trị phân loại định tính.

Với cặp giá trị (x_k, y_k) độc lập và được phân phối dựa trên một phân phối chưa biết $P(.,.)$, mục tiêu của tác vụ học là huấn luyện hàm $F : R^m \to R$ nhằm tối thiểu hóa giá trị mất mát kì vọng $\mathscr{L}(F) := E(L(y, F(x)))$. Trong đó, $L(.,.)$ là một hàm mất mát liên tục và cặp (x, y) là một cặp mẫu thử lấy từ P và độc lập với bộ dữ liệu huấn luyện $\mathscr{D}$.

Thủ tục Gradient Boosting xây dựng lặp lại theo từng bước dãy các giá trị xấp xỉ dần tiến gần đến nghiệm (solution) thật $F : R^m \to R$, $t = 0, 1, ...$ theo cách tiếp cận tham lam - tối ưu cục bộ. Giá trị thứ F^t được tính từ giá trị xấp xỉ trước F^{t-1} bằng cách cộng dồn các giá trị trước đó: $F^t = F^{t-1} + \alpha h^t$, với giá trị α là kích thước bước nhảy và hàm $h^t : R^m \to R$ (base predictor) được chọn là một họ các hàm H được dùng để tối thiểu hóa giá trị mất mát kì vọng:

$$h^t = \arg\min_{h \in H} \mathscr{L}(F^{t-1} + h) = \arg\min_{h \in H} \mathbb{E}L\big(y, F^{t-1}(\mathbf{x}) + h(\mathbf{x})\big). \tag{3.4}$$

Bài toán tối thiểu hóa thường được tiếp cận bằng phương pháp Newton sử dụng xấp xỉ bậc 2 của hàm $\mathscr{L}(F^{t-1} + h^t)$ tại F^{t-1} hoặc sử dụng bước nhảy gradient (chiều giảm). Cả hai phương pháp đều là các dạng của functional gradient descent. Gradient bước h^t được chọn làm cách để $h^t(x)$ xấp xỉ $-g^t(x, y)$ với $g^t(\mathbf{x}, y) := \frac{\partial L(y,s)}{\partial s}\Big|_{s=F^{t-1}(\mathbf{x})}$.
Tại đây, h^t được tính toán thông qua sấp xỉ bình phương tối thiểu:

$$h^t = \arg\min_{h \in H} \mathbb{E}(-g^t(\mathbf{x}, y) - h(\mathbf{x})^2). \tag{3.5}$$

3.3 Categorical Boosting (Mô hình CatBoost) [?]

3.3.1 Thuật toán CatBoost

Catboost (tăng cường phân loại) là một thư viện máy học mã nguồn mở mạnh mẽ được thiết kế đặc biệt để xử lý các tính năng phân loại và thúc đẩy cây quyết định. Được phát triển bởi Yandex, CatBoost nổi bật với khả

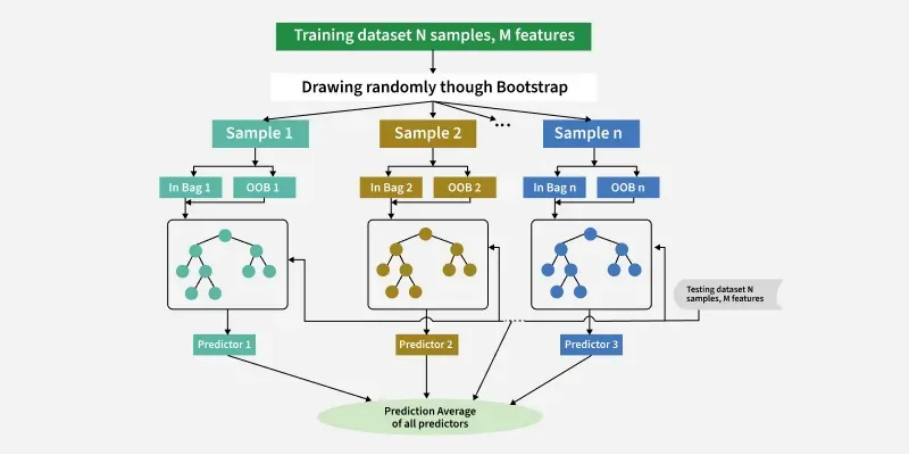
năng làm việc hiệu quả với các biến phân loại mà không cần tiền xử lý rộng rãi. Thuật toán này đã trở nên phổ biến do tính mạnh mẽ, hiệu suất cao và dễ sử dụng trong các tác vụ máy học khác nhau.

Catboost dựa trên khái niệm kỹ thuật **Gradient Boosting** trong đó cây quyết định được xây dựng tuần tự để giảm thiểu lỗi và cải thiện dự đoán. Trong mô hình này, cây quyết định được xây dựng dựa trên phân hoạch đệ quy của không gian đặc trưng R^m thành các khu vực tách biệt (Các node của cây) dựa theo giá trị của một vài *thuộc tính chia tách a*. Các thuộc tính thường là các biến nhị phân để định nghĩa các đặc trưng x^k vượt qua ngưỡng t ($a = \mathbf{1}_{\{x^k > t\}}$) tại x^k với giá trị số học hoặc nhị phân trong trường hợp đạt ngưỡng $t = 0.5$. Tại vùng cuối cùng (lá của cây) được gán một giá trị là một ước lượng phản hồi y trong khu vực cho các tác vụ hồi quy hoặc dự đoán lớp nhãn trong bài toán phân loại. Theo cách này, cây quyết định h có thể được biểu diễn như sau:

$$h(\mathbf{x}) = \sum_{j=1}^{J} b_j \mathbf{1}_{\{\mathbf{x} \in R_j\}} \tag{3.6}$$

Trong đó, R_j là các phân vùng biệt lập tương đồng với các node lá của cây.

Quá trình của CatBoost hoạt động bằng cách xây dựng một cây quyết định và đánh giá mức độ sai số trong các dự đoán. Khi cây đầu tiên được xây dựng, cây tiếp theo được tạo ra để sửa lỗi của cây trước. Quá trình này tiếp tục lặp đi lặp lại với mỗi cây mới tập trung vào việc cải thiện dự đoán của mô hình bằng cách giảm các lỗi trước đó, quá trình này tiếp tục cho đến khi đáp ứng số lần lặp lại được xác định trước. Kết quả là một tập hợp các cây quyết định hoạt động cùng nhau để đưa ra dự đoán chính xác.



Hình 3.5: *Mô hình CatBoost*

Nó đặc biệt phù hợp với các bộ dữ liệu quy mô lớn với nhiều tính năng độc lập. Không giống như các thuật toán tăng cường gradient khác, CatBoost được thiết kế đặc biệt để xử lý liền mạch cả tính năng phân loại và số mà không yêu cầu mã hóa tính năng thủ công.

CatBoost được ứng dụng cả trong bài toán **Hồi quy - Regressor**, **Phân loại - Classifier** và **Xếp hạng - Ranking**.

3.3.2 Biến phân loại và Target Statistics

Một đặc trưng dạng phân loại (categorical feature) là đặc trưng có tập giá trị rời rạc (gọi là các category) và các giá trị này không thể so sánh trực tiếp với nhau. Một kỹ thuật phổ biến để xử lý các đặc trưng dạng phân loại trong các mô hình boosted trees là one-hot encoding; đối với mỗi giá trị của một thuộc tính phân loại (category), ta sẽ thêm một đặc trưng nhị phân (binary feature) để biểu thị sự xuất hiện của giá trị đó. Tuy nhiên, trong trường hợp các đặc trưng phân loại có số lượng giá trị lớn thì kỹ thuật như vậy (one-hot encoding) có thể dẫn đến số lượng đặc trưng mới cực kỳ lớn và khó khả thi.

Để giải quyết vấn đề này, người ta có thể gom các giá trị phân loại (categories) vào một số cụm (clusters) giới hạn, sau đó mới áp dụng one-hot encoding. Một phương pháp phổ biến là gom nhóm các category dựa trên thống kê của nhãn (target statistics – TS), tức là ước lượng giá trị kỳ vọng của nhãn (target) trong từng category.

Thống kê nhãn là phương pháp xử lý đặc trưng dạng phân loại i bằng cách Thay thế giá trị phân loại x_k^i của mẫu huấn luyện thứ k bằng một đặc trưng số có giá trị bằng một thống kê của nhãn (target statistic – TS) $\hat{x}_k^i$. Thông thường, nó ước lượng giá trị kỳ vọng của nhãn y có điều kiện theo từng giá trị category $\hat{x}_k^i \approx \mathbb{E}(y \mid x^i = x_k^i)$.

* Greedy Target Statistics - Thống kê nhãn tham lam

Đây là một cách tiếp cận đơn giản, trực tiếp để ước lượng $\mathbb{E}(y|x^i = x_k^i)$ làm giá trị trung bình của y trên tập huấn luyện mẫu với cùng phân loại x_k^i. Ước lượng này thường có nhiều nhiễu (noisy) đối với các category xuất hiện ít lần (low-frequency categories), và người ta thường làm trơn (smooth) nó bằng cách sử dụng một prior p:

$$\hat{x}_k^i = \frac{\sum_{j=1}^{n} \mathbf{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{j=1}^{n} \mathbf{1}_{\{x_j^i = x_k^i\}} + a} \tag{3.7}$$

Trong đó, $a > 0$ là một tham số. Một thiết lập thông thường cho p là giá trị nhãn trung bình trong tập dữ liệu.

Vấn đề của cách tiếp cận tham lam là rò rỉ nhãn: đặc trưng x_k^i được tính toán sử dụng nhãn y_k - nhãn của $\mathbf{x}_k$. Điều này dẫn đến sự dịch chuyển có điều kiện: phân phối của $\hat{x}^i|y$ sẽ khác nhau giữa các mẫu huấn luyện và mẫu kiểm tra.

Ví dụ sau đây sẽ mô tả sự ảnh hưởng nghiêm trọng của phương pháp này đến sai tố tổng quát hóa của mô hình học. Cho đặc trưng thứ i là dạng phân loại, các giá trị của nó là duy nhất và với mỗi phân loại A, ta có $P(y = 1|x^i = A) = 0.5$ cho tác vụ phân loại. Sau đó, bộ dữ liệu huấn luyện $\hat{x}^i = \frac{y_k + ap}{1+a}$ vì vậy nó chỉ phù hợp để thực hiện 1 phân tách tại ngưỡng $t = \frac{0.5+ap}{1+a}$ để phân loại các mẫu huấn luyện một cách hoàn hảo. Tuy nhiên, với tất cả các mẫu thử, giá trị của thống kê nhãn tham lam là p, giá trị mô hình dự đoán là 0 trên toàn mẫu thử khi $p < t$ và 1 trong trường hợp ngược lại vì vậy dẫn đễ độ chính xác là 0.5 cho cả 2 trường hợp. Để giải quyết vấn đề này, một tiêu chí mong muốn được đưa vào thống kê nhãn:

P1 $\mathbb{E}(\hat{x}^i|y = v) = \mathbb{E}(\hat{x}_k^i|y = v)$ tại $(\mathbf{x_k}, \mathbf{y_k})$ là mẫu huấn luyện thứ k.

Ví dụ trên dẫn đến kết quả $\mathbb{E}(\hat{x}^i|y = v) = \frac{y_k + ap}{1+a}$ và $\mathbb{E}(\hat{x}^i|y) = p$, hai giá trị này là khác nhau.

Nhằm khắc phục sự thay đổi có điều kiện trên, ý tưởng được đưa ra là tính toán thống kê nhãn cho $\mathbf{x}_k$ trên tập con của mẫu $\mathscr{D}_k \subset \mathscr{D} \setminus \{\mathbf{x}_k\}$:

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathscr{D}_k} \mathbf{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{\mathbf{x}_j \in \mathscr{D}_k} \mathbf{1}_{\{x_j^i = x_k^i\}} + a} \tag{3.8}$$

* Holdout Target Statistics

Cách thức của phương pháp này là phân hoạch bộ dữ liệu huấn luyện thành 2 phần $\mathscr{D} = \hat{\mathscr{D}}_0 \sqcup \hat{\mathscr{D}}_1$ và sử dụng $\mathscr{D}_k = \hat{\mathscr{D}}_0$ để tính thống kê nhãn theo công thức số **(5)** và $\hat{\mathscr{D}}_1$ để huấn luyện. Mặc dù cách tiếp cận Holdout Target Statistics (TS) thoả mãn tính chất P1, nhưng nó lại làm giảm đáng kể lượng dữ liệu được sử dụng cả cho việc huấn luyện mô hình lẫn cho việc tính toán TS. Do đó, nó vi phạm tính chất:

P2 *Sử dụng hiệu quả toàn bộ dữ liệu huấn luyện để tính toán thống kê nhãn đặc trưng và để huấn luyện mô hình.*

* Leave-one-out Target Statistics

Ban đầu có thể kỹ thuật leave-one-out sẽ hoath động tốt với phương pháp thực hiện: Lấy $\mathscr{D}_k = \mathscr{D} \setminus \mathbf{x}_k$ sử dụng cho mẫu dữ liệu huấn luyện, trong khi đó $\mathbf{x}_k$ và $\mathscr{D}_k = \mathscr{D}$ cho mẫu thử. Tuy nhiên, phương pháp này không ngăn chặn được vấn đề rò rỉ nhãn. Điều này dẫn đến giá trị hằng của đặc trưng dạng phân loại: $\hat{x}_k^i = A$ đối với mọi mẫu. Khi đặt n^+ làm số lượng mẫu với $y = 1$ dẫn đến $\hat{x}_k^i = \frac{n^+ - y_k + ap}{n-1+a}$ và có thể phân loại một cách hoàn hảo bộ dữ liệu huấn luyện bằng cách tạo phân tách với ngưỡng $t = \frac{n^+ - 0.5 + ap}{n-1+a}$.

* Ordered Target Statistics

Mô hình CatBoost sử dụng chiến lược hiệu quả hơn dựa trên nguyên tắc sắp xếp thứ tự. Trong phương pháp Ordered Target Statistics, với mỗi mẫu, tất cả "lịch sử" khả dụng được sử dụng để tính toán thống kê nhãn.

Ví dụ, ta sử đụng $\mathscr{D}_k = \mathbf{x}_j : \sigma(j) < \sigma(k)$ (σ là hoán vị ngẫu nhiên của bộ huấn luyện) áp dụng vào phương trình số (5) cho mẫu huấn luyện và $\mathscr{D}_k = \mathscr{D}$ cho mẫu thử. Kết quả của phương pháp này thỏa mãn yêu cầu P1 và

đồng thời cho phép sử dụng toàn bộ dữ liệu huấn luyện cho việc học mô hình (P2). Lưu ý rằng, nếu chỉ sử dụng một phép hoán vị ngẫu nhiên duy nhất, thì các mẫu đứng trước sẽ có giá trị thống kê nhãn với phương sai cao hơn nhiều so với các mẫu đứng sau. Để khắc phục điều này, CatBoost sử dụng nhiều phép hoán vị khác nhau cho các bước khác nhau của quá trình gradient boosting.

