-----🙞🙜🕮🙞🙜-----

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC THĂNG LONG**

Ảnh có chứa Đồ họa, thiết kế đồ họa, ảnh chụp màn hình, Phông chữ

Mô tả được tạo tự động

**BÁO CÁO BÀI TẬP LỚN**

TIỀN XỬ LÝ DỮ LIỆU

**ĐỀ TÀI**

**BÁO CÁO XÂY DỰNG HỆ THỐNG DỰ ĐOÁN RỦI RO TÍN DỤNG**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | **Tưởng Văn Đạt** |
| **Lớp:** | **251CS32103** |
| **Nhóm thực hiện:** | **Nhóm 1** |
| **Thành viên:** | **A39592 – Phan Danh Đạt** |

|  |
| --- |
| **Hà Nội - 2025** |

**MỤC LỤC**

[CHƯƠNG 1. GIỚI THIỆU 9](#_Toc208981644)

[1.1. Bối cảnh và tầm quan trọng của bài toán 9](#_Toc208981645)

[1.2. Phát biểu vấn đề 9](#_Toc208981646)

[1.3. Mục tiêu của dự án 9](#_Toc208981647)

[1.4. Cấu trúc báo cáo 11](#_Toc208981648)

[CHƯƠNG 2. CƠ SỞ LÝ THUYẾT 12](#_Toc208981649)

[2.1. Các kỹ thuật xử lý dữ liệu mất cân bằng 12](#_Toc208981650)

[2.2. Các Thuật toán học máy được sử dụng 12](#_Toc208981651)

[2.3. Các Thước đo đánh giá mô hình 13](#_Toc208981652)

[2.4. Diễn giải mô hình với SHAP 13](#_Toc208981653)

[2.5. Vấn đề Overfitting và Underfitting trong học máy 14](#_Toc208981654)

[CHƯƠNG 3. PHƯƠNG PHÁP LUẬN VÀ THUYẾT KẾ HỆ THỐNG 15](#_Toc208981655)

[3.1. Quy trình tổng thể 15](#_Toc208981656)

[3.2. Mô tả dữ liệu 16](#_Toc208981657)

[3.3. Giai đoạn 1: Làm sạch dữ liệu với Apache Spark (Cleaning.ipynb) 16](#_Toc208981658)

[3.4. Giai đoạn 2: Kỹ thuật tạo đặc trưng (Featuring.py) 21](#_Toc208981659)

[3.5. Giai đoạn 3: Xây dựng và huấn luyện mô hình (Modeling.ipynb) 24](#_Toc208981660)

[CHƯƠNG 4. KẾT QUẢ VÀ THẢO LUẬN 33](#_Toc208981661)

[4.1. Phân tích khám phá dữ liệu (EDA) 33](#_Toc208981662)

[4.2. Hiệu suất các mô hình 34](#_Toc208981663)

[4.3. Phân tích tầm quan trọng của đặc trưng 35](#_Toc208981664)

[4.4. Phân tích lỗi (Error Analysis) 37](#_Toc208981665)

[CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 39](#_Toc208981666)

[5.1. Tổng kết kết quả 39](#_Toc208981667)

[5.2. Đóng góp của dự án 39](#_Toc208981668)

[5.3. Hạn chế của dự án 39](#_Toc208981669)

[5.4. Hướng phát triển trong tương lai 39](#_Toc208981670)

[5.5. Phụ lục và tài liệu tham khảo 39](#_Toc208981671)

**DANH MỤC HÌNH ẢNH**

[Hình 3.1‑1 Sơ đồ quy trình thực hiện dự án. 13](#_Toc208982233)

[Hình 4.1‑1 Biểu đồ thể hiện sự mất cân bằng của dữ liệu. 31](#_Toc208982234)

[Hình 4.1‑2 Biểu đồ phân phối theo điểm tín dụng. 32](#_Toc208982235)

[Hình 4.3‑1 Biểu đồ SHAP. 34](#_Toc208982236)

[Hình 4.4‑1 Ma trận nhầm lẫn của các mô hình. 35](#_Toc208982237)

[Hình 5.1‑1 Biểu đồ phân tán tỉ lệ vỡ nợ. 37](#_Toc208982238)

**DANH MỤC BẢNG BIỂU**

[Bảng 3.3‑1 Thư viên cần thiết giai đoạn 1. 14](#_Toc208982035)

[Bảng 3.3‑2 Đọc và phân tích khám phá dữ liệu. 15](#_Toc208982036)

[Bảng 3.3‑3 Xử lý các giá trị ngoại lai. 15](#_Toc208982037)

[Bảng 3.3‑4 Xử lý giá trị thiếu. 17](#_Toc208982038)

[Bảng 3.3‑5 Xử lý khoảng trắng. 18](#_Toc208982039)

[Bảng 3.3‑6 Chuẩn hóa dữ liệu. 18](#_Toc208982040)

[Bảng 3.3‑7 Lưu dữ liệu đã làm sạch. 19](#_Toc208982041)

[Bảng 3.4‑1 Thư viện cần thiết giai đoạn 2. 19](#_Toc208982042)

[Bảng 3.4‑2 Tải dữ liệu cho giai đoạn 2. 19](#_Toc208982043)

[Bảng 3.4‑3 Feature từ bureau, bureau\_balance. 20](#_Toc208982044)

[Bảng 3.4‑4 Feature từ previous\_app. 21](#_Toc208982045)

[Bảng 3.4‑5 Feature từ installments. 22](#_Toc208982046)

[Bảng 3.4‑6 Lưu dữ liệu đã Feature Engineering. 22](#_Toc208982047)

[Bảng 3.5‑1 Thư viện cần thiết tải dữ liệu cho giai đoạn 3. 23](#_Toc208982048)

[Bảng 3.5‑2 Chuẩn bị dữ liệu cho Modeling. 24](#_Toc208982049)

[Bảng 3.5‑3 Huấn luyện mô hình Logistic Regression (Baseline). 24](#_Toc208982050)

[Bảng 3.5‑4 Mô hình LightGBM tìm Threshold với Cross\_Validation. 27](#_Toc208982051)

[Bảng 3.5‑5 Mô hình Stacking LightGBM + Xgboost. 30](#_Toc208982052)

[Bảng 4.2‑1 So sánh hiệu suất các mô hình. 32](#_Toc208982053)

[Bảng 4.2‑2 Báo cáo phân loại chi tiết cho mô hình Stacking. 33](#_Toc208982054)

# lời mở đầu

Trong bối cảnh ngành tài chính tiêu dùng ngày càng phát triển, việc đánh giá chính xác rủi ro tín dụng là một yếu tố quyết định sự thành công của các tổ chức cho vay. Dự án này giải quyết bài toán dự đoán khả năng vỡ nợ của khách hàng cho Home Credit, một bài toán điển hình với thách thức về dữ liệu quy mô lớn và mất cân bằng nghiêm trọng.

Quy trình thực hiện dự án được chia thành ba giai đoạn chính. Giai đoạn đầu tập trung vào việc làm sạch và chuẩn bị dữ liệu thô, sử dụng sức mạnh của Apache Spark để xử lý hiệu quả các tệp dữ liệu lớn, bao gồm các tác vụ như xử lý giá trị thiếu, chuẩn hóa kiểu dữ liệu và xử lý giá trị ngoại lai. Giai đoạn hai là kỹ thuật tạo đặc trưng (feature engineering), chuyển sang sử dụng thư viện Pandas để tổng hợp thông tin từ nhiều nguồn dữ liệu phụ trợ (bureau, previous\_application, installments\_payments), qua đó làm giàu thông tin và tăng cường sức mạnh dự báo cho mô hình.

Giai đoạn cuối cùng là xây dựng và tối ưu hóa mô hình. Bắt đầu từ một mô hình cơ sở (Logistic Regression), dự án đã phát triển đến một kiến trúc Stacking tiên tiến, kết hợp sức mạnh của hai thuật toán Gradient Boosting mạnh mẽ là LightGBM và XGBoost. Các kỹ thuật xử lý dữ liệu mất cân bằng như SMOTE và cost-sensitive learning đã được áp dụng, cùng với các phương pháp giảm thiểu overfitting như Cross-Validation, regularization, và Early Stopping. Mô hình cuối cùng đạt được hiệu suất vượt trội với điểm ROC AUC là 0.776 và F1-score trên lớp thiểu số là 0.328, sau khi đã tối ưu hóa ngưỡng quyết định để phù hợp với mục tiêu kinh doanh. Kết quả diễn giải từ SHAP cũng chỉ ra các yếu tố rủi ro tín dụng quan trọng nhất, cung cấp những hiểu biết sâu sắc cho các nhà phân tích.

