Project Documentation: Customer Lifetime Value Prediction

Title of the Project

Customer Lifetime Value Prediction using RFM Analysis and XGBoost Classifier

Objective

The goal of this project is to predict Customer Lifetime Value (LTV) for an online retail dataset by segmenting customers into Low, Mid, and High LTV categories based on their purchasing behavior. Using Recency, Frequency, and Monetary (RFM) features from the first 6 months of data, we cluster customers by their subsequent 6-month revenue and train a machine learning model to classify new customers into these LTV groups. This enables businesses to identify high-value customers for targeted marketing, addressing challenges like class imbalance.

Dataset Used

- **Name**: Online Retail II
- **Source**: UCI Machine Learning Repository (assumed based on data structure: 536365, 85123A, ...).
- **Description**: Transaction records from a UK-based online retailer, spanning December 1, 2010, to December 9, 2011, with 541,909 rows and 8 columns:
    - Invoice: Transaction ID (e.g., 536365).
    - StockCode: Product code (e.g., 85123A).
    - Description: Product name (e.g., WHITE HANGING HEART T-LIGHT HOLDER).
    - Quantity: Items purchased (e.g., 6).
    - InvoiceDate: Purchase date/time (e.g., 01-12-2010 08:26).
    - Price: Unit price in GBP (e.g., 2.55).
    - Customer ID: Customer identifier (e.g., 17850).
    - Country: Customer location (e.g., United Kingdom).

Preprocessing:

Dropped rows with missing Customer ID (~135,080 removed).

Filtered out negative Quantity values (returns), retaining positive transactions.

Computed Total_Spending as Quantity * Price.

Split into two 6-month periods: first for RFM features, second for LTV target (m6_Revenue).

Model Chosen

Model: XGBoost Classifier

Reason: Selected for its superior handling of imbalanced datasets and multi-class classification, outperforming simpler models like Random Forest in tuning flexibility.

Configuration:

n_estimators=400: Many trees for robust learning.

max_depth=6: Balances complexity and overfitting.

learning_rate=0.03: Slow learning for precision.

scale_pos_weight=4: Weights minority classes.

gamma=2: Regularizes tree splits.

min_child_weight=5: Prevents overfitting to small classes.

random_state=42: Reproducibility.

eval_metric='mlogloss': Optimizes multi-class log loss.

Pipeline:

RFM features extracted and enhanced with Revenue_per_Transaction.

K-Means clustering (3 clusters) on m6_Revenue to define LTV labels.

SMOTE oversampling to balance classes (Low: 2000+, Mid: 1200, High: 300).

Probability threshold of 0.9 for High class predictions.

Performance Metrics

Test Accuracy: 0.87

Class Metrics (Precision | Recall | F1):

Low: 0.95 | 0.85 | 0.90

High precision and solid recall reflect strong majority-class performance.

Mid: 0.51 | 0.75 | 0.61

Decent F1, with high recall but lower precision indicating overprediction.

High: 0.50 | 0.50 | 0.50

Balanced precision and recall, impressive given ~2-5 original samples.

Evaluation:

87% accuracy shows overall reliability.

Low class excels due to data abundance.

Mid and High classes improved significantly from initial runs (High F1 from 0.22 to 0.50) but are constrained by limited samples.

Challenges & Learnings

Challenge 1: Class Imbalance

Problem: High LTV class had ~2-5 samples vs. 1000s in Low, skewing initial predictions (High F1 at 0.22).

Solution: Used SMOTE for oversampling and scale_pos_weight in XGBoost.

Learning: Synthetic data helps, but real samples are irreplaceable for minority classes.

Challenge 2: Precision for Mid and High

Problem: Mid precision dropped to 0.51, High started at 0.14 due to false positives.

Solution: Applied regularization (gamma, min_child_weight) and a 0.9 threshold for High.

Learning: Threshold tuning trades recall for precision effectively in imbalanced settings.

Challenge 3: Data Scarcity

Problem: Limited High LTV data capped performance.

Solution: Conservative SMOTE (300 for High) and feature engineering (Revenue_per_Transaction).

Learning: More data or a longer LTV period (e.g., 12 months) could push metrics higher.