3.3.3 Độ lệch dự đoán (Prediction Shift) và Boosting có thứ tự (Ordered Boosting)

a) Độ lệch dự đoán

Như các phương pháp tính thống kê nhãn, độ lệch dự đoán được gây ra bởi một dạng đặc biệt của rò rỉ nhãn. Giải pháp cho vấn đề trên là Boosting được sắp xếp thứ tự tương tự như phương pháp thống kê nhãn có thứ tự. Trong bài toán này, kì vọng được lấy xấp xỉ sử dụng bộ dữ liệu $\mathscr{D}$ giống nhau:

$$h^t = \arg\min_{h \in H} \frac{1}{n} \sum_{k=1}^{n} (-g^t(\mathbf{x}_k, y_k) - h(\mathbf{x}_k))^2 \tag{3.9}$$

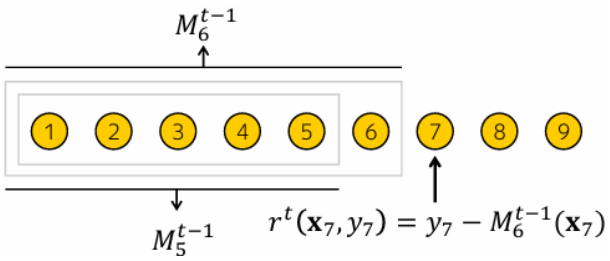
Trong đó, chuỗi dịch chuyển được mô tả như sau:

- Phân bố có điều kiện của gradient $g^t(\mathbf{x}_k, y_k)|\mathbf{x}_k$ (xét đến tính ngẫu nhiên của $\mathscr{D} \setminus \{\mathbf{x}_k\}$) bị dịch chuyển so với phân phối đó trên một mẫu kiểm thử $g^t(\mathbf{x}, y)|\mathbf{x}$.
- Lần lượt, bộ dự đoán cơ sở h^t được định nghĩa theo phương trình số (6) bị chệch so với nghiệm của phương trình số (2).
- Cuối cùng, điều này gây ảnh hưởng đến khả năng khái quát hóa của mô hình được huấn luyện F^t

Những vấn đề này dẫn đến rò rỉ nhãn. Chính vì vậy, gradients sử dụng ở mỗi bước được ước lượng bằng giá trị nhãn của bộ dữ liệu tương tự mà mô hình F^{t-1} hiện tại được xây dựng trên đó. Tuy nhiên, phân bố có điều kiện $F^{t-1}(\mathbf{x}_k)|\mathbf{x}_k$ cho mẫu huấn luyện $\mathbf{x}_k$ bị lệch, về tổng quát, từ phân bố $F^{t-1}(\mathbf{x})|\mathbf{x}$ cho mẫu thử $\mathbf{x}$. Đây được gọi là ***độ lệch dự đoán***.

b) Boosting có sắp xếp thứ tự

Giả sử mô hình được huấn luyện với I cây. Để tạp phần dư $\gamma^{I-1}(\mathbf{x}_k, y_k)$ không bị chệch, mô hình F^{I-1} cần được huấn luyện mà không sử dụng mẫu $\mathbf{x}_k$. Vì kết quả cần thu được phần dư không chệch cho tất cả các mẫu huấn luyện, không mẫu nào được sử dụng để huấn luyện F^{I-1} - điều mà có vẻ mấy khả thi cho quá trình huấn luyện. Tuy nhiên có thể duy trì tập các mô hình khác nhau dựa trên các mẫu được dùng để huấn luyện chúng. Khi đó, để tính phần dư (residual) của một mẫu, ta sử dụng mô hình được huấn luyện mà không bao gồm mẫu đó. Để xây dựng tập mô hình như vậy, ta có thể sử dụng nguyên tắc sắp xếp (ordering principle) đã được áp dụng trước đó cho thống kê nhãn. Nhằm biểu diễn ý tưởng của của phương pháp này, giả sử lấy một hoán vị ngẫu nhiên σ từ mẫu huấn luyện và duy trì n mô hình hỗ trợ khác nhau $M_1, M_2, ..., M_n$ với mô hình M_i được học bằng cachs chỉ sử dụng i mẫu đầu tiên trong hoán vị. Tại mỗi bước, để thu được phần dư cho mẫu thứ i, mô hình M_{j-1}.



Hình 3.6: *Nguyên tắc sắp xếp*

Thuật toán thu được cuối cùng được gọi là ordered boosting. Tuy nhiên, thuật toán này không khả thi trong hầu hết các tác vụ thực tế, do cần phải huấn luyện n mô hình khác nhau, khiến độ phức tạp tính toán và yêu cầu bộ nhớ tăng lên gấp n lần. Trong CatBoost, một phiên bản sửa đổi của thuật toán này được triển khai, dựa trên thuật toán gradient boosting với cây quyết định làm bộ dự đoán cơ sở (GBDT).

Algorithm 1: Ordered boosting

input : $\{(\mathbf{x}_k, y_k)\}_{k=1}^{n}$, I;

$\sigma \leftarrow$ random permutation of $[1, n]$;
$M_i \leftarrow 0$ for $i = 1..n$;
for $t \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** n **do**
    $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$;
  for $i \leftarrow 1$ **to** n **do**
    $\Delta M \leftarrow$
     $LearnModel((\mathbf{x}_j, r_j) :$
     $\sigma(j) \leq i)$;
    $M_i \leftarrow M_i + \Delta M$;
return M_n

Hình 3.7: *Thuật toán ordering boosting*

* Ordered boosting với các đặc trưng dạng phân loại

Ở các mục 3.2.2 và 3.2.3, các hoán vị ngẫu nhiên σ_{cat} và σ_{boost} của mẫu huấn luyện lần lượt được sử dụng cho việc tính toán thống kê nhãn và boosting có sắp xếp thứ tự. Khi thực hiện kết hợp cả 2 hoán vị trên vào thuật toán, cần lấy $\sigma_{cat} = \sigma_{boost}$ để tránh sai lệch dự đoán. Điều này nhằm đảm bảo nhãn y_i không được sự dụng trong việc huấn luyện mô hình M_i kể cả trong việc tính toán thống kê nhãn và ước lượng gradient để đảm bảo được yêu cầu về mặt lý thuyết.

3.3.4 Hiện thực mô hình CatBoost

CatBoost có 2 chế độ boosting là Ordered và Plain. Chế độ Plain là thuật toán GBDT tiêu chuẩn đã được tích hợp sẵn thống kê nhãn có thứ tự trong khi đó chế độ Ordered thể hiện một biến thể hiệu quả của Thuật toán Ordering Boosting.

Khi bắt đầu, CatBoost thực hiện $s+1$ hoán vị ngẫu nhiên độc lập của bộ dữ liệu huấn luyện. Các hoán vị $\sigma_1, ..., \sigma_s$ được sử dụng để đánh giá các phép chia tách nhằm xác định kiến trúc cây, trong khi đó σ_0 phục vụ cho việc chọn các giá trị lá b_j của các cây. Trong những mẫu dữ liệu với lịch sử ngắn trong các hoán vị được cho, cả giá trị thống kê nhãn lẫn dự đoán được sử dụng trong ordered boosting ($M_{\sigma(i)-1}(\mathbf{x}_i)$) có phương sai cao. Vì vậy, việc chỉ sử dụng một hoán vị có thể làm tăng phương sai của kết quả cuối cùng mà mô hình dự đoán, trong khi đó việc sử dụng nhiều hoán vị cho phép làm giảm mức độ ảnh hưởng trong cách mô tả về sau.

Trong CatBoost, hệ dự đoán cơ sở là các cây quyết định hay còn được gọi là bảng quyết định. Các node trên cùng 1 tầng cây sẽ được áp dụng các điều kiện chia tách giống nhau. Những cây mang tính chất trên thì cân bằng, ít khả năng bị overfitting hơn và cho phép tăng tốc quá trình thực thi tại thời điểm kiểm thử. Thủ tục xây dựng cây trong CatBoost được mô tả như sau:

Algorithm 2: Building a tree in CatBoost

input : $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}, \alpha, L, \{\sigma_i\}_{i=1}^{s}, Mode$

$grad \leftarrow CalcGradient(L, M, y)$;
$r \leftarrow random(1, s)$;
if $Mode = Plain$ **then**
  $G \leftarrow (grad_r(i)$ for $i = 1..n)$;
if $Mode = Ordered$ **then**
  $G \leftarrow (grad_{r,\sigma_r(i)-1}(i)$ for $i = 1..n)$;
$T \leftarrow$ empty tree;
foreach *step of top-down procedure* **do**
  foreach *candidate split* c **do**
    $T_c \leftarrow$ add split c to T;
    if $Mode = Plain$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_r(p)$ for $p : leaf_r(p) = leaf_r(i))$ for $i = 1..n$;
    if $Mode = Ordered$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_{r,\sigma_r(i)-1}(p)$ for $p : leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i))$ for $i = 1..n$;
    $loss(T_c) \leftarrow \cos(\Delta, G)$
  $T \leftarrow \arg\min_{T_c}(loss(T_c))$
if $Mode = Plain$ **then**
  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha\,\text{avg}(grad_{r'}(p)$ for $p : leaf_{r'}(p) = leaf_{r'}(i))$ for $r' = 1..s, i = 1..n$;
if $Mode = Ordered$ **then**
  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha\,\text{avg}(grad_{r',j}(p)$ for $p : leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \le j)$ for $r' = 1..s$, $i = 1..n$, $j \ge \sigma_{r'}(i) - 1$;
return T, M

Hình 3.8: *Giải thuật xây dựng cây trong CatBoost*

Trong chế độ Ordered Boosting, trong suốt quá trình học, các mô hình hỗ trợ $M_{r,j}$ được duy trì với $M_{r,j}(i)$ là dự đoán hiện tại cho mẫu thứ i dựa trên j mẫu đầu tiên trong hoán vị σ_r. Tại mỗi lần lặp t của giải thuật, một hoán vị ngẫu nhiên σ_r được lấy mẫu từ $\{\sigma_1, ..., \sigma_s\}$ và xây dựng cây T_t dựa trên hoán vị đó. Đầu tiên, với các đặc trưng dạng phân loại, tất cả giá trị thống kê nhãn được tính toán được tính theo hoán vị này. Tiếp đó, hoán vị gây ảnh hưởng lên thủ tục học dạng cây.

3.3.5 Các tham số chung của mô hình CatBoost

Mô hình CatBoost được thiết lập với các thông số dưới đây:

Tham số	Kiểu dữ liệu / Default	Ý nghĩa
iterations	`int`, mặc định = 1000	Số lượng cây (boosting rounds). Giá trị càng lớn, mô hình càng mạnh nhưng dễ overfitting.
learning_rate	`float`, mặc định = None (CatBoost tự chọn)	Tốc độ học, thường chọn trong khoảng 0.01–0.3. Nhỏ thì học chậm nhưng ổn định, lớn thì hội tụ nhanh nhưng dễ overfit.
depth	`int`, mặc định = 6	Độ sâu của mỗi cây. Thường chọn trong khoảng 4–10. Sâu hơn → mô hình phức tạp hơn nhưng dễ overfitting.
l2_leaf_reg	`float`, mặc định = 3.0	Hệ số regularization L2 trên lá cây. Giúp giảm overfitting. Giá trị lớn → regularization mạnh hơn.

Tham số	Kiểu dữ liệu / Default	Ý nghĩa
loss_function	`str`, mặc định: 'Logloss' (classification), 'RMSE' (regression)	Hàm mất mát chính: – Regression: RMSE, MAE, Quantile, Huber – Classification: Logloss, CrossEntropy – Ranking: YetiRank, PairLogit
custom_loss	`list of str`	Các metric phụ để tính trong quá trình huấn luyện (không ảnh hưởng training). Ví dụ: ["AUC", "Accuracy"].
bootstrap_type	`str`, mặc định = "Bayesian"	Loại sampling: "Bayesian", "Bernoulli", "Poisson", "No".
bagging_temperature	`float`	Kiểm soát mức độ random trong sampling (chỉ dùng khi bootstrap_type = Bayesian).
rsm	`float`, mặc định = 1.0	Random Subspace Method, tỷ lệ feature được chọn cho mỗi cây.
nan_mode	`str`, mặc định = "Min"	Cách xử lý giá trị thiếu: "Min", "Max", "Forbidden".
cat_features	`list of int/str`	Danh sách các cột dạng categorical (CatBoost sẽ tự mã hóa).
one_hot_max_size	`int`, mặc định = None	Nếu biến categorical có số lượng giá trị $\leq$ threshold thì mã hóa one-hot.
text_features	`list of int/str`	Danh sách cột dữ liệu dạng text.
ignored_features	`list of int/str`	Các feature bị bỏ qua trong quá trình huấn luyện.
eval_metric	`str`, mặc định = None	Metric chính để đánh giá, ví dụ: "AUC", "Accuracy", "RMSE"...
early_stopping_rounds	`int`, mặc định = None	Dừng sớm nếu metric trên validation không cải thiện sau n vòng.
use_best_model	`bool`, mặc định = False	Sử dụng model tốt nhất (trên tập validation) sau khi dừng huấn luyện.
random_seed	`int`, mặc định = None	Đặt seed để tái lập kết quả.
verbose	`int/bool`, mặc định = True	Tần suất log kết quả.
thread_count	`int`, mặc định = -1	Số luồng CPU sử dụng. -1 nghĩa là dùng tất cả.
task_type	`str`, mặc định = "CPU"	Loại thiết bị huấn luyện: "CPU" hoặc "GPU".
devices	`str`, mặc định = None	Chỉ định GPU nào để train (nếu task_type = GPU).

3.4 Light Gradient Boosting Machine (LightGBM) [?]

3.4.1 Sơ lược về LightGBM

LightGBM được xem là phiên bản cải tiến của thuật toán Gradient Boosting truyền thống, tập trung vào việc tăng tốc độ huấn luyện và giảm tiêu thụ bộ nhớ, nhờ việc sử dụng các kỹ thuật tối ưu như:

- **Histogram-based algorithm**: thay thế phương pháp pre-sort-based thông thường, giúp tăng tốc độ tìm điểm chia (split point) trong quá trình xây dựng cây và tiết kiệm bộ nhớ.
- **Gradient-based One-Side Sampling (GOSS) và Exclusive Feature Bundling (EFB)**: hai thuật toán cốt lõi giúp giảm khối lượng tính toán mà vẫn đảm bảo độ chính xác của mô hình.

Không chỉ nổi bật về hiệu suất, LightGBM còn hỗ trợ các bài toán phân loại và hồi quy, có khả năng xử lý dữ liệu cực lớn, hỗ trợ tính toán song song và phân tán, cùng khả năng tùy chỉnh tham số linh hoạt. Nhờ đó, LightGBM trở thành lựa chọn hàng đầu cho các hệ thống dự đoán, phân tích dữ liệu tài chính, y tế, tiếp thị, và nhiều lĩnh vực khác trong học máy hiện đại.

3.4.2 Cơ sở toán học cho LightGBM

a) GBDT và phân tích độ phức tạp

GBDT (Gradient Boosting Decision Tree) là mô hình tổ hợp nhiều cây quyết định (decision trees) được huấn luyện nối tiếp nhau. Ở mỗi vòng lặp, mô hình huấn luyện một cây mới để xấp xỉ negative gradient của hàm mất mát đối với đầu ra hiện tại của mô hình. Về bản chất, GBDT thực hiện một dạng tối ưu hoá theo gradient trong không gian hàm.

Giả sử ta có tập dữ liệu huấn luyện $\{(x_i, y_i)\}_{i=1}^{n}$, mô hình dự đoán tại vòng lặp t là:

$$\hat{y}_i^{(t)} = F_{t-1}(x_i) + f_t(x_i),$$

trong đó F_{t-1} là mô hình tổ hợp của các cây đã học trước đó, và f_t là cây mới được huấn luyện tại vòng t.
Cây f_t được học để xấp xỉ hướng gradient âm của hàm mất mát $L(y, \hat{y})$, tức là:

$$g_i^{(t)} = -\frac{\partial L(y_i, F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}.$$

Do đó, tại mỗi vòng, bài toán của GBDT trở thành bài toán hồi quy:

$$f_t = \arg\min_f \sum_{i=1}^{n} \left(g_i^{(t)} - f(x_i)\right)^2.$$

Nói cách khác, ta phải xây dựng cây sao cho giá trị dự đoán của các lá xấp xỉ tốt nhất các giá trị $g_i^{(t)}$.

Trong quá trình huấn luyện cây phần tốn nhiều chi phí nhất là tìm điểm chia (**split point**) tối ưu cho từng đặc trưng. Có hai cách phổ biến để tìm split:

- **Pre-sorted**: duyệt qua tất cả các giá trị đặc trưng đã được sắp xếp và kiểm tra từng điểm chia có thể xảy ra. Cách này tìm được split tối ưu nhưng tốn nhiều bộ nhớ và thời gian.
- **Histogram-based**: discretize các giá trị đặc trưng liên tục thành các bin rời rạc, sau đó cộng dồn gradient và số lượng mẫu theo từng bin để xây dựng histogram, rồi chọn split tối ưu trên histogram.