Báo cáo này sẽ trình bày chi tiết toàn bộ phương pháp luận, kết quả phân tích và thảo luận về những đóng góp cũng như các hướng phát triển tiềm năng của dự án.

# Giới thiệu

## Bối cảnh và tầm quan trọng của bài toán

Ngành tài chính tiêu dùng đóng một vai trò thiết yếu trong nền kinh tế hiện đại, cung cấp các sản phẩm tín dụng giúp người dân tiếp cận các sản phẩm, dịch vụ và cơ hội đầu tư. Tuy nhiên, hoạt động cho vay luôn gắn liền với rủi ro tín dụng - khả năng khách hàng không thể thực hiện nghĩa vụ trả nợ. Quản lý hiệu quả rủi ro này là yếu tố sống còn đối với các tổ chức tài chính. Một hệ thống đánh giá rủi ro kém hiệu quả có thể dẫn đến hai loại tổn thất chính: tổn thất tín dụng khi cho vay những khách hàng rủi ro cao, và tổn thất cơ hội khi từ chối những khách hàng có khả năng trả nợ tốt.

Dự án này được thực hiện dựa trên bộ dữ liệu từ cuộc thi "Home Credit Default Risk" trên nền tảng Kaggle, một bài toán mô phỏng sát với thực tế mà các ngân hàng và công ty tài chính phải đối mặt hàng ngày. Việc xây dựng một mô hình dự đoán chính xác không chỉ giúp tối ưu hóa lợi nhuận mà còn góp phần vào sự ổn định của hệ thống tài chính.

## Phát biểu vấn đề

Bài toán chính cần giải quyết là xây dựng một mô hình phân loại nhị phân để dự đoán xác suất một người nộp đơn vay vốn sẽ vỡ nợ. Biến mục tiêu trong bộ dữ liệu là TARGET, với giá trị 1 thể hiện khách hàng có khó khăn trong thanh toán (vỡ nợ) và 0 thể hiện ngược lại.

Thách thức cốt lõi của bài toán này đến từ các đặc điểm của dữ liệu:

* Quy mô lớn: Dữ liệu bao gồm hàng trăm nghìn hồ sơ vay vốn và được liên kết với nhiều bảng dữ liệu phụ, đòi hỏi các công cụ xử lý dữ liệu lớn.
* Độ phức tạp cao: Dữ liệu chứa nhiều loại biến (số, phân loại) với số lượng đặc trưng lớn.
* Dữ liệu mất cân bằng (Imbalanced Data): Tỷ lệ khách hàng vỡ nợ (TARGET=1) chỉ chiếm khoảng 8% tổng số hồ sơ. Đây là thách thức lớn nhất, vì các mô hình học máy tiêu chuẩn có xu hướng bỏ qua lớp thiểu số này, dẫn đến một mô hình có độ chính xác (accuracy) cao nhưng không có giá trị thực tiễn trong việc phát hiện rủi ro.

## Mục tiêu của dự án

Dự án được thực hiện với các mục tiêu cụ thể sau:

* Xây dựng một quy trình xử lý dữ liệu toàn diện, từ khâu làm sạch, tiền xử lý đến tạo đặc trưng.
* Áp dụng và đánh giá hiệu quả của Apache Spark trong việc xử lý dữ liệu lớn ở giai đoạn đầu.
* Phát triển các đặc trưng mới có sức mạnh dự báo cao bằng cách kết hợp thông tin từ nhiều nguồn dữ liệu.
* Xây dựng, huấn luyện và so sánh hiệu suất của nhiều mô hình, từ mô hình cơ sở đến các mô hình ensemble nâng cao.
* Giải quyết hiệu quả vấn đề mất cân bằng dữ liệu thông qua các kỹ thuật chuyên biệt.
* Tối ưu hóa và lựa chọn mô hình cuối cùng dựa trên các thước đo đánh giá phù hợp với bài toán kinh doanh.
* Diễn giải kết quả của mô hình để cung cấp những hiểu biết có giá trị về các yếu tố rủi ro.

## Cấu trúc báo cáo

Báo cáo này được cấu trúc thành năm chương. Chương 2 trình bày các cơ sở lý thuyết về khoa học dữ liệu và học máy liên quan đến dự án. Chương 3 đi sâu vào phương pháp luận và thiết kế hệ thống, mô tả chi tiết các bước đã được thực hiện. Chương 4 là nơi trình bày, phân tích và thảo luận về các kết quả đạt được. Cuối cùng, Chương 5 tổng kết lại toàn bộ dự án và đề xuất các hướng phát triển trong tương lai.

# Cơ Sở Lý Thuyết

## Các kỹ thuật xử lý dữ liệu mất cân bằng

Dữ liệu mất cân bằng là một vấn đề phổ biến trong nhiều lĩnh vực như y tế, tài chính, phát hiện gian lận. Để giải quyết vấn đề này, dự án đã xem xét và áp dụng các phương pháp sau:

* SMOTE (Synthetic Minority Over-sampling Technique): Đây là một kỹ thuật over-sampling thông minh. Thay vì chỉ nhân bản các mẫu thuộc lớp thiểu số, SMOTE tạo ra các mẫu dữ liệu "tổng hợp" mới. Thuật toán hoạt động bằng cách chọn một mẫu thuộc lớp thiểu số, tìm k hàng xóm gần nhất của nó, sau đó tạo ra một mẫu mới nằm trên đường thẳng nối mẫu ban đầu và một trong các hàng xóm được chọn ngẫu nhiên. Điều này giúp mở rộng vùng quyết định của lớp thiểu số và giúp mô hình tổng quát hóa tốt hơn.
* Cost-Sensitive Learning (scale\_pos\_weight): Kỹ thuật này không làm thay đổi dữ liệu mà thay đổi hàm mất mát (loss function) của thuật toán. Nó gán một "trọng số" hay "chi phí" cao hơn cho việc phân loại sai các mẫu thuộc lớp thiểu số. Ví dụ, tham số scale\_pos\_weight trong LightGBM, được tính bằng số lượng mẫu lớp 0 / số lượng mẫu lớp 1, sẽ "phạt" mô hình nặng hơn khi nó dự đoán sai một mẫu lớp 1, buộc mô hình phải nỗ lực hơn để học các đặc điểm của lớp này.

## Các thuật toán học máy được sử dụng

* Logistic Regression: Là một mô hình hồi quy tuyến tính được sử dụng cho bài toán phân loại. Nó hoạt động tốt trên dữ liệu có quan hệ tuyến tính và dễ diễn giải. Trong dự án này, nó được sử dụng làm mô hình cơ sở (baseline) để có một thước đo so sánh ban đầu.
* Gradient Boosting Machines (GBM): Là một họ các thuật toán ensemble mạnh mẽ, trong đó các mô hình (thường là cây quyết định) được xây dựng một cách tuần tự. Mỗi mô hình sau sẽ học cách sửa lỗi (dựa trên gradient của hàm mất mát) của các mô hình trước đó.
* LightGBM & XGBoost: Là hai triển khai phổ biến và hiệu quả của GBM.
* XGBoost (Extreme Gradient Boosting) phát triển cây theo từng tầng (level-wise).
* LightGBM (Light Gradient Boosting Machine) sử dụng chiến lược phát triển cây theo lá (leaf-wise), giúp hội tụ nhanh hơn và thường hiệu quả hơn trên các bộ dữ liệu lớn. Nó cũng sử dụng các kỹ thuật như Gradient-based One-Side Sampling (GOSS) và Exclusive Feature Bundling (EFB) để tăng tốc độ huấn luyện.
* Ensemble Learning và Stacking: Stacking là một kỹ thuật ensemble nâng cao. Thay vì chỉ kết hợp các dự đoán một cách đơn giản (như voting hay averaging), Stacking sử dụng một mô hình mới (gọi là meta-model hoặc Tầng 1) để học cách kết hợp tốt nhất các dự đoán từ các mô hình cơ sở (base models hoặc Tầng 0). Trong dự án này, LightGBM và XGBoost đóng vai trò là base models, và Logistic Regression là meta-model.