Giả sử ta có m đặc trưng, N mẫu, và mỗi đặc trưng được chia thành K bin. Độ phức tạp của thuật toán histogram-based gồm hai phần:

- **Chi phí xây dựng histogram**: $O(N \times m)$
- **Chi phí tìm điểm chia (split)**: $O(K \times m)$

Vì $K \ll N$, phần xây histogram chiếm ưu thế. Do đó, nếu giảm được số mẫu hoặc số đặc trưng hiệu dụng, thời gian huấn luyện sẽ giảm đáng kể. Đây là cơ sở để LightGBM cải tiến hiệu suất sau này (với GOSS và EFB).

Algorithm 1: Histogram-based Algorithm

```
Input: I: training data, d: max depth
Input: m: feature dimension
nodeSet ← {0} ▷ tree nodes in current level
rowSet ← {{0, 1, 2, ...}} ▷ data indices in tree nodes
for i = 1 to d do
    for node in nodeSet do
        usedRows ← rowSet[node]
        for k = 1 to m do
            H ← new Histogram()
            ▷ Build histogram
            for j in usedRows do
                bin ← I.f[k][j].bin
                H[bin].y ← H[bin].y + I.y[j]
                H[bin].n ← H[bin].n + 1
            Find the best split on histogram H.
            ...
    Update rowSet and nodeSet according to the best
    split points.
    ...
```

Hình 3.9: *Thuật toán Histogram-based Algorithm trong GBDT*

b) Gain dựa trên phương sai (Variance Gain)

Khi xây dựng cây quyết định, mục tiêu là chọn đặc trưng và điểm chia (j,d) sao cho giảm được phương sai của giá trị gradient trong từng nút là lớn nhất. LightGBM định nghĩa *variance gain* cho đặc trưng j tại điểm chia d như sau:

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{\left(\sum_{x_i\in O:x_{ij}\le d} g_i\right)^2}{n_{jl|O}(d)} + \frac{\left(\sum_{x_i\in O:x_{ij}>d} g_i\right)^2}{n_{jr|O}(d)}\right), \tag{3.10}$$

trong đó:

- O: tập các mẫu thuộc nút đang xét, có kích thước n_O.
- g_i: gradient của mẫu i.
- $n_{jl|O}(d)$ và $n_{jr|O}(d)$: số mẫu rơi vào nhánh trái và phải sau khi cắt tại d.

Thuật toán chọn điểm chia tối ưu cho đặc trưng j là:

$$d_j^* = \arg\max_d V_{j|O}(d),$$

và chọn đặc trưng tối ưu:

$$j^* = \arg\max_j V_{j|O}(d_j^*).$$

Công thức (**??**) phản ánh mức độ "giảm phương sai" (variance reduction) của gradient khi nút hiện tại được chia thành hai nhánh con. Nói cách khác, nó đo lường mức độ mà việc chia nút giúp làm cho các giá trị gradient trong mỗi nhánh trở nên thuần nhất hơn.

Trong bối cảnh của GBDT, gradient g_i biểu thị hướng và độ lớn của sai lệch (*residual error*) giữa dự đoán của mô hình và giá trị thật. Nếu một tập mẫu có các giá trị gradient tương tự nhau (ví dụ, cùng dương hoặc cùng âm, và có độ lớn gần nhau), điều đó có nghĩa là các mẫu này "đang sai theo cùng một hướng". Khi đó, nếu gom chúng lại trong cùng một nút, mô hình có thể sửa sai cho chúng một cách hiệu quả bằng một giá trị dự đoán duy nhất ở lá. Ngược lại, nếu trong một nút chứa các mẫu có gradient trái dấu (một số dương, một số âm), thì việc gán cùng một giá trị đầu ra cho tất cả các mẫu này sẽ làm tăng sai số, vì mô hình không thể cùng lúc hiệu chỉnh theo hai hướng ngược nhau.

Do đó, khi xem xét việc chia một nút thành hai nhánh, thuật toán sẽ tính giá trị $V_{j|O}(d)$ — gọi là *variance gain* hay *information gain* — cho từng đặc trưng j và từng điểm chia d. Giá trị này càng lớn thì sự "khác biệt về hướng gradient" giữa hai nhánh càng rõ, và mỗi nhánh trở nên đồng nhất hơn về mặt gradient nội bộ. Việc chọn (j^*,d^*) sao cho $V_{j|O}(d)$ đạt cực đại giúp mô hình giảm tổng sai số nhanh nhất sau mỗi bước chia.

Có thể hiểu trực giác như sau:

- Nếu tổng gradient trong mỗi nhánh có giá trị tuyệt đối lớn, tức là hầu hết các mẫu trong nhánh đó đều "sai cùng hướng", thì việc chia theo điểm đó giúp mô hình dễ dàng điều chỉnh sai số cho toàn bộ nhóm.
- Nếu tổng gradient nhỏ (gần 0), điều này có nghĩa là các sai số trong nhánh triệt tiêu lẫn nhau (một số dương, một số âm), và việc chia tại điểm đó sẽ không mang lại nhiều lợi ích.

Từ góc nhìn thống kê, công thức (**??**) tương tự như tiêu chuẩn "giảm phương sai" trong hồi quy tuyến tính, hay "giảm độ hỗn tạp" (impurity reduction) trong cây quyết định phân loại (như chỉ số Gini). Điểm khác biệt là ở đây, biến mục tiêu không phải là y_i mà là gradient g_i, phản ánh lỗi của mô hình tại bước hiện tại. Như vậy, công thức này chính là nền tảng để LightGBM (và các mô hình boosting khác) xác định cách phát triển cây trong mỗi vòng huấn luyện. Các cải tiến như GOSS và EFB sau này đều tác động trực tiếp lên quá trình ước lượng $V_{j|O}(d)$ — bằng cách giảm số mẫu hoặc số đặc trưng được sử dụng — nhưng vẫn đảm bảo giá trị ước lượng không bị sai lệch đáng kể so với giá trị gốc, nhờ vào tính chất thống kê ổn định của công thức này khi kích thước dữ liệu lớn.

3.4.3 GOSS: Lấy mẫu một phía theo độ lớn gradient

a) Sơ lược về GOSS

GOSS (Gradient-based One-Side Sampling) là kỹ thuật giảm số mẫu hiệu dụng trong quá trình xây dựng cây, dựa trên quan sát rằng các mẫu có trị tuyệt đối gradient lớn thường chứa nhiều thông tin hơn cho việc tối ưu hoá hàm mất mát. Các mẫu có gradient lớn tương ứng với những điểm mà mô hình đang dự đoán sai nhiều nhất, vì vậy chúng cần được ưu tiên trong quá trình học. Ngược lại, các mẫu có gradient nhỏ chỉ góp phần tinh chỉnh mô hình, có thể được lấy mẫu ngẫu nhiên mà không làm sai lệch đáng kể kết quả huấn luyện. Nếu chỉ lấy mẫu ngẫu nhiên toàn bộ dữ liệu (như trong Stochastic Gradient Boosting), ta có nguy cơ bỏ sót các mẫu "quan trọng", dẫn tới sai số lớn hơn. GOSS giải quyết vấn đề này bằng cách giữ lại toàn bộ các mẫu có gradient lớn và chỉ lấy ngẫu nhiên một phần trong số các mẫu có gradient nhỏ, sau đó nhân trọng số bù cho nhóm mẫu bị giảm để đảm bảo cân bằng thống kê.

b) Mô tả thuật toán

Tại mỗi vòng boosting, GOSS thực hiện các bước sau:

1. **Tính gradient g_i cho tất cả các mẫu.**
   Ở vòng t ta tính $g_i^{(t)} = -\partial L(y_i, F_{t-1}(x_i))/\partial F_{t-1}(x_i)$ cho mọi i. Việc này yêu cầu dự đoán trên toàn bộ tập dữ liệu, tức GOSS *không* tránh được chi phí tính gradient (và dự đoán) trên toàn bộ dữ liệu — nhưng mục đích là để xác định mẫu nào "quan trọng" (có gradient lớn) để giữ lại.

2. **Sắp xếp các mẫu theo $|g_i|$ giảm dần.**
   Ta dùng trị tuyệt đối của gradient vì độ lớn (magnitude), bất kể dấu, phản ánh mức "chưa được học" của mẫu. Việc sắp xếp đưa các mẫu có ảnh hưởng lớn lên đầu để giữ lại toàn bộ nhóm này. Về hiệu năng, thay vì sắp toàn bộ ($O(N \log N)$), có thể dùng thuật toán chọn phần tử thứ k (nth_element) để lấy top-aN trong thời gian trung bình $O(N)$.

3. **Giữ lại tập A gồm $a \times 100\%$ mẫu có gradient lớn nhất.**
   Việc giữ toàn bộ top-a đảm bảo không bỏ các mẫu *under-trained* có ảnh hưởng lớn đến thông tin gain. Kích thước a thường nhỏ (ví dụ 0.05–0.1).

4. **Từ phần còn lại, chọn ngẫu nhiên tập B gồm $b \times 100\%$ mẫu.**
   Có hai cách thường gặp để chọn: lấy ngẫu nhiên trực tiếp bN mẫu từ toàn bộ tập, hoặclấy ngẫu nhiên $b(1-a)N$ mẫu từ phần còn lại. Mục tiêu là giữ một mẫu đại diện cho phần lớn mẫu có gradient nhỏ, nhằm duy trì thông tin về phân phối tổng thể mà không cần duyệt tất cả.

5. **Khi tính gain, nhân các mẫu trong B với hệ số bù**

   $$\text{fact} = \frac{1-a}{b}.$$

   Giải thích: gọi $S_{A^c} = \sum_{i \in A^c} g_i$ là tổng gradient trên phần dữ liệu bị xem là nhỏ. Nếu ta chỉ lấy một mẫu ngẫu nhiên B có kích thước bằng bN, thì kỳ vọng tổng gradient trên B bằng $\frac{b}{1-a} S_{A^c}$. Để ước lượng lại tổng gradient ban đầu S_{A^c} khi sử dụng B, ta cần nhân tổng trên B với hệ số $\frac{1-a}{b}$; đó chính là nguồn gốc của `fact`. Nhờ vậy, phép ước lượng gain dựa trên $A \cup B$ sẽ không bị lệch về quy mô so với việc dùng toàn bộ dữ liệu.

6. **Dùng tập $A \cup B$ (với trọng số đã điều chỉnh) để xây histogram và tìm split tối ưu.**
   Các bước xây histogram, tính gain theo công thức variance gain, và chọn split được thực hiện trên tập rút gọn $A \cup B$, trong đó các mẫu thuộc B được coi có trọng số bằng `fact` (còn mẫu trong A có trọng số 1). Mô hình yếu (weak learner) được huấn luyện trên tập này (với trọng số tương ứng) để sinh cây mới.

Algorithm 2: Gradient-based One-Side Sampling

Input: I: training data, d: iterations
Input: a: sampling ratio of large gradient data
Input: b: sampling ratio of small gradient data
Input: $loss$: loss function, L: weak learner
models ← {}, fact ← $\frac{1-a}{b}$
topN ← a × len(I) , randN ← b × len(I)
for $i = 1$ ***to*** d **do**
  preds ← models.predict(I)
  g ← $loss$(I, preds), w ← {1,1,...}
  sorted ← GetSortedIndices(abs(g))
  topSet ← sorted[1:topN]
  randSet ← RandomPick(sorted[topN:len(I)], randN)
  usedSet ← topSet + randSet
  w[randSet] × = fact ▷ Assign weight $fact$ to the small gradient data.
  newModel ← L(I[usedSet], − g[usedSet], w[usedSet])
  models.append(newModel)

Hình 3.10: *Thuật toán Gradient-based One-Side Sampling.*

Ví dụ: Giả sử $N = 10\,000$, $a = 0.1$, $b = 0.1$. Khi đó:

- $|A| = aN = 1\,000$ (giữ 1.000 mẫu có $|g|$ lớn nhất).
- $|B| = bN = 1\,000$ (lấy ngẫu nhiên 1.000 mẫu).
- $\text{fact} = \dfrac{1-a}{b} = \dfrac{0.9}{0.1} = 9.$

Khi xây histogram, mỗi mẫu trong B được nhân hệ số 9 để bù cho việc bỏ đi 8.000 mẫu còn lại trong phần gradient nhỏ. Như vậy tổng gradient ước lượng từ B (sau nhân) xấp xỉ tổng gradient ban đầu trên phần A^c.

c) Công thức toán học của GOSS

Giả sử tại một nút đang xét O có kích thước n_O, LightGBM sử dụng công thức *variance gain* được điều chỉnh theo kỹ thuật GOSS để đánh giá chất lượng điểm chia d cho đặc trưng j như sau:

$$\widetilde{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i\in A_l} g_i + \frac{1-a}{b}\sum_{x_i\in B_l} g_i\right)^2}{n_{jl}(d)} + \frac{\left(\sum_{x_i\in A_r} g_i + \frac{1-a}{b}\sum_{x_i\in B_r} g_i\right)^2}{n_{jr}(d)}\right), \tag{3.11}$$

trong đó:

- A_l và A_r: các mẫu trong tập A (các mẫu có gradient lớn nhất) rơi vào nhánh trái và phải sau khi chia tại d.
- B_l và B_r: các mẫu thuộc tập B (các mẫu được chọn ngẫu nhiên từ phần còn lại) rơi vào nhánh trái và phải.
- $n_{jl}(d)$ và $n_{jr}(d)$: số mẫu rơi vào từng nhánh tương ứng.
- a và b: tỉ lệ phần trăm mẫu được chọn theo gradient lớn và nhỏ.

Cấu trúc của công thức (**??**) thể hiện rõ hai ý tưởng quan trọng. Thứ nhất, tổng gradient trong tập A được giữ nguyên vì đây là nhóm mẫu có đóng góp lớn vào thông tin gradient, giúp mô hình tập trung vào các vùng dữ liệu mà hàm mất mát vẫn còn lớn. Thứ hai, phần gradient trong tập B (mẫu ngẫu nhiên từ phần còn lại) được nhân với hệ số $\dfrac{1-a}{b}$ để hiệu chỉnh quy mô, đảm bảo rằng tổng ước lượng gradient trên phần này xấp xỉ tổng gradient gốc trên toàn bộ phần bị bỏ qua. Hệ số này xuất phát từ lập luận sau: nếu lấy mẫu ngẫu nhiên $b(1-a)n$ phần tử từ A^c,

thì kỳ vọng của tổng gradient trên tập mẫu tỉ lệ với $b/(1-a)$ của tổng gradient thật. Do đó, nhân với $\frac{1-a}{b}$ sẽ đưa giá trị ước lượng về đúng kỳ vọng gốc.

Tuy nhiên, do biểu thức variance gain có dạng bình phương của tổng gradient, nên mặc dù phép nhân bù này giúp đảm bảo ước lượng *trung bình không lệch* (unbiased) cho tổng tuyến tính, nó không đảm bảo hoàn toàn tính không lệch cho giá trị gain. Cụ thể, khi ta tính kỳ vọng của bình phương tổng gradient, sẽ có thêm một thành phần phương sai của mẫu do phép lấy ngẫu nhiên, làm xuất hiện sai số trong ước lượng. Để định lượng sai số này, ta lấy biên trên cho sai số xấp xỉ $E(d) = |\widetilde{V}_j(d) - V_j(d)|$, với xác suất ít nhất $1-\delta$:

$$E(d) \leq C_{a,b}^2 \ln\frac{1}{\delta} \cdot \max\left\{\frac{1}{n_{jl}(d)}, \frac{1}{n_{jr}(d)}\right\} + 2DC_{a,b}\sqrt{\frac{\ln(1/\delta)}{n}},$$

trong đó:

- $C_{a,b} = \frac{1-a}{\sqrt{b}} \max_{x_i \in A^c} |g_i|$: hệ số thể hiện ảnh hưởng của các mẫu bị loại bỏ (gradient nhỏ), cho biết phạm vi lớn nhất của gradient bị bỏ qua.
- $D = \max\{\bar{g}_{jl}(d), \bar{g}_{jr}(d)\}$: đại diện cho giá trị trung bình của gradient trên hai nhánh trái/phải.
- $\bar{g}_{jl}(d) = \frac{\sum_{x_i} |g_i|}{n_{jl}(d)}$: giá trị trung bình của gradient tuyệt đối trong nhánh trái (tương tự cho nhánh phải).

Hai hằng số $C_{a,b}$ và D phản ánh phạm vi gradient bị loại bỏ và quy mô của từng nhánh sau khi chia, ảnh hưởng trực tiếp đến sai số xấp xỉ $E(d)$. Phần đầu của bất đẳng thức liên quan đến $\max\{1/n_{jl}, 1/n_{jr}\}$ cho thấy nếu một nhánh sau khi chia có rất ít mẫu, sai số xấp xỉ có thể tăng mạnh do giá trị gain có chứa phép chia cho số lượng mẫu. Phần thứ hai giảm theo bậc $O(1/\sqrt{n})$, cho thấy rằng khi kích thước dữ liệu lớn, sai số ước lượng do lấy mẫu là không đáng kể. Đây chính là cơ sở lý thuyết quan trọng lý giải tại sao GOSS đặc biệt hiệu quả với các tập dữ liệu lớn: dù giảm số lượng mẫu sử dụng trong việc xây histogram, mô hình vẫn gần như không mất mát về chất lượng chia nút.

Từ các phân tích trên, có thể rút ra một số hệ quả thực tiễn. Thứ nhất, GOSS giúp giảm đáng kể chi phí tính toán khi xây histogram (vốn có độ phức tạp $O(N \times m)$), vì chỉ cần duyệt qua các mẫu trong $A \cup B$, mà kích thước của tập này thường chỉ chiếm 10–20% dữ liệu gốc. Thứ hai, để sai số xấp xỉ thấp, cần chọn các giá trị a và b hợp lý, thông thường $a, b \in [0.05, 0.1]$. Nếu b quá nhỏ, hệ số bù $\frac{1-a}{b}$ trở nên lớn, dẫn đến tăng phương sai ước lượng và gây bất ổn số học khi tính tổng gradient. Ngược lại, nếu a quá lớn, hiệu năng giảm vì lượng mẫu được giữ lại quá nhiều. Do đó, LightGBM khuyến nghị chọn a và b tương đương nhau, hoặc b hơi nhỏ hơn một chút để đạt cân bằng giữa tốc độ và độ chính xác.

Tổng kết lại, công thức (**??**) là điểm cốt lõi thể hiện tinh thần của GOSS: ưu tiên giữ các mẫu có gradient lớn — nơi hàm mất mát còn cao — đồng thời ước lượng lại ảnh hưởng của các mẫu còn lại thông qua phép nhân bù trọng số. Cơ sở toán học của GOSS cho thấy độ sai lệch của ước lượng gain giảm nhanh theo kích thước dữ liệu, trong khi tốc độ huấn luyện lại tăng đáng kể nhờ giảm khối lượng tính toán. Điều này giúp LightGBM đạt được sự cân bằng giữa tốc độ và độ chính xác, trở thành lựa chọn tối ưu cho các bài toán học máy có quy mô lớn.

3.4.4 EFB: Gom bó đặc trưng loại trừ nhau

a) Sơ lược về EFB

EFB (Exclusive Feature Bundling) là kỹ thuật giúp giảm số lượng đặc trưng hiệu dụng trong quá trình huấn luyện bằng cách gộp các đặc trưng loại trừ nhau (*mutually exclusive features*) vào cùng một nhóm (bundle). Các đặc trưng được gọi là loại trừ nhau nếu chúng hiếm khi hoặc không bao giờ cùng nhận giá trị khác 0 trên cùng một mẫu. Ví dụ, trong dữ liệu one-hot hoặc bag-of-words, các đặc trưng biểu diễn "từ A xuất hiện" và "từ B xuất hiện" hiếm khi đồng thời xuất hiện, nên có thể gom lại mà không làm mất thông tin.

EFB tận dụng đặc tính này để giảm đáng kể chi phí xây histogram, vốn chiếm phần lớn thời gian trong quá trình xây dựng cây quyết định. Nếu số đặc trưng ban đầu là m, sau khi áp dụng EFB còn lại k bundle ($k \ll m$), từ đó độ phức tạp xây histogram giảm từ $O(N \times m)$ xuống $O(N \times k)$, trong đó N là số mẫu huấn luyện.

b) Mô tả thuật toán

Ý tưởng chính của EFB là gom các đặc trưng có xung đột nhỏ (ít khi cùng khác 0) vào cùng một bundle, mỗi đặc trưng trong bundle được dịch giá trị bằng một *offset* để tránh chồng lấn miền giá trị. Quy trình thực hiện có thể mô tả như sau:

1. **Tính độ xung đột giữa các đặc trưng.**
   Với mỗi cặp (f_i, f_j), tính số lượng mẫu mà cả hai cùng có giá trị khác 0 (gọi là *conflict count*). Cặp có số xung đột thấp được coi là loại trừ nhau.

2. **Sắp xếp và gom bó.**
   Các đặc trưng được sắp xếp theo độ thưa (số lượng giá trị khác 0) giảm dần. Duyệt lần lượt từng đặc trưng và gán nó vào bundle hiện có nếu số xung đột với tất cả đặc trưng trong bundle nhỏ hơn ngưỡng ε. Nếu không có bundle phù hợp, tạo một bundle mới.

3. **Gộp giá trị bằng offset.**
   Khi gộp một đặc trưng f_i vào bundle B_k, ta cộng thêm một giá trị dịch offset_i sao cho các miền giá trị của các đặc trưng trong cùng bundle không chồng lên nhau. Khi đó, bundle B_k được biểu diễn:

$$B_k(x) = f_i(x) + \text{offset}_i, \quad \forall f_i \in B_k.$$

Algorithm 3: Greedy Bundling

```
Input: F: features, K: max conflict count
Construct graph G
searchOrder ← G.sortByDegree()
bundles ← {}, bundlesConflict ← {}
for i in searchOrder do
    needNew ← True
    for j = 1 to len(bundles) do
        cnt ← ConflictCnt(bundles[j],F[i])
        if cnt + bundlesConflict[i] ≤ K then
            bundles[j].add(F[i]), needNew ← False
            break
    if needNew then
        Add F[i] as a new bundle to bundles
Output: bundles
```

Hình 3.11: *Thuật toán gom bó đặc trưng loại trừ nhau*

Algorithm 4: Merge Exclusive Features

```
Input: numData: number of data
Input: F: One bundle of exclusive features
binRanges ← {0}, totalBin ← 0
for f in F do
    totalBin += f.numBin
    binRanges.append(totalBin)
newBin ← new  Bin(numData)
for i = 1 to numData do
    newBin[i] ← 0
    for j = 1 to len(F) do
        if F[j].bin[i] ≠ 0 then
            newBin[i] ← F[j].bin[i] + binRanges[j]
Output: newBin, binRanges
```

Hình 3.12: *Thuật toán gom bó với offset để tránh chồng lấn*

Hai hình trên minh họa quy trình cụ thể của hai thuật toán trong EFB. Thuật toán đầu tiên mô tả quá trình gom bó các đặc trưng loại trừ nhau theo hướng tham lam. Mỗi đặc trưng được duyệt lần lượt theo thứ tự đã sắp xếp, thường là theo độ thưa giảm dần. Với mỗi đặc trưng đang xét, thuật toán kiểm tra lần lượt các bundle hiện có và tính số lượng xung đột nếu đặc trưng này được thêm vào bundle đó. Nếu tổng số xung đột không vượt quá ngưỡng cho phép, đặc trưng được gộp vào bundle tương ứng; ngược lại, một bundle mới sẽ được tạo ra. Cách tiếp cận này giúp tận dụng tối đa các bundle đã có mà vẫn đảm bảo giới hạn mức chồng lấn giữa các đặc trưng trong cùng nhóm. Nhờ việc sử dụng cấu trúc dữ liệu thưa (như danh sách vị trí khác 0 hoặc bitset), việc tính toán số xung đột giữa các đặc trưng được thực hiện nhanh chóng, giảm đáng kể chi phí so với việc quét toàn bộ ma trận dữ liệu. Độ phức tạp trung bình của thuật toán phụ thuộc vào số lượng bundle hiện có, thường nhỏ hơn nhiều so với số lượng đặc trưng gốc, nên quy trình gom bó vẫn rất hiệu quả ngay cả khi số đặc trưng ban đầu lớn.

Thuật toán thứ hai trình bày cách ghép các đặc trưng trong cùng một bundle thành một đặc trưng duy nhất bằng cơ chế *offset*. Với mỗi bundle, thuật toán đầu tiên xác định tổng số bin cần thiết bằng cách cộng dồn số bin của từng đặc trưng thành phần. Sau đó, mỗi đặc trưng f_j trong bundle được ánh xạ vào một miền giá trị riêng bằng cách cộng thêm một giá trị dịch offset_j, được xác định dựa trên vị trí tích lũy của các đặc trưng trước đó trong bundle. Trong quá trình duyệt dữ liệu, nếu một mẫu có giá trị khác 0 ở đặc trưng f_j, giá trị bin của mẫu đó sẽ được gán bằng $\text{bin}(f_j) + \text{offset}_j$. Nhờ cơ chế này, các đặc trưng trong cùng bundle sẽ chiếm các khoảng giá trị rời rạc không chồng lấn nhau, giúp bảo toàn toàn bộ thông tin gốc khi tính histogram.

Phương pháp ghép bin với offset giúp giảm mạnh số lượng histogram cần xây mà vẫn giữ được khả năng tách biệt của từng đặc trưng trong không gian giá trị hợp nhất. Trong trường hợp hiếm hoi có xung đột nhỏ (tức là nhiều đặc trưng trong cùng bundle khác 0 trên cùng một mẫu), thuật toán giả định sai số này không đáng kể so với tổng thể và thường bỏ qua vì ngưỡng xung đột đã được kiểm soát rất nhỏ trong giai đoạn gom bó. Nhờ hai bước này, EFB vừa đảm bảo được độ chính xác thống kê của các đặc trưng sau gộp, vừa giúp giảm đáng kể độ phức tạp tính toán và bộ nhớ trong quá trình huấn luyện mô hình.

c) Công thức toán học của EFB

Xét m đặc trưng ban đầu $\{f_1, f_2, \ldots, f_m\}$ và tập k bundle thu được $\{B_1, B_2, \ldots, B_k\}$ sau khi áp dụng EFB, ta có:

$$B_{p(i)}(x) = f_i(x) + \Delta_i,$$

với $p(i)$ là chỉ số bundle chứa f_i và Δ_i là offset ứng với đặc trưng đó.
Khi tính gain trong quá trình xây cây, thay vì duyệt tất cả đặc trưng f_i, LightGBM chỉ cần xét các bundle B_k:

$$V_{B_k}(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in L_k(d)} g_i\right)^2}{n_{L_k(d)}} + \frac{\left(\sum_{x_i \in R_k(d)} g_i\right)^2}{n_{R_k(d)}}\right),$$

trong đó:

- $L_k(d)$ và $R_k(d)$: các mẫu rơi vào nhánh trái và phải khi chia bundle B_k tại điểm d;
- $n_{L_k(d)}$, $n_{R_k(d)}$: số mẫu trong từng nhánh tương ứng;
- g_i: gradient của mẫu x_i.

Công thức trên cho thấy EFB chỉ thay đổi không gian đặc trưng (gộp nhiều đặc trưng thành một bundle) mà vẫn giữ nguyên nguyên tắc tính gain theo variance như GBDT gốc. Vì các đặc trưng trong cùng bundle gần như loại trừ nhau, tổng gradient trong từng bin vẫn được bảo toàn, đảm bảo sai lệch ước lượng không đáng kể.

Tổng kết lại, EFB giúp giảm đáng kể chi phí tính toán khi số lượng đặc trưng lớn mà không ảnh hưởng tới độ chính xác của mô hình. Khi kết hợp với GOSS, LightGBM đồng thời giảm số lượng mẫu và số lượng đặc trưng hiệu dụng, đạt được hiệu suất huấn luyện vượt trội.

3.4.5 Hiện thực mô hình LightGBM (tóm lược thực nghiệm)

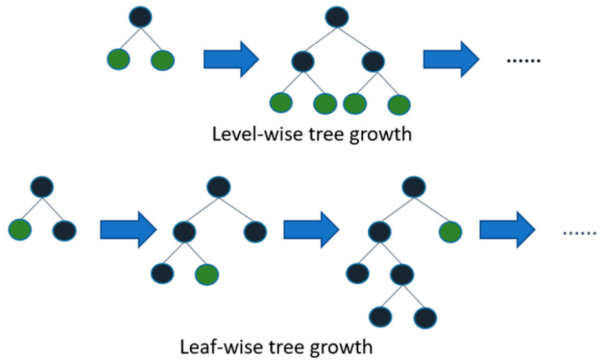
Sau khi trình bày hai kỹ thuật cốt lõi *GOSS* và *EFB*, phần này tập trung mô tả cách LightGBM được hiện thực và đánh giá trong quá trình huấn luyện thực tế. LightGBM được phát triển trên nền tảng của Gradient Boosted Decision Tree (GBDT) truyền thống, nhưng được tối ưu hóa mạnh mẽ ở nhiều khía cạnh như tốc độ huấn luyện,

quản lý bộ nhớ và khả năng mở rộng. Những cải tiến này giúp LightGBM trở thành một trong những thuật toán ensemble nhanh và hiệu quả nhất hiện nay.

Trong giai đoạn tiền xử lý, dữ liệu được discretize nhằm chuyển các giá trị liên tục thành các bin rời rạc, thông thường giới hạn ở 255 bin cho mỗi đặc trưng. Cách làm này không chỉ giảm đáng kể dung lượng lưu trữ mà còn cho phép việc xây dựng histogram diễn ra nhanh hơn nhiều lần so với việc xử lý trực tiếp trên giá trị thực. Đối với các đặc trưng thưa, LightGBM áp dụng kỹ thuật *Exclusive Feature Bundling (EFB)* để gom nhóm các đặc trưng loại trừ nhau, từ đó giảm số chiều đặc trưng mà vẫn bảo toàn thông tin cần thiết.

Trong quá trình huấn luyện, thuật toán *Gradient-based One-Side Sampling (GOSS)* được sử dụng để tối ưu hóa tốc độ xây dựng histogram. Ở mỗi vòng boosting, LightGBM giữ lại toàn bộ các mẫu có gradient lớn nhất – tương ứng với những điểm mà mô hình đang dự đoán sai nhiều – đồng thời chọn ngẫu nhiên một phần nhỏ các mẫu còn lại. Nhờ nhân trọng số bù thích hợp cho nhóm mẫu này, ước lượng thông tin gradient vẫn được đảm bảo không lệch đáng kể so với toàn bộ dữ liệu gốc. Cách tiếp cận này giúp giảm đáng kể chi phí tính toán, khi chỉ cần xử lý khoảng 10–20% dữ liệu mà vẫn duy trì được chất lượng mô hình gần như không thay đổi.

Một điểm khác biệt nổi bật khiến LightGBM đạt hiệu quả cao là chiến lược phát triển cây theo hướng *leaf-wise* thay vì *level-wise* như trong XGBoost. Trong phương pháp truyền thống, cây được mở rộng đồng loạt trên toàn bộ các nút cùng cấp, điều này giúp cấu trúc cây cân đối nhưng lại lãng phí tài nguyên khi phải chia cả những nhánh có mức độ sai số nhỏ. Ngược lại, LightGBM chọn phát triển cây theo hướng lá, tức là ở mỗi bước, thuật toán chỉ chia tiếp lá có giá trị *information gain* lớn nhất. Việc tập trung vào những vùng dữ liệu phức tạp nhất giúp mô hình giảm nhanh giá trị hàm mất mát và đạt độ hội tụ nhanh hơn.