## Các thước đo đánh giá mô hình

* Accuracy (Độ chính xác tổng thể): Tỷ lệ dự đoán đúng trên tổng số mẫu. Thước đo này rất dễ gây hiểu lầm trên dữ liệu mất cân bằng.
* Precision, Recall, F1-Score:
* Precision (Độ chính xác của dự đoán Positive): Trong số các mẫu được dự đoán là lớp 1, có bao nhiêu mẫu thực sự là lớp 1.
* Recall (Độ phủ): Trong số các mẫu thực sự là lớp 1, mô hình đã phát hiện được bao nhiêu mẫu.
* F1-Score: Là trung bình điều hòa của Precision và Recall, cung cấp một thước đo cân bằng duy nhất, đặc biệt hữu ích khi cần cân bằng giữa hai yếu tố trên.
* ROC AUC (Area Under the Receiver Operating Characteristic Curve): Đường cong ROC thể hiện sự đánh đổi giữa Tỷ lệ True Positive (Recall) và Tỷ lệ False Positive tại các ngưỡng quyết định khác nhau. AUC là diện tích dưới đường cong này, có giá trị từ 0 đến 1. AUC đo lường khả năng tổng thể của mô hình trong việc phân biệt giữa hai lớp. Một mô hình có AUC = 0.5 tương đương với dự đoán ngẫu nhiên.

## Diễn giải mô hình với SHAP

SHAP (SHapley Additive exPlanations) là một phương pháp tiên tiến để giải thích dự đoán của mô hình. Dựa trên khái niệm Giá trị Shapley từ lý thuyết trò chơi, SHAP tính toán sự đóng góp của từng đặc trưng vào việc "đẩy" dự đoán của mô hình từ giá trị cơ sở đến giá trị cuối cùng. Điều này giúp hiểu được tại sao mô hình đưa ra một quyết định cụ thể cho một hồ sơ nhất định, tăng tính minh bạch và độ tin cậy.

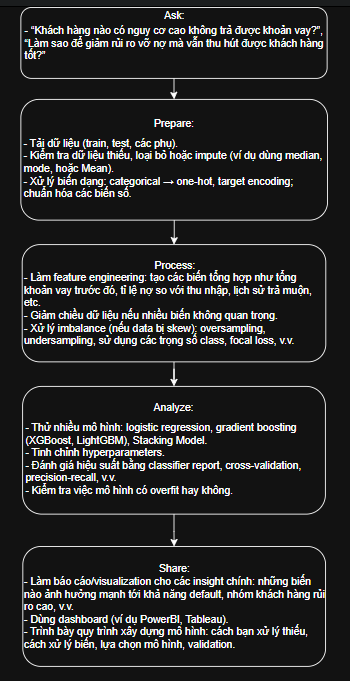
## Vấn đề Overfitting và Underfitting trong học máy

* Overfitting (Học vẹt): Xảy ra khi mô hình quá phức tạp, học thuộc cả nhiễu trong dữ liệu huấn luyện và mất khả năng tổng quát hóa trên dữ liệu mới.
* Underfitting (Học chưa tới): Xảy ra khi mô hình quá đơn giản, không nắm bắt được quy luật của dữ liệu.
* Các biện pháp đã áp dụng: Dự án đã sử dụng nhiều kỹ thuật để giảm overfitting như Cross-Validation (StratifiedKFold), Regularization (lambda\_l1, lambda\_l2), giới hạn độ phức tạp của cây (max\_depth, num\_leaves), Subsampling (feature\_fraction, bagging\_fraction) và Early Stopping.

# Phương Pháp Luận và Thiết Kế Hệ Thống

## Quy trình tổng thể

Dự án được thực hiện theo một quy trình khoa học dữ liệu chuẩn, bao gồm các bước chính được minh họa trong biểu đồ dưới đây. Quy trình này đảm bảo tính logic, chặt chẽ và khả năng tái lập.



Hình 3.1‑1 Sơ đồ quy trình thực hiện dự án.

## Mô tả dữ liệu

Bộ dữ liệu chính từ Home Credit bao gồm 8 tệp, các tệp chính được sử dụng trong dự án:

* application\_train/test.csv: Dữ liệu chính về hồ sơ vay vốn.
* bureau.csv & bureau\_balance.csv: Dữ liệu về các khoản vay trước đây của khách hàng tại các tổ chức tín dụng khác, được báo cáo bởi Credit Bureau.
* previous\_application.csv: Dữ liệu về các khoản vay trước đây của khách hàng tại chính Home Credit.
* installments\_payments.csv: Dữ liệu về lịch sử trả góp cho các khoản vay trước đây tại Home Credit.

## Giai đoạn 1: Làm sạch dữ liệu với Apache Spark (Cleaning.ipynb)

Các thư viện cần thiết trong giai đoạn 1:

|  |
| --- |
| import pandas as pd  import os  import shutil  import numpy as np  from sklearn.preprocessing import LabelEncoder  from pyspark.sql import SparkSession  from pyspark.sql.functions import col, count, when, isnan, trim, mean, expr, stddev, sum as spark\_sum  from pyspark.ml.feature import StringIndexer, OneHotEncoder, Imputer, VectorAssembler, StandardScaler  from pyspark.ml import Pipeline  from pyspark.sql import functions as F  from pyspark.sql.types import StringType, IntegerType, FloatType, DoubleType  import csv |

Bảng 3.3‑1 Thư viên cần thiết giai đoạn 1.

Lý do lựa chọn Spark: Do quy mô lớn của các tệp dữ liệu ban đầu, việc sử dụng Apache Spark cho phép thực hiện các tác vụ làm sạch và tiền xử lý một cách hiệu quả nhờ khả năng xử lý song song và phân tán.

* Các bước thực hiện:
* Đọc và khám phá dữ liệu: Sử dụng spark.read.csv để tải dữ liệu, sau đó dùng df.printSchema() và df.describe() để kiểm tra cấu trúc và các thống kê cơ bản.

|  |
| --- |
| # Tạo thư mục temp trước khi khởi tạo Spark  temp\_dir = "C:/temp/spark-temp"  warehouse\_dir = "C:/temp/spark-warehouse"  if not os.path.exists(temp\_dir):      os.makedirs(temp\_dir, exist\_ok=True)  if not os.path.exists(warehouse\_dir):      os.makedirs(warehouse\_dir, exist\_ok=True)  # Stop any existing Spark session  if 'spark' in globals():      spark.stop()  # Khởi tạo SparkSession với cấu hình tối ưu  spark = SparkSession.builder \      .appName("HomeCreditCleaning") \      .master("local[\*]") \      .config("spark.sql.adaptive.enabled", "true") \      .config("spark.sql.adaptive.coalescePartitions.enabled", "true") \      .config("spark.serializer", "org.apache.spark.serializer.KryoSerializer") \      .config("spark.local.dir", temp\_dir) \      .config("spark.sql.warehouse.dir", warehouse\_dir) \      .config("spark.driver.extraJavaOptions", f"-Djava.io.tmpdir={temp\_dir}") \      .config("spark.executor.extraJavaOptions", f"-Djava.io.tmpdir={temp\_dir}") \      .config("spark.sql.shuffle.partitions", "200") \      .config("spark.driver.memory", "4g") \      .config("spark.executor.memory", "4g") \      .config("spark.driver.maxResultSize", "2g") \      .getOrCreate()  # Set Spark log level to reduce verbosity  spark.sparkContext.setLogLevel("WARN")  # Đọc dữ liệu, tự động suy luận schema và sử dụng dòng đầu tiên làm header  file\_path = r"C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_train.csv"  df = spark.read.csv(file\_path, header=True, inferSchema=True)  df.show(5) |

Bảng 3.3‑2 Đọc và phân tích khám phá dữ liệu.