Hình 3.13: *So sánh chiến lược phát triển cây: Level-wise và Leaf-wise.*

Nếu ký hiệu ΔL_l là mức giảm hàm mất mát khi chia lá l, thì tại mỗi bước, LightGBM sẽ chọn lá tối ưu theo công thức:

$$l^* = \arg\max_l \Delta L_l.$$

Sau khi chia, hai lá mới được thêm vào cây, và quá trình tiếp tục cho đến khi đạt số lá tối đa (`num_leaves`) hoặc giới hạn độ sâu (`max_depth`). Phương pháp leaf-wise giúp mô hình giảm lỗi nhanh hơn và đạt được độ chính xác cao chỉ với ít cây hơn. Tuy nhiên, vì có thể tạo ra các cây rất sâu ở một số nhánh, mô hình dễ bị overfitting nếu không kiểm soát. Do đó, LightGBM giới hạn độ sâu và kích thước tối thiểu của mỗi lá (`min_data_in_leaf`) để duy trì khả năng tổng quát hoá.

Bên cạnh đó, LightGBM tận dụng cơ chế huấn luyện song song và phân tán để tăng tốc xử lý. Quá trình xây histogram được chia đều cho nhiều luồng tính toán, đồng thời sử dụng phương pháp *histogram subtraction* — tức histogram của nhánh phải có thể được tính nhanh bằng cách lấy hiệu giữa histogram của nút cha và nhánh trái. Cách làm này giảm đáng kể số phép tính cần thiết, giúp mô hình đạt hiệu năng gần tuyến tính theo số lượng lõi xử lý.

Sau khi huấn luyện, mô hình được đánh giá bằng các chỉ số như Accuracy, F1-score, hoặc AUC, tùy thuộc vào bài toán phân loại hay hồi quy. Các tham số quan trọng cần tinh chỉnh bao gồm `num_leaves`, `max_depth`, `learning_rate`, `feature_fraction`, và `bagging_fraction`. Thực nghiệm cho thấy LightGBM thường đạt hội tụ nhanh hơn và sử dụng ít bộ nhớ hơn so với XGBoost, đặc biệt trên những tập dữ liệu có kích thước lớn hoặc nhiều chiều đặc trưng.

3.4.6 Các tham số chung của mô hình LightGBM

Mô hình LightGBM được thiết lập với các thông số dưới đây:

Tham số	Kiểu dữ liệu / Default	Ý nghĩa
num_leaves	`int`, mặc định = 31	Số lượng lá tối đa của mỗi cây. Tham số này trực tiếp kiểm soát độ phức tạp của mô hình (đặc biệt trong chiến lược **Leaf-wise**). Giá trị càng lớn giúp mô hình học chi tiết hơn, nhưng cũng dễ dẫn tới overfitting.
max_depth	`int`, mặc định = -1	Độ sâu tối đa của cây. Nếu đặt là -1, cây sẽ phát triển tự do theo Leaf-wise cho đến khi đạt điều kiện dừng (ví dụ num_leaves). Tham số này giúp giới hạn chiều sâu và ngăn overfitting.
learning_rate	`float`, mặc định = 0.1	Tốc độ học, xác định mức độ đóng góp của mỗi cây vào mô hình cuối cùng. Giá trị nhỏ giúp học ổn định hơn nhưng cần nhiều vòng boosting hơn. Thường chọn trong khoảng 0.01–0.3.
n_estimators	`int`, mặc định = 100	Số lượng cây (vòng boosting). Kết hợp với learning_rate để điều chỉnh độ mạnh của mô hình.
feature_fraction	`float`, mặc định = 1.0	Tỷ lệ đặc trưng được chọn ngẫu nhiên khi xây mỗi cây. Giúp giảm overfitting và tăng tính đa dạng. Kỹ thuật này kết hợp hiệu quả với **EFB** để giảm số chiều đặc trưng trước khi chọn.
bagging_fraction	`float`, mặc định = 1.0	Tỷ lệ mẫu được chọn ngẫu nhiên để huấn luyện mỗi cây. Khi nhỏ hơn 1.0, LightGBM thực hiện sampling theo mini-batch, giúp giảm thời gian và tăng tính tổng quát hóa. Kết hợp hiệu quả với **GOSS** để chọn mẫu quan trọng theo độ lớn gradient.
bagging_freq	`int`, mặc định = 0	Tần suất thực hiện bagging. Nếu = 0 thì không bagging, nếu = 5 thì cứ mỗi 5 vòng boosting sẽ lấy mẫu lại.
lambda_l1	`float`, mặc định = 0.0	Hệ số regularization L1 (Lasso) trên trọng số lá, giúp làm thưa mô hình (sparse) và giảm overfitting.
lambda_l2	`float`, mặc định = 0.0	Hệ số regularization L2 (Ridge) trên trọng số lá, giúp ổn định mô hình và giảm dao động trong gradient.
min_data_in_leaf	`int`, mặc định = 20	Số lượng mẫu tối thiểu trong một lá. Tham số này giúp giới hạn việc chia lá quá nhỏ, tránh việc mô hình học nhiễu. Giá trị lớn làm mô hình đơn giản hơn nhưng có thể giảm độ chính xác.
min_gain_to_split	`float`, mặc định = 0.0	Ngưỡng gain tối thiểu cần đạt được để thực hiện chia nút. Nếu gain nhỏ hơn giá trị này, nút sẽ không được chia tiếp. Tham số này đặc biệt quan trọng trong chiến lược **Leaf-wise** nhằm kiểm soát việc chia quá sâu.
boosting_type	`str`, mặc định = "gbdt"	Loại boosting được sử dụng: `gbdt`, `dart`, hoặc `goss`.
objective	`str`, mặc định = "regression"	Hàm mất mát chính: `regression`, `binary`, `multiclass`, `lambdarank`, v.v. Xác định loại bài toán cần tối ưu (hồi quy, phân loại, xếp hạng...).
metric	`str/list`, mặc định = "auto"	Chỉ định hàm đánh giá: RMSE, Logloss, AUC, Accuracy, NDCG... Có thể dùng nhiều metric cùng lúc để theo dõi hiệu năng.

Tham số	Kiểu dữ liệu / Default	Ý nghĩa
early_stopping_round	`int`, mặc định = None	Dừng sớm nếu metric trên tập validation không được cải thiện sau n vòng. Giúp tiết kiệm thời gian và tránh overfitting.
max_bin	`int`, mặc định = 255	Số lượng **bin** khi thực hiện phân loại giá trị đặc trưng liên tục. LightGBM sử dụng kỹ thuật histogram-based binning để tăng tốc quá trình tìm split, giảm độ phức tạp từ $O(N \times m)$ xuống $O(\text{num\_bins} \times m)$.
verbosity	`int`, mặc định = 1	Mức độ hiển thị log trong quá trình huấn luyện. 0 để tắt, >1 để hiển thị chi tiết.
num_threads	`int`, mặc định = -1	Số lượng luồng CPU được sử dụng. -1 nghĩa là dùng tất cả các luồng khả dụng.
device_type	`str`, mặc định = "cpu"	Chọn thiết bị huấn luyện: cpu hoặc gpu. Khi dùng GPU, LightGBM sử dụng song song hóa theo histogram để tăng tốc đáng kể.
seed	`int`, mặc định = None	Giá trị khởi tạo random seed để đảm bảo kết quả tái lập được.

3.5 Các phương pháp đánh giá mô hình

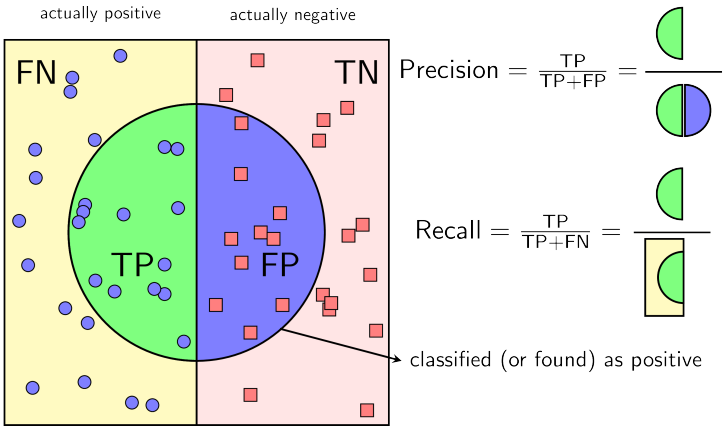
3.5.1 Ma trận nhầm lẫn (Confusion Matrix)

Ma trận nhầm lẫn là một bảng đơn giản được sử dụng để đo lường hiệu suất của mô hình phân loại. Nó so sánh các dự đoán do mô hình đưa ra với kết quả thực tế và cho thấy mô hình đã đúng hoặc sai ở đâu. Điều này giúp hiểu được nơi mô hình đang mắc lỗi để có thể cải thiện nó.

a) Các thành phần của Confusion Matrix [?]

Ma trận nhầm lẫn phân tích các dự đoán thành bốn danh mục chính:

- **Dương tính Thực (True Positive - TP)**: Mô hình dự đoán đúng kết quả dương tính, tức kết quả thực tế là dương tính.
- **Âm tính Thực (True Negative - TN)**: Mô hình dự đoán đúng kết quả âm tính, tức kết quả thực tế là âm tính.
- **Dương tính Giả (False Positive - FP)**: Mô hình dự đoán sai kết quả dương tính, tức kết quả thực tế là âm tính. Còn được gọi là **Lỗi Loại I (Type I Error)**.
- **Âm tính Giả (False Negative - FN)**: Mô hình dự đoán sai kết quả âm tính, tức kết quả thực tế là dương tính. Còn được gọi là **Lỗi Loại II (Type II Error)**.



Hình 3.14: *Các chỉ số đánh giá*

b) Biểu diễn Confusion Matrix

Thực tế (Actual)	Dự đoán (Predicted)		Tổng
	Positive	Negative	
Positive	TP (True Positive)	FN (False Negative)	TP + FN
Negative	FP (False Positive)	TN (True Negative)	FP + TN
Tổng	TP + FP	FN + TN	TP + TN + FP + FN

Bảng 3.6: *Confusion Matrix cho bài toán phân loại nhị phân*

c) Các chỉ số đánh giá dựa trên Confusion Matrix

Accuracy (Độ chính xác) Độ chính xác cho thấy có bao nhiêu dự đoán mà mô hình đã đúng trong tổng số các dự đoán. Nó cho ý tưởng về hiệu suất tổng thể nhưng có thể gây hiểu nhầm khi một lớp chiếm ưu thế hơn lớp khác.

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

Precision (Độ chính xác dương) Precision tập trung vào chất lượng của các dự đoán dương tính của mô hình. Nó cho chúng ta biết có bao nhiêu dự đoán "dương tính" thực sự chính xác.

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

Recall (Độ nhạy) Recall đo lường mô hình tốt như thế nào trong việc dự đoán các trường hợp dương tính. Nó cho thấy tỷ lệ các trường hợp dương tính thực tế được phát hiện.

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

F1-score F1-score kết hợp Precision và Recall thành một chỉ số duy nhất để cân bằng sự đánh đổi giữa chúng. Nó cung cấp ý nghĩa tốt hơn về hiệu suất tổng thể của mô hình, đặc biệt đối với các tập dữ liệu mất cân bằng.

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

Specificity (Độ đặc hiệu) Specificity đo lường khả năng của mô hình trong việc xác định chính xác các trường hợp âm tính. Specificity còn được gọi là Tỷ lệ Âm tính Thực (True Negative Rate).

$$Specificity = \frac{TN}{TN+FP}$$

Lỗi Loại I và Loại II (Type 1 and Type 2 Error)

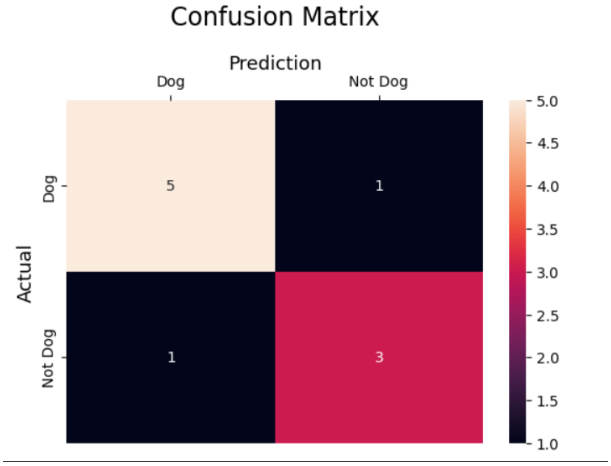
- **Lỗi Loại I (Type 1 Error)**: Xảy ra khi mô hình dự đoán sai một trường hợp dương tính nhưng trường hợp thực tế là âm tính. Đây còn được gọi là **dương tính giả (false positive)**. Lỗi Loại I ảnh hưởng đến **precision** của mô hình.
$$\text{Type 1 Error} = \frac{FP}{FP+TN}$$
- **Lỗi Loại II (Type 2 Error)**: Xảy ra khi mô hình không dự đoán được một trường hợp dương tính mặc dù nó thực sự là dương tính. Đây còn được gọi là **âm tính giả (false negative)**. Lỗi Loại II ảnh hưởng đến **recall** của mô hình.
$$\text{Type 2 Error} = \frac{FN}{TP+FN}$$

d) Ví dụ minh họa

Xét bài toán phân loại hình ảnh "Chó" và "Không phải chó" với 10 mẫu dữ liệu:

Mẫu	1	2	3	4	5	6	7	8	9	10
Thực tế	Chó	Chó	Chó	Không	Chó	Không	Chó	Chó	Không	Không
Dự đoán	Chó	Không	Chó	Không	Chó	Chó	Chó	Chó	Không	Không
Kết quả	TP	FN	TP	TN	TP	FP	TP	TP	TN	TN

Bảng 3.7: *Ví dụ chi tiết các dự đoán*



Hình 3.15: *Confusion Matrix cho bài toán phân loại*

Thực tế	**Dự đoán**		**Tổng**
	Chó	**Không phải**	
Chó	TP = 5	FN = 1	6
Không phải	FP = 1	TN = 3	4
Tổng	6	4	10

Bảng 3.8: *Confusion Matrix tổng hợp*

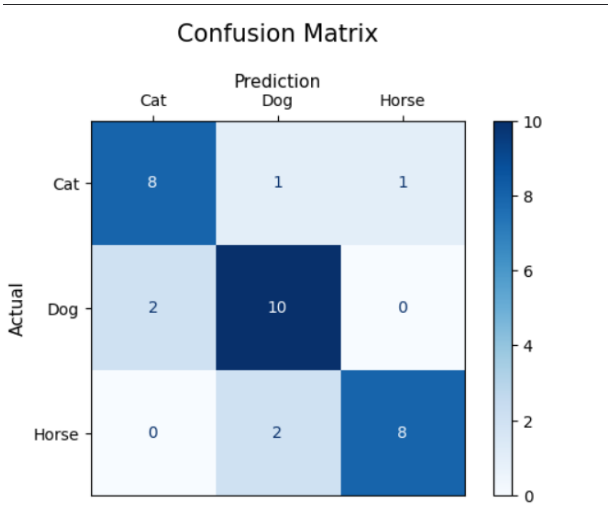
Tính các chỉ số:

- **Accuracy** = $\frac{5+3}{10} = 0.8$ (80%)
- **Precision** = $\frac{5}{5+1} = 0.833$ (83.3%)
- **Recall** = $\frac{5}{5+1} = 0.833$ (83.3%)
- **F1-score** = $\frac{2\times0.833\times0.833}{0.833+0.833} = 0.833$ (83.3%)
- **Specificity** = $\frac{3}{3+1} = 0.75$ (75%)

e) Confusion Matrix cho bài toán Đa lớp (Multi-class)

Trong phân loại đa lớp, ma trận nhầm lẫn được mở rộng để xem xét nhiều lớp:

- **Hàng** đại diện cho các lớp thực tế (ground truth)
- **Cột** đại diện cho các lớp dự đoán
- Mỗi ô trong ma trận cho thấy tần suất một lớp thực tế cụ thể được dự đoán là một lớp khác



Hình 3.16: *Confusion Matrix cho bài toán phân loại 3 lớp*

Thực tế	Dự đoán			Tổng
	Mèo	Chó	Ngựa	
Mèo	8	1	1	10
Chó	2	10	0	12
Ngựa	0	2	8	10
Tổng	10	13	9	32

Bảng 3.9: *Confusion Matrix cho bài toán phân loại 3 lớp*

3.5.1.0.1 Ví dụ: Phân loại hình ảnh Mèo, Chó, Ngựa

Trong ví dụ này:

- **Mèo**: 8 được xác định chính xác, 1 bị xác định nhầm thành chó, 1 bị xác định nhầm thành ngựa
- **Chó**: 10 được xác định chính xác, 2 bị xác định nhầm thành mèo
- **Ngựa**: 8 được xác định chính xác, 2 bị xác định nhầm thành chó

3.5.2 Đường cong ROC và AUC [?]

a) Giới thiệu về đường cong ROC

Đường cong ROC (Receiver Operating Characteristic) là một công cụ trực quan được sử dụng để đánh giá hiệu suất của các mô hình phân loại nhị phân. Nó minh họa sự đánh đổi giữa hai tỷ lệ:

- **Tỷ lệ Dương tính Thực (True Positive Rate - TPR)**: Còn được gọi là **Độ nhạy (Sensitivity)** hoặc **Recall**.
- **Tỷ lệ Dương tính Giả (False Positive Rate - FPR)**: Tỷ lệ các trường hợp âm tính bị phân loại sai.

b) Công thức tính toán

Các chỉ số được sử dụng trong đường cong ROC được tính dựa trên Ma trận Nhầm lẫn (Confusion Matrix):

$$TPR = \frac{TP}{TP+FN}$$

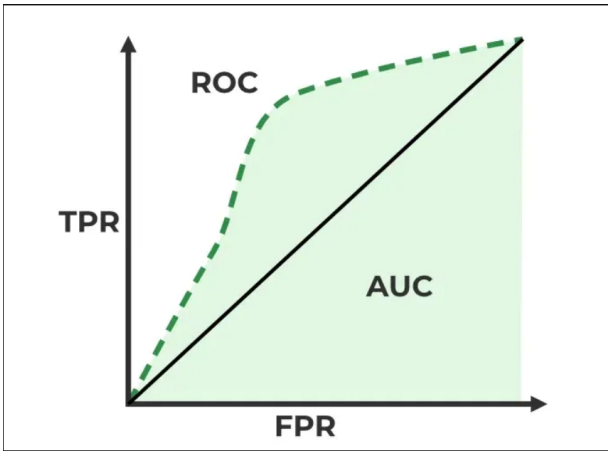
$$FPR = \frac{FP}{FP+TN}$$

$$Specificity = \frac{TN}{TN+FP} = 1 - FPR$$

c) Ý nghĩa của đường cong ROC

Đường cong ROC được tạo ra bằng cách vẽ TPR theo FPR tại **tất cả các ngưỡng phân loại có thể**. Mỗi điểm trên đường cong đại diện cho một ngưỡng quyết định khác nhau.

- Điểm ở góc trên bên trái $(0,1)$ biểu thị một bộ phân loại hoàn hảo (TPR=1, FPR=0).
- Đường chéo từ $(0,0)$ đến $(1,1)$ biểu thị hiệu suất của một bộ phân loại ngẫu nhiên, không có kỹ năng phân biệt.
- Một đường cong càng "cong" lên phía trên góc trên bên trái thì mô hình càng tốt.



Hình 3.17: *Chỉ số đánh giá phân loại ROC-AUC*

d) AUC (Area Under the Curve)

AUC là diện tích nằm dưới đường cong ROC, cung cấp một thước đo tổng hợp về khả năng phân loại của mô hình.

$$AUC = \int_0^1 TPR(FPR)d(FPR)$$

e) Diễn giải giá trị AUC

- **AUC = 1.0**: Mô hình hoàn hảo, có khả năng phân biệt tuyệt đối giữa hai lớp.
- **AUC = 0.9 - 0.99**: Mô hình xuất sắc.
- **AUC = 0.8 - 0.89**: Mô hình tốt.
- **AUC = 0.7 - 0.79**: Mô hình trung bình.
- **AUC = 0.5**: Mô hình không có khả năng phân biệt, tương đương với đoán ngẫu nhiên.
- **AUC < 0.5**: Mô hình hoạt động tệ hơn cả đoán ngẫu nhiên; có thể đảo ngược dự đoán để cải thiện.

f) Khi nào sử dụng AUC-ROC?

AUC-ROC đặc biệt hữu ích khi:

- Tập dữ liệu có sự cân bằng tương đối giữa các lớp.
- Chi phí của False Positive (FP) và False Negative (FN) được coi là tương đương.
- Khi muốn đánh giá hiệu suất mô hình trên nhiều ngưỡng phân loại khác nhau mà không cần chọn một ngưỡng cụ thể.

Lưu ý: Đối với các tập dữ liệu mất cân bằng nghiêm trọng, đường cong Precision-Recall thường cung cấp đánh giá thực tế hơn so với AUC-ROC.

Chương 4

Hiện thực bài toán

4.1 Xác định bài toán

Trong bài toán này, nhóm sử dụng mô hình học máy có giám sát (Supervised) để thực hiện huấn luyện mô hình do bộ dữ liệu của bài toán được đánh nhãn. Nhãn của bộ dữ liệu là dữ liệu dạng phân loại nhị phân (0, 1) nên trong bài toán này nhóm sẽ sử dụng các mô hình dạng phân loại (Classifier).

Trong bài toán này, nhóm sử dụng bộ dữ liệu bao gồm 121 đặc trưng và nhãn **TARGET**.

4.2 Tiền xử lý dữ liệu

Trước khi huấn luyện mô hình, thực hiện tiền xử lý dữ liệu nhằm loại bỏ các biến không có ý nghĩa, xử lý dữ liệu khuyết, xử lý outliers.

4.2.1 Loại bỏ các biến không có ý nghĩa trong bài toán

Tại bước này, tổng cộng có 43 đặc trưng không có ý nghĩa bị loại bỏ được thống kê trong bảng dưới đây:

```
    # Drop unneccessary features
    df = df.drop(columns=[
        "SK_ID_CURR", "FLAG_MOBIL", "FLAG_EMP_PHONE", "FLAG_CONT_MOBILE",
        "FLAG_WORK_PHONE", "FLAG_PHONE", "FLAG_EMAIL", "FLAG_DOCUMENT_2",
        "FLAG_DOCUMENT_3", "FLAG_DOCUMENT_4", "FLAG_DOCUMENT_5", "FLAG_DOCUMENT_6",
        "REG_REGION_NOT_LIVE_REGION", "FLAG_DOCUMENT_7", "FLAG_DOCUMENT_8",
        "FLAG_DOCUMENT_9", "FLAG_DOCUMENT_10", "FLAG_DOCUMENT_11", "FLAG_DOCUMENT_12",
        "FLAG_DOCUMENT_13", "FLAG_DOCUMENT_14", "FLAG_DOCUMENT_15", "FLAG_DOCUMENT_16",
        "FLAG_DOCUMENT_17", "FLAG_DOCUMENT_18", "FLAG_DOCUMENT_19", "FLAG_DOCUMENT_20",
        "FLAG_DOCUMENT_21", "LIVE_REGION_NOT_WORK_REGION", "REG_CITY_NOT_LIVE_CITY",
        "REG_CITY_NOT_WORK_CITY", "LIVE_CITY_NOT_WORK_CITY", "DAYS_LAST_PHONE_CHANGE",
        "AMT_REQ_CREDIT_BUREAU_HOUR", "AMT_REQ_CREDIT_BUREAU_DAY",
        "AMT_REQ_CREDIT_BUREAU_WEEK", "AMT_REQ_CREDIT_BUREAU_MON",
        "AMT_REQ_CREDIT_BUREAU_QRT", "AMT_REQ_CREDIT_BUREAU_YEAR",
        "REG_REGION_NOT_WORK_REGION", "DAYS_ID_PUBLISH", "DAYS_BIRTH",
        'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START'
    ])
```

Listing 4.1: Loại bỏ các cột không có nghĩa

Bảng 4.1: *Thông tin các biến liên quan đến thông tin liên lạc, khu vực và tài liệu khách hàng*

STT	Column	Type	Description
1	SK_ID_CURR	int64	ID of loan in our sample
2	FLAG_MOBIL	int64	Did client provide mobile phone (1=YES, 0=NO)
3	FLAG_EMP_PHONE	int64	Did client provide work phone (1=YES, 0=NO)
4	FLAG_WORK_PHONE	int64	Did client provide home phone (1=YES, 0=NO)
5	FLAG_CONT_MOBILE	int64	Was mobile phone reachable (1=YES, 0=NO)

STT	Column	Type	Description
6	FLAG_PHONE	int64	Did client provide home phone (1=YES, 0=NO)
7	FLAG_EMAIL	int64	Did client provide email (1=YES, 0=NO)
8	REG_REGION_NOT_LIVE_REGION	int64	Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)
9	REG_REGION_NOT_WORK_REGION	int64	Flag if client's permanent address does not match work address (1=different, 0=same, at region level)
10	LIVE_REGION_NOT_WORK_REGION	int64	Flag if client's contact address does not match work address (1=different, 0=same, at region level)
11	REG_CITY_NOT_LIVE_CITY	int64	Flag if client's permanent address does not match contact address (1=different, 0=same, at city level)
12	REG_CITY_NOT_WORK_CITY	int64	Flag if client's permanent address does not match work address (1=different, 0=same, at city level)
13	LIVE_CITY_NOT_WORK_CITY	int64	Flag if client's contact address does not match work address (1=different, 0=same, at city level)
14	FLAG_DOCUMENT_2	int64	Did client provide document 2
15	FLAG_DOCUMENT_3	int64	Did client provide document 3
16	FLAG_DOCUMENT_4	int64	Did client provide document 4
17	FLAG_DOCUMENT_5	int64	Did client provide document 5
18	FLAG_DOCUMENT_6	int64	Did client provide document 6
19	FLAG_DOCUMENT_7	int64	Did client provide document 7
20	FLAG_DOCUMENT_8	int64	Did client provide document 8
21	FLAG_DOCUMENT_9	int64	Did client provide document 9
22	FLAG_DOCUMENT_10	int64	Did client provide document 10
23	FLAG_DOCUMENT_11	int64	Did client provide document 11
24	FLAG_DOCUMENT_12	int64	Did client provide document 12
25	FLAG_DOCUMENT_13	int64	Did client provide document 13
26	FLAG_DOCUMENT_14	int64	Did client provide document 14
27	FLAG_DOCUMENT_15	int64	Did client provide document 15
28	FLAG_DOCUMENT_16	int64	Did client provide document 16
29	FLAG_DOCUMENT_17	int64	Did client provide document 17
30	FLAG_DOCUMENT_18	int64	Did client provide document 18
31	FLAG_DOCUMENT_19	int64	Did client provide document 19
32	FLAG_DOCUMENT_20	int64	Did client provide document 20
33	FLAG_DOCUMENT_21	int64	Did client provide document 21
34	AMT_REQ_CREDIT_BUREAU_HOUR	float64	Number of enquiries to Credit Bureau about the client one hour before application
35	AMT_REQ_CREDIT_BUREAU_DAY	float64	Number of enquiries to Credit Bureau about the client one day before application (excluding one hour before application)
36	AMT_REQ_CREDIT_BUREAU_WEEK	float64	Number of enquiries to Credit Bureau about the client one week before application (excluding one day before application)
37	AMT_REQ_CREDIT_BUREAU_MON	float64	Number of enquiries to Credit Bureau about the client one month before application (excluding one week before application)
38	AMT_REQ_CREDIT_BUREAU_QRT	float64	Number of enquiries to Credit Bureau about the client 3 months before application (excluding one month before application)
39	AMT_REQ_CREDIT_BUREAU_YEAR	float64	Number of enquiries to Credit Bureau about the client one year before application (excluding last 3 months before application)

STT	Column	Type	Description
40	DAYS_LAST_PHONE_CHANGE	int64	Number of days since last phone change
41	DAYS_ID_PUBLISH	int64	Number of days since ID was published
42	DAYS_BIRTH	int64	Number of days since client's birth
43	WEEKDAY_APPR_PROCESS_START	object	On which day of the week did the client apply for previous application
44	HOUR_APPR_PROCESS_START	int64	Approximately at what hour did the client apply for the previous application

Sau khi loại bỏ các đặc trưng không quan trọng, bộ dữ liệu còn 307,511 quan sát và 78 đặc trưng.

4.2.2 Xử lý dữ liệu khuyết

*** Bảng phân tích tỉ lệ khuyết dữ liệu:**

STT	Column	Missing (%)
1	COMMONAREA_MEDI	69.87
2	COMMONAREA_MODE	69.87
3	COMMONAREA_AVG	69.87
4	NONLIVINGAPARTMENTS_AVG	69.43
5	NONLIVINGAPARTMENTS_MODE	69.43
6	NONLIVINGAPARTMENTS_MEDI	69.43
7	FONDKAPREMONT_MODE	68.39
8	LIVINGAPARTMENTS_AVG	68.35
9	LIVINGAPARTMENTS_MODE	68.35
10	LIVINGAPARTMENTS_MEDI	68.35
11	FLOORSMIN_AVG	67.85
12	FLOORSMIN_MODE	67.85
13	FLOORSMIN_MEDI	67.85
14	YEARS_BUILD_AVG	66.50
15	YEARS_BUILD_MODE	66.50
16	YEARS_BUILD_MEDI	66.50
17	OWN_CAR_AGE	65.99
18	LANDAREA_AVG	59.38
19	LANDAREA_MODE	59.38
20	LANDAREA_MEDI	59.38
21	BASEMENTAREA_AVG	58.52
22	BASEMENTAREA_MODE	58.52
23	BASEMENTAREA_MEDI	58.52
24	EXT_SOURCE_1	56.38
25	NONLIVINGAREA_AVG	55.18
26	NONLIVINGAREA_MODE	55.18
27	NONLIVINGAREA_MEDI	55.18
28	ELEVATORS_AVG	53.30
29	ELEVATORS_MODE	53.30
30	ELEVATORS_MEDI	53.30
31	WALLSMATERIAL_MODE	50.84
32	APARTMENTS_AVG	50.75
33	APARTMENTS_MODE	50.75
34	APARTMENTS_MEDI	50.75
35	ENTRANCES_AVG	50.35
36	ENTRANCES_MODE	50.35
37	ENTRANCES_MEDI	50.35
38	LIVINGAREA_AVG	50.19
39	LIVINGAREA_MODE	50.19
40	LIVINGAREA_MEDI	50.19

STT	Column	Missing (%)
41	HOUSETYPE_MODE	50.18
42	FLOORSMAX_AVG	49.76
43	FLOORSMAX_MODE	49.76
44	FLOORSMAX_MEDI	49.76
45	YEARS_BEGINEXPLUATATION_AVG	48.78
46	YEARS_BEGINEXPLUATATION_MODE	48.78
47	YEARS_BEGINEXPLUATATION_MEDI	48.78
48	TOTALAREA_MODE	48.27
49	EMERGENCYSTATE_MODE	47.40
50	OCCUPATION_TYPE	31.35
51	EXT_SOURCE_3	19.83
52	ORGANIZATION_TYPE	18.01
53	NAME_TYPE_SUITE	0.42
54	OBS_30_CNT_SOCIAL_CIRCLE	0.33
55	DEF_30_CNT_SOCIAL_CIRCLE	0.33
56	OBS_60_CNT_SOCIAL_CIRCLE	0.33
57	DEF_60_CNT_SOCIAL_CIRCLE	0.33
58	EXT_SOURCE_2	0.21
59	AMT_GOODS_PRICE	0.09
60	TARGET	0.00
61	NAME_CONTRACT_TYPE	0.00
62	CODE_GENDER	0.00
63	FLAG_OWN_CAR	0.00
64	FLAG_OWN_REALTY	0.00
65	CNT_CHILDREN	0.00
66	AMT_INCOME_TOTAL	0.00
67	AMT_CREDIT	0.00
68	AMT_ANNUITY	0.00
69	NAME_INCOME_TYPE	0.00
70	NAME_EDUCATION_TYPE	0.00
71	NAME_FAMILY_STATUS	0.00
72	NAME_HOUSING_TYPE	0.00
73	REGION_POPULATION_RELATIVE	0.00
74	DAYS_EMPLOYED	0.00
75	DAYS_REGISTRATION	0.00
76	CNT_FAM_MEMBERS	0.00
77	REGION_RATING_CLIENT	0.00
78	REGION_RATING_CLIENT_W_CITY	0.00

Bảng 4.2: *Tỉ lệ khuyết dữ liệu*

a) Loại bỏ các đặc trưng bị khuyết quá 40%

Từ bảng thống kê tỉ lệ dữ liệu khuyết, nhóm loại bỏ thêm 49 đặc trưng bị khuyết trên 40%.

```
df = df.replace("NaN", np.nan)
for col in df.columns:
    if df[col].isna().sum() / len(df) > 0.4:
        df = df.drop(columns=[col])
df.head()
```

Listing 4.2: Loại bỏ các đặc trưng bị khuyết quá 40%

b) Xử lý các phần dữ liệu bị khuyết còn lại

Đối với biến đặc trưng dạng phân loại, các quan sát bị khuyết bị loại bỏ. Còn đối với biến đặc trưng liên tục, IterativeImputer - một thuật toán gán đa biến (MICE) được thực hiện trong scikit-learn - được sử dụng để điền các ô dữ liệu bị khuyết.

```
    cat_columns = df.select_dtypes(include=['object']).columns
    print(cat_columns)
    num_columns = df.select_dtypes(include=['int64', 'float64']).columns.drop('TARGET',
        errors='ignore')

    df = df.dropna(subset=cat_columns)

    numeric_imputer = IterativeImputer(
        estimator=DecisionTreeRegressor(random_state=42),
        missing_values=np.nan,
        n_nearest_features=10,
        max_iter=3,
        initial_strategy='median',
        skip_complete=True,
        verbose=1,
        random_state=42
    )

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numeric_imputer, num_columns),
            ('cat', OrdinalEncoder(handle_unknown='use_encoded_value', unknown_value=-1),
                cat_columns)
        ]
    )

    y = df['TARGET']
    X = df.drop(columns=['TARGET'])

    data_fit = preprocessor.fit_transform(X)

    processed_columns = list(num_columns) + list(cat_columns)
    X_processed = pd.DataFrame(data_fit, columns=processed_columns, index=df.index)

    df = pd.concat([X_processed, y], axis=1)
```

Listing 4.3: Thực hiện điền khuyết dữ liệu định

Sau khi thực hiện xử lý dữ liệu khuyết, bộ dữ liệu còn 210211 quan sát và 29 đặc trưng.

4.2.3 Đánh giá chỉ số giá trị thông tin (IV - Information Value)

Nhằm đánh giá mức độ ảnh hưởng của các đặc trưng đến nhãn, chỉ số giá trị thông tin được sử dụng. Đối với các biến phân loại chỉ số được tính toán theo từng category của biến đó. Đối với biên liên tục, các giá trị sẽ được chia thành các category trước khi thực hiện tính toán chỉ số IV. Đối với các đặc trưng có IV dưới 0.02 thì sẽ bị loại bỏ khỏi bộ dữ liệu để huấn luyện mô hình.

```
import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier

def auto_binning(df, feature, target, max_bins=5, min_bin_size=0.05):
    """Bin partitioning By DecisionTree"""
    x = df[[feature]].copy()
    y = df[target]

    if not np.issubdtype(x[feature].dtype, np.number):
        return df[feature]

    tree = DecisionTreeClassifier(
        max_leaf_nodes=max_bins,
        min_samples_leaf=min_bin_size
    ).fit(x, y)

    thresholds = sorted(tree.tree_.threshold[tree.tree_.threshold > -2])
    bins = [-np.inf] + thresholds + [np.inf]
    binned = pd.cut(df[feature], bins=bins, duplicates='drop')
    return binned

```

```python

def calc_woe_iv(df, feature, target):
    """Calculate WOE and IV for numeric feature"""
    lst = []
    total_good = (df[target] == 0).sum()
    total_bad = (df[target] == 1).sum()

    grouped = df.groupby(feature)
    for val, group in grouped:
        good = (group[target] == 0).sum()
        bad = (group[target] == 1).sum()
        dist_good = good / total_good if total_good > 0 else 0
        dist_bad = bad / total_bad if total_bad > 0 else 0

        # Tr nh log(0)
        if dist_good == 0 or dist_bad == 0:
            woe = 0
        else:
            woe = np.log(dist_good / dist_bad)

        iv = (dist_good - dist_bad) * woe
        lst.append({
            'Feature': feature,
            'Bin': val,
            'Good': good,
            'Bad': bad,
            'WOE': round(woe, 2),
            'IV': round(iv, 2)
        })
    iv_df = pd.DataFrame(lst)
    total_iv = iv_df['IV'].sum()
    return iv_df, total_iv


def woe_iv_categorical(df, feature, target):
    """Calculate WOE and IV for categorical feature"""
    lst = []
    total_good = (df[target] == 0).sum()
    total_bad = (df[target] == 1).sum()

    for val in df[feature].dropna().unique():
        good = ((df[feature] == val) & (df[target] == 0)).sum()
        bad = ((df[feature] == val) & (df[target] == 1)).sum()
        dist_good = good / total_good if total_good > 0 else 0
        dist_bad = bad / total_bad if total_bad > 0 else 0

        woe = np.log(dist_good / dist_bad) if (dist_good > 0 and dist_bad > 0) else 0
        iv = (dist_good - dist_bad) * woe

        lst.append({
            'Feature': feature,
            'Category': val,
            'Good': good,
            'Bad': bad,
            'WOE': round(woe, 2),
            'IV': round(iv, 2)
        })

    iv_df = pd.DataFrame(lst)
    total_iv = iv_df['IV'].sum()
    return iv_df, total_iv


def compute_woe_iv_dataset(df, target, cat_features=[], max_bins=5):
    results = []
    all_dfs = []

    for feature in [c for c in df.columns if c != target]:
        if feature in cat_features:
            iv_df, total_iv = woe_iv_categorical(df, feature, target)
        else:
```

```
            binned = auto_binning(df, feature, target, max_bins=max_bins)
            df_temp = df.copy()
            df_temp[feature] = binned
            iv_df, total_iv = calc_woe_iv(df_temp, feature, target)

        all_dfs.append(iv_df)
        results.append({
            'Feature': feature,
            'IV_Total': round(total_iv, 2)
        })

    summary_df = pd.DataFrame(results).sort_values('IV_Total', ascending=False).reset_
        index(drop=True)
    full_detail_df = pd.concat(all_dfs, ignore_index=True)
    return summary_df, full_detail_df
```

Listing 4.4: Phân tích giá trị thông tin

*** Bảng Giá trị thông tin IV:**

STT	Feature	IV_Total
1	EXT_SOURCE_2	0.32
2	EXT_SOURCE_3	0.26
3	AMT_GOODS_PRICE	0.09
4	OCCUPATION_TYPE	0.07
5	REGION_RATING_CLIENT_W_CITY	0.06
6	NAME_EDUCATION_TYPE	0.06
7	AMT_CREDIT	0.06
8	REGION_RATING_CLIENT	0.06
9	CODE_GENDER	0.03
10	REGION_POPULATION_RELATIVE	0.03
11	NAME_INCOME_TYPE	0.03
12	AMT_ANNUITY	0.02
13	FLAG_OWN_CAR	0.02
14	NAME_CONTRACT_TYPE	0.02
15	NAME_HOUSING_TYPE	0.01
16	NAME_FAMILY_STATUS	0.01
17	AMT_INCOME_TOTAL	0.01
18	ORGANIZATION_TYPE	0.01
19	DEF_60_CNT_SOCIAL_CIRCLE	0.01
20	DEF_30_CNT_SOCIAL_CIRCLE	0.01
21	CNT_FAM_MEMBERS	0.00
22	DAYS_REGISTRATION	0.00
23	DAYS_EMPLOYED	0.00
24	CNT_CHILDREN	0.00
25	OBS_30_CNT_SOCIAL_CIRCLE	0.00
26	OBS_60_CNT_SOCIAL_CIRCLE	0.00
27	FLAG_OWN_REALTY	0.00
28	NAME_TYPE_SUITE	0.00

Bảng 4.3: *Information Value (IV) của các biến*

Sau khi thực hiện xử lý dữ liệu khuyết, bộ dữ liệu còn 210211 quan sát và 14 đặc trưng.

4.2.4 Loại bỏ outliers với Isolation Forest

Đối với thuật toán `Isolation Forest`, các giá trị ngoại lai sẽ trả về kết quả là 1, ngược lại thuật toán sẽ trả về kết quả là -1.

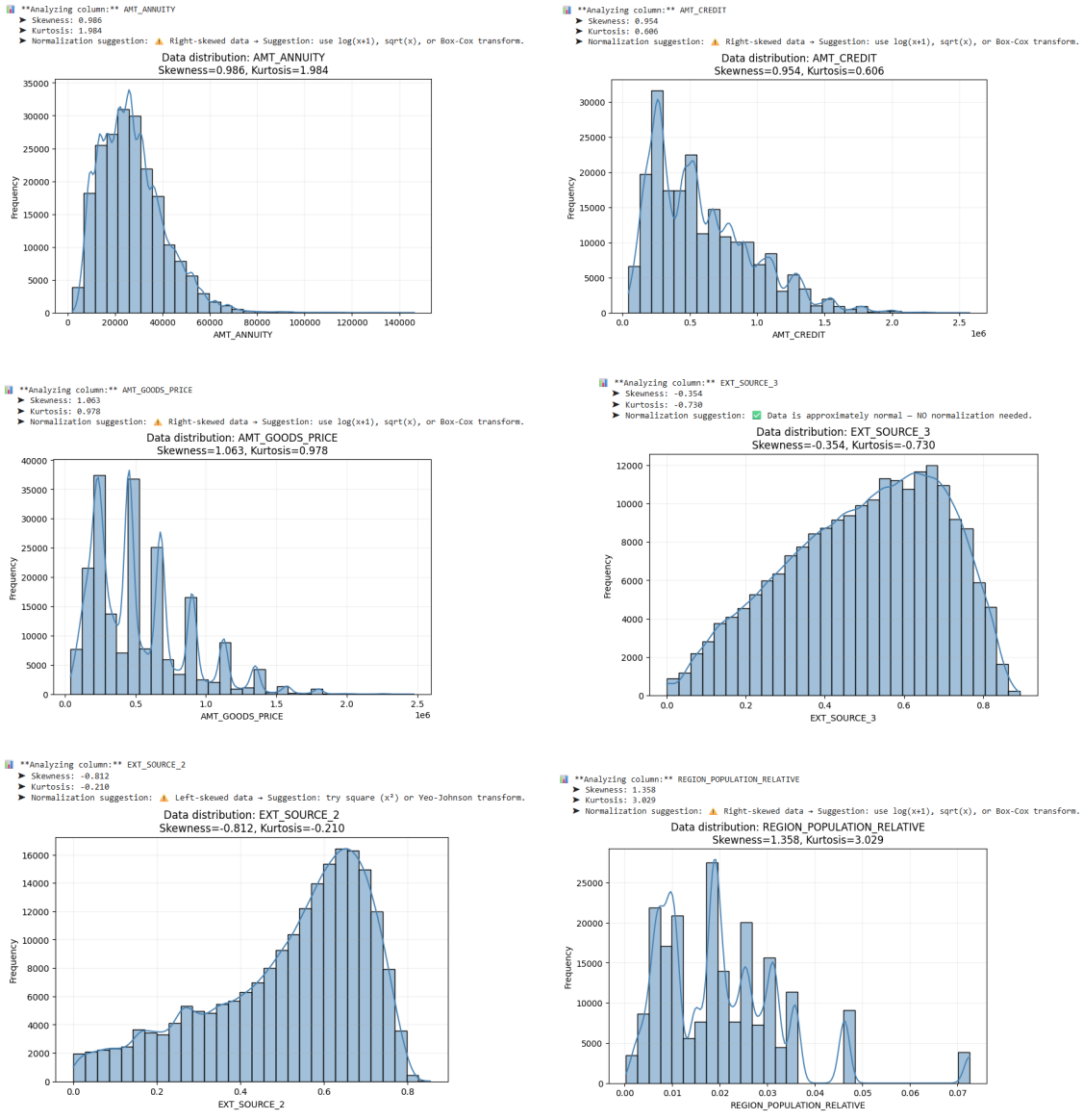
```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest

```

```python
cat_columns = categorical_features

num_columns = [x for x in df.columns if (x not in cat_columns) and x != 'TARGET']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[num_columns])

iso = IsolationForest(contamination=0.02, random_state=42, n_jobs=-1)
outlier_pred = iso.fit_predict(X_scaled)

df['is_outlier'] = outlier_pred == -1
df_clean = df[df['is_outlier'] == False]
df = df_clean.dropna()
df = df.drop(columns=['is_outlier'])
df.head()
```

Listing 4.5: Loại bỏ giá trị ngoại lai

4.2.5 Phân tích độ nhọn (Kurtosis) và độ lệch (Skewness)

*** Phân phối dữ liệu dựa trên độ nhọn và độ lệch đối với các biến liên tục:**



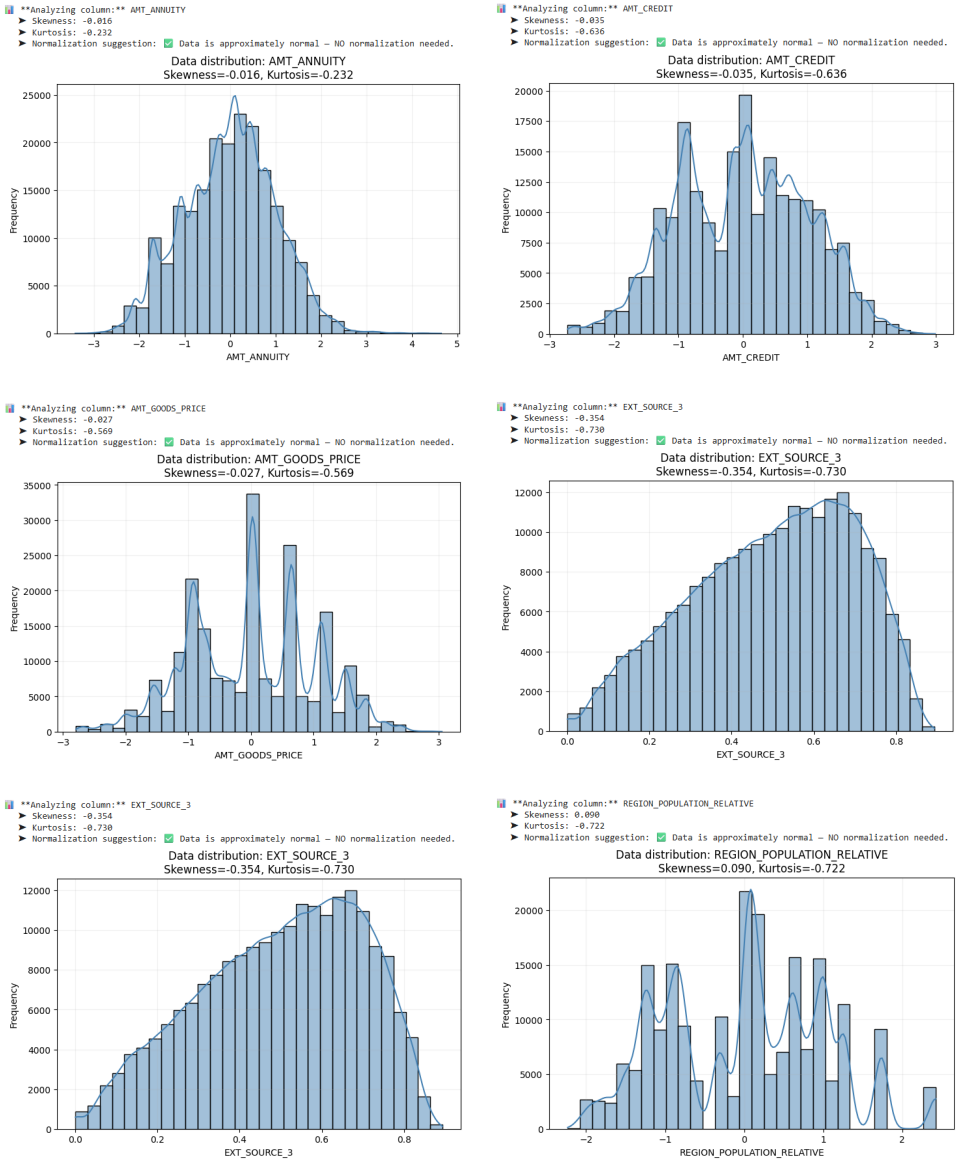
Hình 4.1: *Phân phối dữ liệu, độ nhọn và độ lệch*