* Xử lý giá trị ngoại lai: Phát hiện và xử lý giá trị bất thường 365243 trong cột DAYS\_EMPLOYED, thay thế bằng giá trị null để xử lý ở bước sau, một số cột khác: AMT\_INCOME\_TOTAL, AMT\_CREDIT, AMT\_ANNUITY, DAYS\_BIRTH thì xử lý theo phân vị.

|  |
| --- |
| # Xử lý giá trị 365243 trong cột DAYS\_EMPLOYED (đại diện cho người không có việc)  df = df.withColumn('DAYS\_EMPLOYED', when(col('DAYS\_EMPLOYED') == 365243, None).otherwise(col('DAYS\_EMPLOYED')))  print("\nĐã xử lý giá trị bất thường trong cột 'DAYS\_EMPLOYED'.")  # Các cột số cần xử lý ngoại lai  numeric\_cols\_for\_outliers = ['AMT\_INCOME\_TOTAL', 'AMT\_CREDIT', 'AMT\_ANNUITY', 'DAYS\_BIRTH']  print("\nXử lý giá trị ngoại lai...")  for c in numeric\_cols\_for\_outliers:      # Tính phân vị 1% và 99%      # 0.001 là sai số tương đối cho phép để tăng tốc độ tính toán      quantiles = df.approxQuantile(c, [0.01, 0.99], 0.001)      q1 = quantiles[0]      q99 = quantiles[1]      if q1 is not None and q99 is not None:          print(f"Cột '{c}': Giới hạn dưới = {q1}, Giới hạn trên = {q99}")          # Áp dụng clipping          df = df.withColumn(c,              when(col(c) < q1, q1)              .when(col(c) > q99, q99)              .otherwise(col(c))          )  print("Đã xử lý xong giá trị ngoại lai.") |

Bảng 3.3‑3 Xử lý các giá trị ngoại lai.

* Xử lý giá trị thiếu:
* Tính toán tỷ lệ giá trị thiếu cho từng cột.
* Xóa các cột có tỷ lệ thiếu lớn hơn 40% để giảm nhiễu và độ phức tạp.
* Sử dụng Imputer của Spark để điền các giá trị thiếu còn lại: median cho các cột số và mode cho các cột phân loại.

|  |
| --- |
| # Kiểm tra giá trị null và NaN cho mỗi cột  missing\_counts = df.select([      count(when(isnan(c) | col(c).isNull(), c)).alias(c)      for c in df.columns  ])  print("Số lượng giá trị thiếu trong mỗi cột:")  missing\_counts.show()  # Xóa các cột có hơn 40% giá trị bị thiếu  total\_count = df.count()  missing\_counts\_row = missing\_counts.collect()[0].asDict()  missing\_percentage = {col: missing\_counts\_row[col] / total\_count for col in df.columns}  cols\_to\_drop = [col for col, pct in missing\_percentage.items() if pct > 0.4]  df = df.drop(\*cols\_to\_drop)  print(f"Đã xóa {len(cols\_to\_drop)} cột có tỷ lệ thiếu > 40%.")  print(f"Số cột còn lại sau khi xóa: {len(df.columns)}")  # Lấy danh sách các cột số và cột phân loại còn lại  numeric\_cols = [field.name for field in df.schema.fields if isinstance(field.dataType, (IntegerType, DoubleType, FloatType))]  categorical\_cols = [field.name for field in df.schema.fields if isinstance(field.dataType, StringType)]  # Điền giá trị thiếu cho các cột số bằng median  imputer\_numeric = Imputer(      inputCols=[c for c in numeric\_cols if c != 'SK\_ID\_CURR'],      outputCols=[f"{c}\_imputed" for c in numeric\_cols if c != 'SK\_ID\_CURR']  ).setStrategy("median")  print("Đã điền giá trị thiếu cho các cột số bằng giá trị trung vị.")  # Điền giá trị thiếu cho các cột phân loại bằng MODE  pipeline\_impute = Pipeline(stages=[imputer\_numeric])  df = pipeline\_impute.fit(df).transform(df)  for c in categorical\_cols:      mode\_val = df.groupBy(c).count().orderBy(F.desc("count")).first()[0]      df = df.withColumn(          f"{c}\_imputed",          F.when(F.col(c).isNull(), mode\_val).otherwise(F.col(c))      )  print("Đã điền giá trị thiếu cho các cột phân loại bằng giá trị mode.")  original\_cols\_to\_drop = [c for c in numeric\_cols if c != 'SK\_ID\_CURR'] + categorical\_cols  df = df.drop(\*original\_cols\_to\_drop)  for c in df.columns:      if c.endswith("\_imputed"):          df = df.withColumnRenamed(c, c.replace("\_imputed", ""))  print("Đã điền giá trị thiếu cho các cột còn lại.")  # Xử lý biến phân loại với StringIndexer và OneHotEncoder  # Lấy lại danh sách các cột chuỗi sau khi đã impute  categorical\_cols = [f.name for f in df.schema.fields if str(f.dataType) == 'StringType']  indexers = [StringIndexer(inputCol=c, outputCol=f"{c}\_index", handleInvalid="keep") for c in categorical\_cols]  encoders = [OneHotEncoder(inputCol=f"{c}\_index", outputCol=f"{c}\_vec") for c in categorical\_cols]  pipeline\_encode = Pipeline(stages=indexers + encoders)  df\_encoded = pipeline\_encode.fit(df).transform(df)  # Xóa các cột chuỗi gốc và các cột index trung gian  df\_encoded = df\_encoded.drop(\*categorical\_cols)  df\_encoded = df\_encoded.drop(\*[f"{c}\_index" for c in categorical\_cols])  print(f"Đã áp dụng StringIndexer và OneHotEncoder.")  print(f"Kích thước cuối cùng của dữ liệu đã xử lý: ({df\_encoded.count()}, {len(df\_encoded.columns)})")  print("\nHoàn tất! Dưới đây là schema của DataFrame đã sẵn sàng cho mô hình:")  df\_encoded.printSchema() |

Bảng 3.3‑4 Xử lý giá trị thiếu.

* Xử lý khoảng trắng thừa: Lấy tất cả các cột có kiểu dữ liệu là 'string' và xóa khoảng trắng như là double space,..

|  |
| --- |
| # Xác định các cột dạng chuỗi cần xử lý  # Lấy tất cả các cột có kiểu dữ liệu là 'string'  string\_columns = [field.name for field in df.schema.fields if isinstance(field.dataType, StringType)]  print(f"Các cột dạng chuỗi sẽ được xóa khoảng trắng: {string\_columns[:5]}...") # In ra 5 cột đầu  # Dùng vòng lặp và withColumn để xóa khoảng trắng  for c in string\_columns:      df = df.withColumn(c, trim(col(c)))  print("Đã xóa khoảng trắng thừa ở các cột dạng chuỗi.") |

Bảng 3.3‑5 Xử lý khoảng trắng.

* Chuẩn hóa dữ liệu: Đảm bảo các feature so sánh công bằng, mô hình học tốt và nhanh hơn.

|  |
| --- |
| # Lấy danh sách các cột số để chuẩn hóa (trừ cột ID và TARGET)  numeric\_cols\_for\_scaling = [      f.name for f in df.schema.fields if isinstance(f.dataType, (IntegerType, DoubleType, FloatType))      and f.name not in ['SK\_ID\_CURR', 'TARGET']  ]  print(f"\nChuẩn bị chuẩn hóa {len(numeric\_cols\_for\_scaling)} cột số...")  # 1. VectorAssembler: Gộp các cột số thành một cột vector duy nhất  assembler = VectorAssembler(inputCols=numeric\_cols\_for\_scaling, outputCol="features", handleInvalid="skip")  df\_assembled = assembler.transform(df)  # 2. StandardScaler: Tính toán mean/std và áp dụng chuẩn hóa  scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=True)  # Fit scaler vào dữ liệu để học các tham số  scaler\_model = scaler.fit(df\_assembled)  # Transform dữ liệu  df\_scaled = scaler\_model.transform(df\_assembled)  print("Đã chuẩn hóa dữ liệu thành công. Cột mới 'scaledFeatures' đã được thêm vào.")  df\_scaled.select("features", "scaledFeatures").show(5, truncate=False) |

Bảng 3.3‑6 Chuẩn hóa dữ liệu.

* Lưu dữ liệu đã làm sạch: Dữ liệu sau khi làm sạch được lưu lại dưới dạng file CSV để sử dụng cho giai đoạn tiếp theo.

|  |
| --- |
| output\_csv\_path = "C:\\Users\\DatGo\\OneDrive\\Documents\\Personal\_Project\\Home\_Credit\_Default\_Risk\\cleaned\_application\_test\_csv"  print(f"\nBắt đầu lưu DataFrame đã xử lý vào thư mục: {output\_csv\_path}")  # Xóa thư mục nếu đã tồn tại để tránh lỗi ghi đè  if os.path.exists(output\_csv\_path):      shutil.rmtree(output\_csv\_path)  df\_encoded.coalesce(1).write.mode("overwrite").option("header", "true").csv(output\_csv\_path)  print(f"Hoàn tất! Đã lưu thành công vào thư mục '{output\_csv\_path}'.")  print("Bạn sẽ tìm thấy một tệp CSV duy nhất bên trong thư mục đó.") |

Bảng 3.3‑7 Lưu dữ liệu đã làm sạch.