Đối với những đặc trưng có phân phối dữ liệu bị lệch, trong khuôn khổ bài toán, nhóm sử dụng phép biến đổi Yeo-Johnson để đưa phân phối dữ liệu về gần phân phối chuẩn.

```
from sklearn.preprocessing import PowerTransformer

def transform_distribution(df, categorical_features):
  l_skew, r_skew, _ = analyze_distribution(df)
  for col in categorical_features:
      df[col] = df[col].apply(lambda x: round(x))

  yeo = PowerTransformer(method='yeo-johnson', standardize=True)
  box = PowerTransformer(method='box-cox', standardize=True)
  if len(l_skew) > 0:
    df[[col for col in l_skew if col not in categorical_features]] = yeo.fit_transform(df
        [[col for col in l_skew if col not in categorical_features]])
  if len(r_skew) > 0:
    df[[col for col in r_skew if col not in categorical_features]] = yeo.fit_transform(df
        [[col for col in r_skew if col not in categorical_features]])
  return df
```

Listing 4.6: Biến đổi phân phối dữ liệu với Yeo-Johnson

*** Phân phối dữ liệu các biến liên tục sau khi thực hiện biến đổi:**



Hình 4.2: *Phân phối dữ liệu, độ nhọn và độ lệch sau khi biến đổi*

4.2.6 Cân bằng dữ liệu

Trong bài toán này, nhóm thực hiện 4 phương án tái lấy mẫu nhằm cân bằng dữ liệu khác nhau:

a) **Oversampling với SMOTE**

SMOTE tạo ra mẫu tổng hợp (synthetic samples) của lớp thiểu số bằng cách nội suy tuyến tính giữa một điểm dữ liệu và các láng giềng gần của nó.

```
from imblearn.over_sampling import SMOTE
from collections import Counter

def smote_resample(X, y):
  smote = SMOTE(sampling_strategy='auto', random_state=42)
  X_smote, y_smote = smote.fit_resample(X, y)
  print(Counter(y))
  print(Counter(y_smote))
  #df_smote = pd.concat([X_smote, y_smote], axis=1)
  print(X_smote.shape)
  return X_smote, y_smote
```

Listing 4.7: Cân bằng nhãn với SMOTH

b) **Undersampling với NearMiss**

NearMiss là kỹ thuật giảm lớp đa số (majority class) một cách có kiểm soát và chọn ra các mẫu đa số nằm "gần" các mẫu thiểu số nhất — giúp giữ lại các điểm dữ liệu có giá trị phân biệt cao cho mô hình học.

```
from imblearn.under_sampling import RandomUnderSampler, NearMiss
from collections import Counter

def under_resample(X, y):
  rus = NearMiss(version=3, n_neighbors=5)
  X_rus, y_rus = rus.fit_resample(X, y)
  print(Counter(y))
  print(Counter(y_rus))
  #df_rus = pd.concat([X_rus, y_rus], axis=1)
  print(X_rus.shape)
  return X_rus, y_rus
```

Listing 4.8: Undersampling với NearMiss

c) **Tái lấy mẫu kết hợp BOTH - SMOTHENN**

SMOTHENN là phương pháp tái lấy mẫu với 2 giai đoạn:

- Cân bằng dữ liệu giữa các lớp (giống SMOTE).
- Làm sạch dữ liệu bằng cách loại bỏ các điểm nhiễu hoặc chồng lấn giữa hai lớp (nhờ ENN).

Từ đó giúp mô hình học ranh giới phân lớp rõ ràng hơn và tránh overfitting do synthetic data của SMOTE.

```
from collections import Counter
from imblearn.combine import SMOTEENN, SMOTETomek
from imblearn.over_sampling import SMOTE

def both_resample(X, y):
  smote_enn = SMOTEENN(sampling_strategy='auto', random_state=42)
  X_smote_enn, y_smote_enn = smote_enn.fit_resample(X, y)
  #df_smote_enn = pd.concat([X_smote_enn, y_smote_enn], axis=1)
  return X_smote_enn, y_smote_enn
```

Listing 4.9: Tái lấy mẫu kết hợp BOTH - SMOTHENN

d) **Tái lấy mẫu ngẫu nhiên**

Nhóm thực hiện lấy n quan sát ngẫu nhiên từ nhóm chiếm đa số với n là số lượng quan sát mẫu thiểu số.

```python
    def normal_resample(X, y):
      df = pd.concat([X, y], axis=1)
      df_0 = df[df['TARGET'] == 0]
      df_1 = df[df['TARGET'] == 1]
      n_1 = len(df_1)
      df_0 = df_0.sample(n=n_1, random_state=42)
      df = pd.concat([df_0, df_1])
      return df.drop(columns=['TARGET']), df['TARGET']
```

Listing 4.10: Tái lấy mẫu ngẫu nhiên

4.3 Mô hình CatBoost

4.3.1 Thiết lập mô hình dự đoán khả năng vỡ nợ tài chính

a) Các biến được sử dụng trong mô hình

- Biến phân loại: [
  'NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR',
  'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE',
  'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY'
  ]
- Biến liên tục: [
  'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE'
  'REGION_POPULATION_RELATIVE', 'EXT_SOURCE_2', 'EXT_SOURCE_3'
  ]
- Biến phụ thuộc: ['TARGET']

b) Thiết lập mô hình CatBoostClassifier

```python
    model = CatBoostClassifier(
        iterations=300,
        learning_rate=0.02,
        depth=6,
        loss_function='Logloss',
        eval_metric='AUC',
        verbose=10,
        early_stopping_rounds=100,
        random_seed=42
    )
```

Listing 4.11: Thiết lập mô hình CatBoostClassifier

4.3.2 Kết quả mô hình

*** Kết quả mô hình được biểu diễn lần lượt theo định dạng sau:**

(1) Normal Resampling	**(2) SMOTE Resampling**
(3) NearMiss Resampling	**(4) SMOTEENN Resampling**

Bảng 4.4: *Định dạng trình bày kết quả mô hình*

a) Báo cáo phân loại - Classification Report

	precision	recall	f1-score	support
0	0.66	0.67	0.67	3637
1	0.67	0.66	0.66	3669
accuracy			0.66	7306
macro avg	0.66	0.66	0.66	7306
weighted avg	0.66	0.66	0.66	7306

	precision	recall	f1-score	support
0	0.82	0.79	0.80	37646
1	0.80	0.82	0.81	37452
accuracy			0.81	75098
macro avg	0.81	0.81	0.81	75098
weighted avg	0.81	0.81	0.81	75098

	precision	recall	f1-score	support
0	0.69	0.82	0.75	3637
1	0.78	0.63	0.70	3669
accuracy			0.72	7306
macro avg	0.73	0.72	0.72	7306
weighted avg	0.73	0.72	0.72	7306

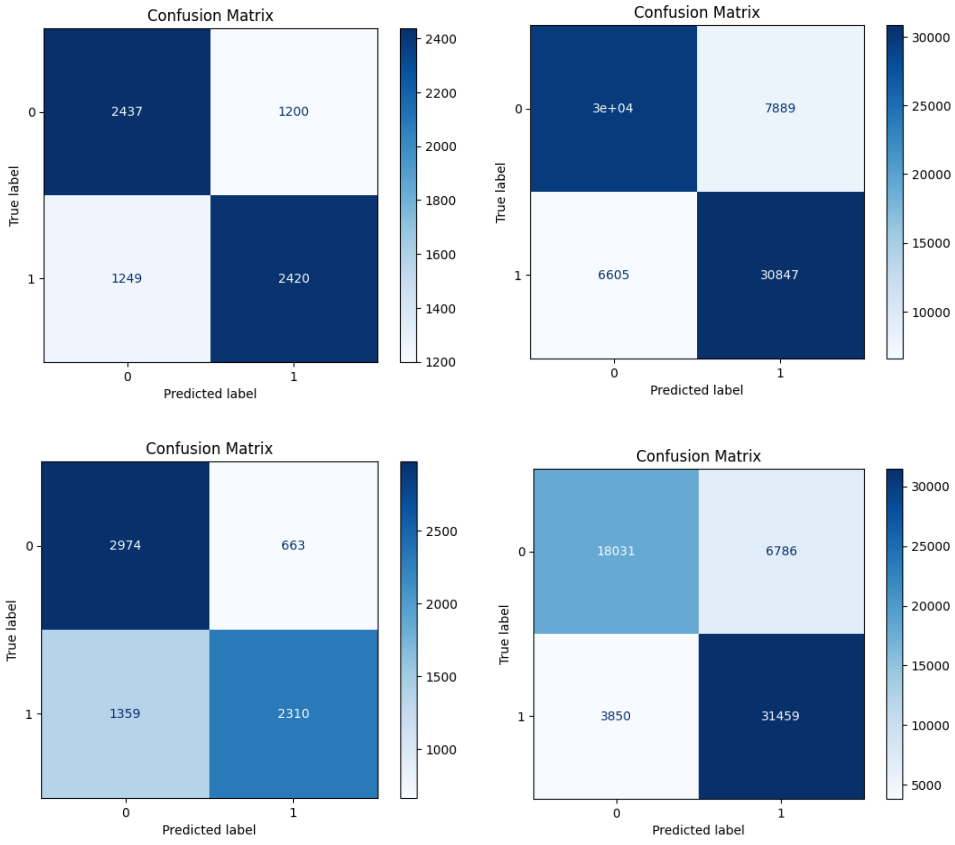
	precision	recall	f1-score	support
0	0.82	0.73	0.77	24817
1	0.82	0.89	0.86	35309
accuracy			0.82	60126
macro avg	0.82	0.81	0.81	60126
weighted avg	0.82	0.82	0.82	60126

Hình 4.3: *Classification Report của các phương án tái lấy mẫu*

*** Nhận xét:**

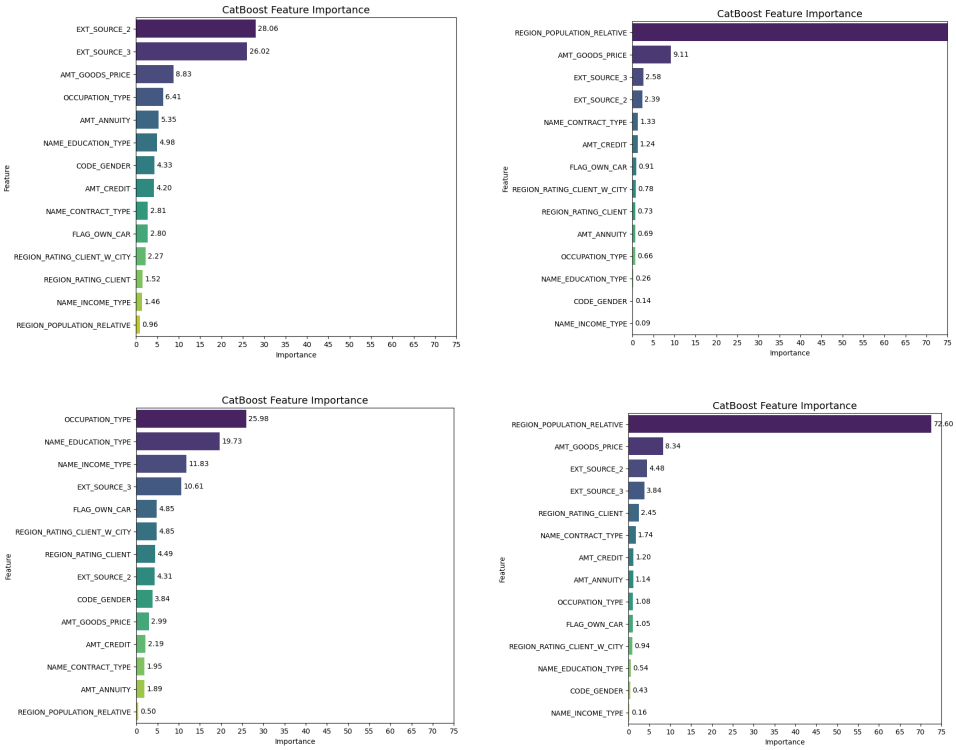
1. **Normal Resampling**
   - Độ chính xác tổng thể đạt khoảng 66% cho thấy khả năng dự đoán của mô hình khi thực hiện lấy mẫu ngẫu nhiên theo lớp tối thiểu.
   - Mô hình cho thấy sự cân bằng trên cả 2 lớp. Tuy nhiên, hai thông số Precision và Recall trên cả 2 lớp đều khá thấp cho thấy lượng sai lầm phân bố khá đều, mô hình dự đoán chưa đủ tốt.
   - **Kết luận:**

b) Ma trận nhầm lẫn - Confusion Matrix



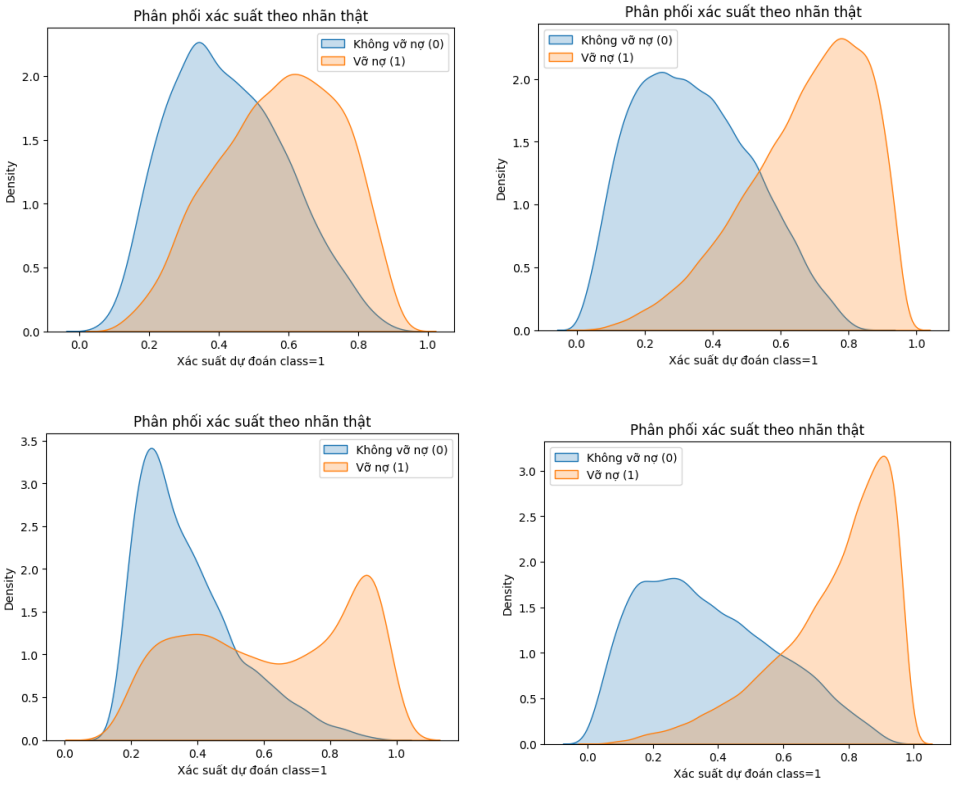
Hình 4.4: *Ma trận nhầm lẫn của các phương án tái lấy mẫu*

c) Mức độ ảnh hưởng của các đặc trưng - Feature Importances



Hình 4.5: *Mức độ ảnh hưởng của các đặc trưng của các phương án tái lấy mẫu*

Phân phối xác suất dự đoán theo nhãn thật



Hình 4.6: *Phân phối xác suất dự đoán theo nhãn thật*

4.4 Mô hình XGBoost

4.5 Mô hình LightGBM

Chương 5

Insight của bài toán