## Giai đoạn 2: Kỹ thuật tạo đặc trưng (Featuring.py)

Các thư viện cần thiết trong giai đoạn 2:

|  |
| --- |
| import pandas as pd  import numpy as np |

Bảng 3.4‑1 Thư viện cần thiết giai đoạn 2.

Load dữ liệu cần thiết:

|  |
| --- |
| # Tải dữ liệu  app\_train = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_train\_cleaned.csv')  app\_test = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_test\_cleaned.csv')  bureau = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\bureau.csv')  bureau\_balance = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\bureau\_balance.csv')  previous\_app = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\previous\_application.csv')  installments = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\installments\_payments.csv') |

Bảng 3.4‑2 Tải dữ liệu cho giai đoạn 2.

Lý do chuyển sang Pandas: Giai đoạn này đòi hỏi các thao tác tổng hợp, join và tính toán phức tạp trên nhiều bảng. Pandas cung cấp một API linh hoạt và mạnh mẽ hơn cho các tác vụ này so với Spark DataFrame API.

* Chi tiết các nhóm đặc trưng được tạo:
* Từ bureau & bureau\_balance: Tạo các đặc trưng tổng hợp về lịch sử tín dụng của khách hàng tại các tổ chức khác, ví dụ: số ngày trung bình/lớn nhất/nhỏ nhất của các khoản vay, tổng số tiền nợ, số lượng khoản vay đang hoạt động/đã đóng....

|  |
| --- |
| def process\_bureau\_and\_balance(bureau, bureau\_balance):      print("Xử lý bureau và bureau\_balance...")      # Tổng hợp bureau\_balance      bb\_agg = bureau\_balance.groupby('SK\_ID\_BUREAU')['MONTHS\_BALANCE'].agg(['count']).reset\_index()      bb\_agg.columns = ['SK\_ID\_BUREAU', 'MONTHS\_COUNT']      bureau = bureau.merge(bb\_agg, on='SK\_ID\_BUREAU', how='left')      # Tổng hợp bureau      bureau\_agg = bureau.groupby('SK\_ID\_CURR').agg({          'DAYS\_CREDIT': ['mean', 'max', 'min', 'var'],          'CREDIT\_DAY\_OVERDUE': ['mean', 'max'],          'AMT\_CREDIT\_SUM': ['mean', 'sum'],          'AMT\_CREDIT\_SUM\_DEBT': ['mean', 'sum'],      }).reset\_index()      bureau\_agg.columns = pd.Index(['SK\_ID\_CURR'] + ['BUREAU\_' + e[0] + "\_" + e[1].upper() for e in bureau\_agg.columns.tolist()[1:]])      # Tổng hợp các cột phân loại      bureau\_cat\_agg = pd.get\_dummies(bureau, columns=['CREDIT\_ACTIVE', 'CREDIT\_TYPE'], dummy\_na=True)      bureau\_cat\_agg = bureau\_cat\_agg.groupby('SK\_ID\_CURR').agg('sum').reset\_index()      dummy\_cols = [col for col in bureau\_cat\_agg.columns if 'CREDIT\_ACTIVE\_' in col or 'CREDIT\_TYPE\_' in col]      return bureau\_agg.merge(bureau\_cat\_agg[['SK\_ID\_CURR'] + dummy\_cols], on='SK\_ID\_CURR', how='left') |

Bảng 3.4‑3 Feature từ bureau, bureau\_balance.

* Từ previous\_application: Tạo các đặc trưng về hành vi vay vốn trước đây tại Home Credit, ví dụ: tỷ lệ hồ sơ được duyệt/từ chối, số tiền vay trung bình/lớn nhất/nhỏ nhất trước đây, tỷ lệ giữa số tiền đăng ký và số tiền được duyệt....

|  |
| --- |
| def process\_previous\_app(previous\_app):      print("Xử lý previous\_application...")      previous\_app[['DAYS\_FIRST\_DRAWING', 'DAYS\_FIRST\_DUE', 'DAYS\_LAST\_DUE\_1ST\_VERSION', 'DAYS\_LAST\_DUE', 'DAYS\_TERMINATION']] = \          previous\_app[['DAYS\_FIRST\_DRAWING', 'DAYS\_FIRST\_DUE', 'DAYS\_LAST\_DUE\_1ST\_VERSION', 'DAYS\_LAST\_DUE', 'DAYS\_TERMINATION']].replace(365243, np.nan)      previous\_app['APP\_CREDIT\_PERCENT'] = previous\_app['AMT\_APPLICATION'] / (previous\_app['AMT\_CREDIT'] + 1e-6)      prev\_app\_agg = previous\_app.groupby('SK\_ID\_CURR').agg({          'AMT\_ANNUITY': ['min', 'max', 'mean'],          'AMT\_CREDIT': ['min', 'max', 'mean'],          'APP\_CREDIT\_PERCENT': ['min', 'max', 'mean', 'var'],          'CNT\_PAYMENT': ['mean', 'sum'],      }).reset\_index()      prev\_app\_agg.columns = pd.Index(['SK\_ID\_CURR'] + ['PREV\_APP\_' + e[0] + "\_" + e[1].upper() for e in prev\_app\_agg.columns.tolist()[1:]])      prev\_app\_cat\_agg = pd.get\_dummies(previous\_app, columns=['NAME\_CONTRACT\_STATUS'], dummy\_na=True)      prev\_app\_cat\_agg = prev\_app\_cat\_agg.groupby('SK\_ID\_CURR').agg('sum').reset\_index()      dummy\_cols = [col for col in prev\_app\_cat\_agg.columns if 'NAME\_CONTRACT\_STATUS\_' in col]      return prev\_app\_agg.merge(prev\_app\_cat\_agg[['SK\_ID\_CURR'] + dummy\_cols], on='SK\_ID\_CURR', how='left') |

Bảng 3.4‑4 Feature từ previous\_app.

* Từ installments\_payments: Tạo các đặc trưng về hành vi thanh toán, ví dụ: số ngày trả nợ trễ/sớm trung bình, tỷ lệ giữa số tiền trả và số tiền phải trả mỗi kỳ....

|  |
| --- |
| def process\_installments(installments):      print("Xử lý installments\_payments...")      installments['PAYMENT\_PERC'] = installments['AMT\_PAYMENT'] / (installments['AMT\_INSTALMENT'] + 1e-6)      installments['PAYMENT\_DIFF'] = installments['AMT\_INSTALMENT'] - installments['AMT\_PAYMENT']      installments['DAYS\_PAST\_DUE'] = installments['DAYS\_ENTRY\_PAYMENT'] - installments['DAYS\_INSTALMENT']      installments['DAYS\_BEFORE\_DUE'] = installments['DAYS\_INSTALMENT'] - installments['DAYS\_ENTRY\_PAYMENT']      installments\_agg = installments.groupby('SK\_ID\_CURR').agg({          'PAYMENT\_PERC': ['mean', 'max', 'var'],          'PAYMENT\_DIFF': ['mean', 'max', 'sum'],          'DAYS\_PAST\_DUE': ['mean', 'max', 'sum'],          'DAYS\_BEFORE\_DUE': ['mean', 'min'],          'NUM\_INSTALMENT\_VERSION': ['nunique']      }).reset\_index()      installments\_agg.columns = pd.Index(['SK\_ID\_CURR'] + ['INSTALL\_' + e[0] + "\_" + e[1].upper() for e in installments\_agg.columns.tolist()[1:]])      return installments\_agg |

Bảng 3.4‑5 Feature từ installments.

* Lưu dữ liệu đã Feature Engineering: Dữ liệu sau khi làm giàu được lưu lại dưới dạng file CSV để sử dụng cho giai đoạn tiếp theo.

|  |
| --- |
| # Gộp tất cả features lại  print("\nGộp tất cả các đặc trưng...")  df\_train = app\_train.merge(bureau\_features, on='SK\_ID\_CURR', how='left')  df\_train = df\_train.merge(prev\_app\_features, on='SK\_ID\_CURR', how='left')  df\_train = df\_train.merge(installments\_features, on='SK\_ID\_CURR', how='left')  df\_test = app\_test.merge(bureau\_features, on='SK\_ID\_CURR', how='left')  df\_test = df\_test.merge(prev\_app\_features, on='SK\_ID\_CURR', how='left')  df\_test = df\_test.merge(installments\_features, on='SK\_ID\_CURR', how='left')  print("Gộp dữ liệu hoàn tất!")  print(f"Kích thước cuối cùng của tập train: {df\_train.shape}")  print(f"Kích thước cuối cùng của tập test: {df\_test.shape}")  # LƯU KẾT QUẢ CUỐI CÙNG  print("\nLưu các tệp dữ liệu đã được làm giàu...")  df\_train.to\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_train\_features.csv', index=False)  df\_test.to\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_test\_features.csv', index=False)  print("\nHoàn tất! Dữ liệu đã được làm giàu và lưu vào các tệp:")  print("- application\_train\_features.csv")  print("- application\_test\_features.csv") |

Bảng 3.4‑6 Lưu dữ liệu đã Feature Engineering.

## Giai đoạn 3: Xây dựng và huấn luyện mô hình (Modeling.ipynb)

Các thư viện cần thiết tải dữ liệu trong giai đoạn 3:

|  |
| --- |
| import pandas as pd  import numpy as np  import optuna  import lightgbm as lgb  import matplotlib.pyplot as plt  import xgboost as xgb  import shap  import seaborn as sns  import matplotlib.pyplot as plt  from imblearn.over\_sampling import SMOTE  from sklearn.pipeline import Pipeline  from sklearn.ensemble import StackingClassifier  from optuna.integration import LightGBMPruningCallback  from sklearn.model\_selection import StratifiedKFold, train\_test\_split, cross\_val\_predict  from sklearn.linear\_model import LogisticRegression  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import (precision\_recall\_curve, roc\_auc\_score, roc\_curve,                               precision\_score, recall\_score, f1\_score, accuracy\_score,                               confusion\_matrix, classification\_report, average\_precision\_score)  import warnings  warnings.filterwarnings('ignore')  # Các tệp này là kết quả từ các bước tiền xử lý trước  train\_df = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_train\_features.csv')  test\_df = pd.read\_csv(r'C:\Users\DatGo\OneDrive\Documents\Personal\_Project\Home\_Credit\_Default\_Risk\application\_test\_features.csv')  print("Tải dữ liệu thành công!")  print(f"Kích thước tập train: {train\_df.shape}")  print(f"Kích thước tập test: {test\_df.shape}") |

Bảng 3.5‑1 Thư viện cần thiết tải dữ liệu cho giai đoạn 3.

* Chuẩn bị dữ liệu cho modeling:

|  |
| --- |
| # --- Chuẩn bị dữ liệu ---  y = train\_df['TARGET']  train\_ids = train\_df['SK\_ID\_CURR']  test\_ids = test\_df['SK\_ID\_CURR']  X = train\_df.drop(['TARGET', 'SK\_ID\_CURR'], axis=1)  X\_test = test\_df.drop('SK\_ID\_CURR', axis=1)  # Mã hóa, căn chỉnh cột  X = pd.get\_dummies(X, drop\_first=True)  X\_test = pd.get\_dummies(X\_test, drop\_first=True)  common\_cols = list(set(X.columns) & set(X\_test.columns))  X = X[common\_cols]  X\_test = X\_test[common\_cols]  # Dọn dẹp infinity và NaN  X.replace([np.inf, -np.inf], np.nan, inplace=True)  X\_test.replace([np.inf, -np.inf], np.nan, inplace=True)  median\_values = X.median()  X.fillna(median\_values, inplace=True)  X\_test.fillna(median\_values, inplace=True)  # Tách dữ liệu  X\_train, X\_val, y\_train, y\_val = train\_test\_split(      X, y, test\_size=0.2, random\_state=42, stratify=y  )  # Scaling  scaler = StandardScaler()  X\_train\_scaled = scaler.fit\_transform(X\_train)  X\_val\_scaled = scaler.transform(X\_val)  X\_test\_scaled = scaler.transform(X\_test)  print("Chuẩn bị dữ liệu hoàn tất.") |

Bảng 3.5‑2 Chuẩn bị dữ liệu cho Modeling.

* Mô hình cơ sở (Baseline):
* Huấn luyện mô hình Logistic Regression với class\_weight='balanced' để làm thước đo so sánh ban đầu.

|  |
| --- |
| # --- Mô hình A: Logistic Regression ---  print("\n...Huấn luyện mô hình Baseline (Logistic Regression)...")  log\_reg = LogisticRegression(C=0.0001, random\_state=42, class\_weight='balanced')  log\_reg.fit(X\_train\_scaled, y\_train)  val\_probs\_lr = log\_reg.predict\_proba(X\_val\_scaled)[:, 1]  auc\_lr = roc\_auc\_score(y\_val, val\_probs\_lr)  print(f"==> Điểm AUC của Logistic Regression: {auc\_lr:.5f}")  val\_preds\_lr\_labels = (val\_probs\_lr >= 0.5).astype(int)  print("--- Báo cáo Phân loại (Classification Report) cho Logistic Regression ---")  print(classification\_report(y\_val, val\_preds\_lr\_labels, target\_names=['Repayer (0)', 'Defaulter (1)'])) |

Bảng 3.5‑3 Huấn luyện mô hình Logistic Regression (Baseline).

* Mô hình LightGBM với Cross-Validation:
* Sử dụng StratifiedKFold với 5-fold để đánh giá mô hình một cách ổn định.
* Trong mỗi fold, áp dụng SMOTE trên tập huấn luyện con để xử lý mất cân bằng.
* Sử dụng early\_stopping để tránh overfitting và tìm số vòng lặp tối ưu.

|  |
| --- |
| # ---------- Stratified K-Fold + hyperparams LightGBM (reduce overfitting) ------  skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)  # Các tham số cơ sở LightGBM được điều chỉnh để giảm tình trạng overfitting  lgb\_params = {      'objective': 'binary',      'metric': 'auc',      'boosting\_type': 'gbdt',      'learning\_rate': 0.03,   # nhỏ để ổn định      'num\_leaves': 31,        # không quá lớn      'max\_depth': 7,      'min\_data\_in\_leaf': 100, # tránh lá quá ít (giảm overfit)      'feature\_fraction': 0.8, # chọn subset feature mỗi iter      'bagging\_fraction': 0.8,      'bagging\_freq': 5,      'lambda\_l1': 0.1,      'lambda\_l2': 0.1,      'verbosity': -1,      'seed': 42  }  # Nếu imbalance nặng, set scale\_pos\_weight:  neg = (y\_train==0).sum()  pos = (y\_train==1).sum()  lgb\_params['scale\_pos\_weight'] = neg/pos  # Hàm để train LGB theo K-fold và trả về prediction\_probs cho validation folds  def lgb\_cv\_predict\_proba(X, y, params, skf, use\_smote=False, n\_rounds=10000):      oof\_preds = np.zeros(len(y))      models = []      for fold, (tr\_idx, val\_idx) in enumerate(skf.split(X, y)):          X\_tr, X\_val = X.iloc[tr\_idx], X.iloc[val\_idx]          y\_tr, y\_val = y[tr\_idx], y[val\_idx]          # SMOTE on train only          if use\_smote:              sm = SMOTE(random\_state=42)              X\_tr\_res, y\_tr\_res = sm.fit\_resample(X\_tr, y\_tr)          else:              X\_tr\_res, y\_tr\_res = X\_tr, y\_tr          dtrain = lgb.Dataset(X\_tr\_res, label=y\_tr\_res)          dval = lgb.Dataset(X\_val, label=y\_val, reference=dtrain)          model = lgb.train(              params,              dtrain,              num\_boost\_round=2000,              valid\_sets=[dtrain, dval],              callbacks=[lgb.early\_stopping(stopping\_rounds=100, verbose=False)]          )          preds = model.predict(X\_val, num\_iteration=model.best\_iteration)          oof\_preds[val\_idx] = preds          models.append(model)          print(f"Fold {fold+1} done. Best iter: {model.best\_iteration}")      return oof\_preds, models  # Chuẩn hóa tên các feature để tránh lỗi JSON của LightGBM  def sanitize\_columns(df):      df = df.copy()      df.columns = (          df.columns.str.replace('[^A-Za-z0-9\_]', '\_', regex=True)      )      return df  X\_train\_sanitized = sanitize\_columns(X\_train.reset\_index(drop=True))  # Chạy CV  oof\_preds\_lgb, lgb\_models = lgb\_cv\_predict\_proba(      X\_train\_sanitized,      y\_train.reset\_index(drop=True).values,      lgb\_params, skf,      use\_smote=True  )  # Metrics trên out-of-fold  print("AUC (oof):", roc\_auc\_score(y\_train, oof\_preds\_lgb))  print("Average precision (PR AUC):", average\_precision\_score(y\_train, oof\_preds\_lgb))  # ------- TÌM THRESHOLD TỐI ƯU  -------------  def find\_best\_threshold(y\_true, probs, metric='f1'):      precisions, recalls, thresholds = precision\_recall\_curve(y\_true, probs)      f1s = 2 \* (precisions \* recalls) / (precisions + recalls + 1e-12)      if metric == 'f1':          ix = np.nanargmax(f1s)          return thresholds[ix], precisions[ix], recalls[ix], f1s[ix]      elif metric == 'balanced\_accuracy':          # Tính toán với nhiều ngưỡng khác nhau bằng cách sử dụng ma trận nhầm lẫn (confusion matrix).          best = (0.0, 0.0, 0.0, 0.0)          for t in np.linspace(0.01, 0.99, 99):              y\_pred = (probs >= t).astype(int)              tn, fp, fn, tp = confusion\_matrix(y\_true, y\_pred).ravel()              recall0 = tn/(tn+fp) if (tn+fp)>0 else 0              recall1 = tp/(tp+fn) if (tp+fn)>0 else 0              bal = (recall0 + recall1)/2              if bal > best[0]:                  best = (bal, t, recall0, recall1)          return best[1], best[2], best[3], best[0]      else:          raise ValueError("metric must be 'f1' or 'balanced\_accuracy'")  best\_t, best\_prec, best\_rec, best\_f1 = find\_best\_threshold(y\_train.values, oof\_preds\_lgb, metric='f1')  print(f"Best threshold (F1): {best\_t:.3f}, precision={best\_prec:.3f}, recall={best\_rec:.3f}, f1={best\_f1:.3f}")  # Vẽ PR curve & mark threshold  precisions, recalls, thresholds = precision\_recall\_curve(y\_train.values, oof\_preds\_lgb)  plt.figure(figsize=(6,5))  plt.plot(recalls, precisions, label='PR curve')  plt.scatter([best\_rec],[best\_prec], color='red', label=f'best f1 @ {best\_t:.3f}')  plt.xlabel('Recall')  plt.ylabel('Precision')  plt.title('Precision-Recall curve (OOF LightGBM)')  plt.legend()  plt.grid(True)  plt.show()  # -------- EVALUATE trên tập val sử dụng best threshold ----------  # Lấy trung bình dự đoán từ tất cả các mô hình LightGBM trên tập X\_val.  val\_probs = np.mean([m.predict(X\_val, num\_iteration=m.best\_iteration) for m in lgb\_models], axis=0)  y\_pred\_val = (val\_probs >= best\_t).astype(int)  print("Validation classification report:")  print(classification\_report(y\_val, y\_pred\_val, digits=4)) |

Bảng 3.5‑4 Mô hình LightGBM tìm Threshold với Cross\_Validation.

* Mô hình Stacking:
* Xây dựng một Pipeline của Scikit-learn, bao gồm StandardScaler và StackingClassifier.
* Định nghĩa LightGBM và XGBoost làm base models, Logistic Regression làm final estimator.
* Sử dụng cv=skf bên trong StackingClassifier để đảm bảo meta-model được huấn luyện trên các dự đoán out-of-fold, tránh rò rỉ dữ liệu.

|  |
| --- |
| # ---------- STACKING: LightGBM + XGBoost ----------  # Huấn luyện các mô hình cơ sở (base learners) trên toàn bộ tập huấn luyện, rồi chồng (stack) chúng lại bằng Logistic Regression.  # Chúng ta sẽ xây dựng các lớp bao bọc (wrapper) tương thích với sklearn cho LightGBM và XGBoost.  from sklearn.base import BaseEstimator, ClassifierMixin  class LGBWrapper(BaseEstimator, ClassifierMixin):      def \_\_init\_\_(self, params=None, n\_estimators=1000):          self.params = params or {}          self.n\_estimators = n\_estimators          self.model = None      def fit(self, X, y):          dtrain = lgb.Dataset(X, label=y)          self.model = lgb.train(              self.params,              dtrain,              num\_boost\_round=self.n\_estimators,              callbacks=[lgb.log\_evaluation(0)]          )          # Thiết lập thuộc tính classes\_ để tương thích với sklearn.          self.classes\_ = np.unique(y)          return self      def predict\_proba(self, X):          return np.vstack([1-self.model.predict(X), self.model.predict(X)]).T      def predict(self, X):          return (self.model.predict(X) >= 0.5).astype(int)  class XGBWrapper(BaseEstimator, ClassifierMixin):      def \_\_init\_\_(self, params=None, n\_estimators=100):          self.params = params or {}          self.n\_estimators = n\_estimators          self.model = xgb.XGBClassifier(\*\*self.params, n\_estimators=self.n\_estimators, use\_label\_encoder=False, eval\_metric='auc')      def fit(self, X, y):          self.model.fit(X, y, verbose=False)          self.classes\_ = np.unique(y)          return self      def predict\_proba(self, X):          return self.model.predict\_proba(X)      def predict(self, X):          return self.model.predict(X)  # Các tham số của XGBoost  xgb\_params = {      'learning\_rate': 0.03,      'max\_depth': 6,      'subsample': 0.8,      'colsample\_bytree': 0.8,      'reg\_alpha': 0.1,      'reg\_lambda': 0.1,      'random\_state': 42  }  lgb\_wrapper = LGBWrapper(params=lgb\_params, n\_estimators=500)  xgb\_wrapper = XGBWrapper(params=xgb\_params, n\_estimators=500)  estimators = [      ('lgb', lgb\_wrapper),      ('xgb', xgb\_wrapper)  ]  stack\_clf = StackingClassifier(      estimators=estimators,      final\_estimator=LogisticRegression(max\_iter=1000),      cv=skf,      n\_jobs=-1,      passthrough=False  )  # Tùy chọn thêm bước chuẩn hóa (scaling) hoặc SMOTE vào pipeline cho mô hình stacker.  pipe = Pipeline([      ('scaler', StandardScaler()),      ('stack', stack\_clf)  ])  # Huấn luyện mô hình stacking trên toàn bộ tập dữ liệu huấn luyện  pipe.fit(X\_train, y\_train)  # Dự đoán trên tập dữ liệu kiểm thử (test set)  stack\_probs = pipe.predict\_proba(X\_test)[:,1]  # Tìm ngưỡng dự đoán tối ưu trên tập validation  oof\_stack = cross\_val\_predict(pipe, X\_train, y\_train, cv=skf, method='predict\_proba', n\_jobs=-1)[:,1]  best\_t\_stack, bp, br, bf = find\_best\_threshold(y\_train.values, oof\_stack, metric='f1')  print("Stack best threshold (F1):", best\_t\_stack, "f1:", bf)  val\_stack\_probs = pipe.predict\_proba(X\_val)[:,1]  y\_pred\_stack\_val = (val\_stack\_probs >= best\_t\_stack).astype(int)  # In báo cáo cho tập validation  print("Stacking classification report (validation):")  print(classification\_report(y\_val, y\_pred\_stack\_val, digits=4))  print("Stack ROC AUC (validation):", roc\_auc\_score(y\_val, val\_stack\_probs))  # ---------- Độ ổn định của Cross-Validation ----------  # Hiển thị độ biến động của các chỉ số đánh giá giữa các fold cho LightGBM  fold\_f1s = []  for tr\_idx, val\_idx in skf.split(X\_train, y\_train):      y\_true = y\_train.iloc[val\_idx].values      yprob = oof\_preds\_lgb[val\_idx]      \_, \_, \_, f1\_val = find\_best\_threshold(y\_true, yprob, metric='f1')      fold\_f1s.append(f1\_val)  print("LGB fold F1s:", fold\_f1s)  print("mean f1:", np.mean(fold\_f1s), "std:", np.std(fold\_f1s)) |

Bảng 3.5‑5 Mô hình Stacking LightGBM + Xgboost.

# Kết Quả và Thảo Luận

## Phân tích khám phá dữ liệu (EDA)

Phân tích ban đầu trên tập application\_train cho thấy sự mất cân bằng rõ rệt của biến mục tiêu, với chỉ 8.07% hồ sơ thuộc lớp TARGET=1.

A purple and blue pie chart

AI-generated content may be incorrect.

Hình 4.1‑1 Biểu đồ thể hiện sự mất cân bằng của dữ liệu.

Phân tích sâu hơn vào các đặc trưng quan trọng như EXT\_SOURCE\_2 và EXT\_SOURCE\_3 (điểm tín dụng từ nguồn bên ngoài) cho thấy sự khác biệt rõ rệt trong phân phối giữa hai nhóm khách hàng.

A graph of a graph

AI-generated content may be incorrect.

Hình 4.1‑2 Biểu đồ phân phối theo điểm tín dụng.

Nhóm khách hàng vỡ nợ (TARGET=1) có xu hướng có điểm EXT\_SOURCE thấp hơn đáng kể so với nhóm trả nợ đúng hạn, cho thấy đây là những đặc trưng có sức mạnh dự báo cao.

## Hiệu suất các mô hình

Quá trình xây dựng mô hình đã đi từ một baseline đơn giản đến một mô hình Stacking phức tạp. Bảng dưới đây tổng hợp hiệu suất của các mô hình chính trên tập kiểm định.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Comparison of Model Performance: | | | | | |
| Model | Accuracy | Precision | Recall | F1-Score | ROC AUC |
| Logistic Regression | 0.6959 | 0.1654 | 0.6840 | 0.2664 | 0.7568 |
| LightGBM | 0.8502 | 0.8502 | 0.2547 | 0.4443 | 0.7739 |
| Stacking | 0.8575 | 0.2649 | 0.4310 | 0.3281 | 0.7763 |

Bảng 4.2‑1 So sánh hiệu suất các mô hình.

Mô hình Stacking cho thấy hiệu suất tốt nhất trên tất cả các thước đo chính, đặc biệt là ROC AUC. Điều này chứng tỏ việc kết hợp nhiều mô hình đã giúp nắm bắt các quy luật phức tạp hơn trong dữ liệu.

Báo cáo phân loại chi tiết của mô hình Stacking cuối cùng trên tập kiểm định được trình bày dưới đây.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stack ROC AUC: 0.7762914421966179 | | | | |
| Stacking classification report: | | | | |
| precision | | recall | f1-score | support |
| 0 | 0.9471 | 0.8950 | 0.9203 | 56538 |
| 1 | 0.2649 | 0.4310 | 0.3281 | 4965 |
| accuracy | 0.8575 | | | 61503 |
| macro avg | 0.6060 | 0.6630 | 0.6242 | 61503 |
| weighted avg | 0.8920 | 0.8575 | 0.8725 | 61503 |

Bảng 4.2‑2 Báo cáo phân loại chi tiết cho mô hình Stacking.

## Phân tích tầm quan trọng của đặc trưng

Sử dụng SHAP, chúng tôi đã phân tích các yếu tố có ảnh hưởng lớn nhất đến dự đoán của mô hình Stacking.

A bar graph with blue and white text

AI-generated content may be incorrect.

Hình 4.3‑1 Biểu đồ SHAP tổng hợp thể hiện tầm quan trọng và ảnh hưởng của các đặc trưng.

* Thảo luận sâu:
* EXT\_SOURCE\_2 & EXT\_SOURCE\_3: Hai đặc trưng này có ảnh hưởng mạnh nhất.
* DAYS\_BIRTH: Đặc trưng này đại diện cho tuổi của khách hàng. Giá trị DAYS\_BIRTH khách hàng lớn tuổi có xu hướng làm giảm rủi ro, trong khi khách hàng trẻ tuổi hơn có rủi ro cao hơn.
* DAYS\_EMPLOYED: Thời gian làm việc cũng là một yếu tố quan trọng. Khách hàng có thời gian làm việc lâu hơn (giá trị DAYS\_EMPLOYED thấp) có rủi ro thấp hơn.

## Phân tích lỗi (Error Analysis)

Phân tích ma trận nhầm lẫn của mô hình cuối cùng trên tập kiểm định giúp hiểu rõ hơn về các loại lỗi mà mô hình mắc phải.

A screenshot of a computer screen

AI-generated content may be incorrect.

Hình 4.4‑1 Ma trận nhầm lẫn của các mô hình.

* False Positives (FP): Mô hình dự đoán sai một khách hàng tốt thành khách hàng vỡ nợ. Lỗi này dẫn đến mất cơ hội kinh doanh.
* False Negatives (FN): Mô hình bỏ sót một khách hàng vỡ nợ. Lỗi này dẫn đến tổn thất tài chính trực tiếp và là loại lỗi nguy hiểm hơn trong bài toán tín dụng.

Việc tối ưu hóa theo F1-Score đã giúp tạo ra một sự cân bằng hợp lý giữa hai loại lỗi này.

# Kết Luận và Hướng Phát Triển

## Tổng kết kết quả

Dự án đã thành công trong việc xây dựng một quy trình học máy toàn diện để giải quyết bài toán dự đoán rủi ro tín dụng. Bằng cách kết hợp xử lý dữ liệu lớn với Spark, kỹ thuật tạo đặc trưng chuyên sâu và các mô hình ensemble tiên tiến, mô hình Stacking cuối cùng đã đạt được hiệu suất cao và đáng tin cậy. Kết quả phân tích cũng đã chỉ ra những yếu tố rủi ro chính, cung cấp những thông tin hữu ích cho hoạt động kinh doanh.

A screen shot of a computer screen

AI-generated content may be incorrect.

Hình 5.1‑1 Biểu đồ phân tán tỉ lệ vỡ nợ.

## Đóng góp của dự án

* Về mặt kỹ thuật: Đã trình bày một pipeline hoàn chỉnh, kết hợp các công nghệ khác nhau (Spark, Pandas, Scikit-learn) một cách hiệu quả.
* Về mặt kinh doanh: Mô hình được xây dựng có thể được tích hợp vào quy trình thẩm định tín dụng để đưa ra các quyết định cho vay nhanh chóng, khách quan hơn, giúp giảm thiểu rủi ro và tối ưu hóa lợi nhuận.

## Hạn chế của dự án

* Dữ liệu tĩnh: Mô hình được xây dựng trên dữ liệu tại một thời điểm, chưa phản ánh được sự thay đổi trong hành vi của khách hàng hay các yếu tố kinh tế vĩ mô theo thời gian.
* Tạo đặc trưng: Mặc dù đã tạo ra nhiều đặc trưng, vẫn còn nhiều tiềm năng để khám phá và tạo ra các đặc trưng phức tạp hơn.

## Hướng phát triển trong tương lai

* Triển khai và giám sát: Đưa mô hình vào môi trường A/B testing để đánh giá hiệu quả thực tế và xây dựng hệ thống giám sát hiệu suất (model monitoring).
* Huấn luyện lại định kỳ (Retraining): Xây dựng quy trình MLOps để tự động huấn luyện lại mô hình với dữ liệu mới, đảm bảo mô hình luôn được cập nhật.
* Nghiên cứu mô hình mới: Thử nghiệm các kiến trúc mô hình phức tạp hơn như mạng nơ-ron sâu (Deep Neural Networks) trên dữ liệu dạng bảng.

## Phụ lục và tài liệu tham khảo

Phụ lục:

* Source datasets: [Home Credit Default Risk | Kaggle](https://www.kaggle.com/competitions/home-credit-default-risk/data)
* Source code và PowerBI: [https://drive.google.com/file/d/1IgeqDgsi\_ofqwFi2KY8hnA3zJGLOe2tk/view?usp=sharing](https://drive.google.com/file/d/1IgeqDgsi_ofqwFi2KY8hnA3zJGLOe2tk/view?usp=sharing%20)

Tài liệu tham khảo:

* Python for Data Analysis – Wes McKinney (3rd Edition): [Python for Data Analysis, 3E](https://wesmckinney.com/book/)
* Python Data Science Handbook**:** [Python Data Science Handbook | Python Data Science Handbook](https://jakevdp.github.io/PythonDataScienceHandbook/)