Lifetime Value (LTV) Prediction with PySpark

Project Overview

This repository implements an end-to-end solution for forecasting the lifetime value (LTV) of user cohorts and individual users using Apache Spark (PySpark). By combining historical one-year revenue labels with short-term engagement data, we train Gradient Boosted Tree (GBT) models to predict annualized revenue. The analysis is split into two parallel pipelines:

- **Cohort-Level Pipeline**: Aggregates user sign-ups into cohorts by date and season, extracting statistical features to forecast per-cohort revenue.
- **User-Level Pipeline**: Encodes each user's first 15 days of activity into behavior and revenue features, then annualizes short-term forecasts.

Both pipelines leverage Spark ML for feature transformations, hyperparameter tuning, and cross-validated model selection.

Table of Contents

1. Repository Structure
2. Getting Started
3. Prerequisites
4. Data Description
5. Feature Engineering
6. Cohort Features
7. User Features
8. Modeling Pipeline
9. Transformations
10. Algorithm & Tuning
11. Evaluation Strategy
12. Usage
13. Results & Metrics
14. Contributing
15. License

Repository Structure

```
├── data/                    # Raw input CSV and Parquet files
│   ├── engagement_events/   # Event logs: timestamps, user_id, event_type, OS, country
│   ├── revenue_history/     # Full first-year revenue per user (historical cohorts)
│   └── revenue_15d/         # 15-day revenue for current-year users
├── notebooks/               # Analysis and modeling workflows
```

```
│   ├── Codeway.ipynb         # Spark setup, data ingestion, initial EDA
│   ├── Cohort Pre-processing.ipynb # Cohort definition & feature extraction
│   ├── Cohort Prediction.ipynb     # Model training & evaluation for cohorts
│   └── User Prediction.ipynb       # User-level LTV forecasting pipeline
├── models/                   # Serialized Spark ML models (saved with version
tags)
├── requirements.txt          # Python dependencies
└── README.md                 # Project overview and instructions
```

Getting Started

Follow these steps to reproduce the analysis and train the models locally:

1. **Clone the repository**

   ```
   git clone https://github.com/your-org/ltv-prediction-spark.git
   cd ltv-prediction-spark
   ```

2. **Install dependencies**

   ```
   pip install -r requirements.txt
   ```

3. **Start a Spark-enabled Jupyter Notebook** Ensure your `SPARK_HOME` and `JAVA_HOME` are correctly set, then launch:

   ```
   jupyter notebook
   ```

4. **Run notebooks in sequence** under the PySpark kernel:
5. `Codeway.ipynb`
6. `Cohort Pre-processing.ipynb`
7. `Cohort Prediction.ipynb`
8. `User Prediction.ipynb`

Prerequisites

- Python 3.8+
- Apache Spark 3.x
- Java JDK 8 or higher
- `pyspark`, `pandas`, `numpy`, `scikit-learn` (for local processing), and `matplotlib` (for plotting)

Data Description

Raw data files are stored in the `data/` directory:

- **engagement_events/**: Logs of user interactions (e.g. `free_trial`, `renewal`, `paywall`) with timestamps, user IDs, operating systems, and country codes.
- **revenue_history/**: CSVs of complete first-year revenue for users who have matured past 365 days.
- **revenue_15d/**: CSVs of revenue realized in the first 15 days for all users (used as proxy for current-year forecasting).

All input files are read into Spark DataFrames, cleaned for nulls, and schema-validated before feature extraction.

Feature Engineering

Cohort Features

For each daily signup cohort, we compute:

- **cohort_size**: Number of users in the cohort.
- **mean and standard deviation** of per-user event counts (`auto_renew_off`, `free_trial`, `paywall`, `refund`, `renewal`, `subscribe`).
- **mean_event_hour** and **std_event_hour**: Temporal activity patterns.
- **platform distribution**: Pivoted columns for iOS vs. Android vs. iPadOS usage.
- **historical revenue**: `mean_revenue_1y` and `std_revenue_1y` for that cohort.
- **season**: One-hot encoding of quarter (Q1–Q4) to capture macro trends.

These 20–25 features form a compact statistical summary of each cohort's composition and past performance.

User Features

For each individual user's first 15 days, we extract:

- **raw counts** of events (`auto_renew_off`, `free_trial`, `paywall`, `refund`, `renewal`, `subscribe`).
- **stickiness_ratio**: Active days ÷ 15 days (normalized retention).
- **avg_event_hour**: Average hour of user activity.
- **total_revenue_15d**: Revenue within the first 15-day window.
- **categorical context**: One-hot encodings for season, country, and operating system.

This \~13-dimensional vector captures early engagement rhythms, short-term monetization, and user context.

Modeling Pipeline

Transformations

Each pipeline uses Spark ML's `Pipeline` API with stages:

1. **StringIndexers** for categorical columns (season, country, OS).
2. **VectorAssembler** to combine numeric and indexed features.
3. **StandardScaler** to normalize feature magnitudes.
4. **GBTRegressor** for regression.

Algorithm & Tuning

We optimize three hyperparameters:

- `maxDepth` (tree depth)
- `maxIter` (number of boosting iterations)
- `stepSize` (learning rate)

Using `ParamGridBuilder` and `CrossValidator`, we perform k-fold cross-validation on the training split, optimizing MAE. The best combination is automatically selected by Spark.

Evaluation Strategy

A strictly **time-aware hold-out** is enforced: we train on all cohorts/users before a cutoff date and validate on the immediately following time window. Performance is measured using **Mean Absolute Error (MAE)** and **relative MAE** on the hold-out sets. For the user-level pipeline, we evaluate the 15-day forecasts directly, reporting both the MAE and the relative MAE (mean). For the cohort-level pipeline, we compute the mean and standard deviation of the 15-day MAE across cohorts as well as the relative MAE (mean and std).

For the user pipeline, forecasts are annualized by multiplying 15-day predictions by 365/15 and evaluated similarly.

Usage

Clone the repository, install dependencies, and step through the notebooks to reproduce feature extraction, model training, and evaluation. To integrate into a production workflow, consider exporting the final Spark model (via `model.write().overwrite().save()`) and loading it in a streaming or batch scoring job.

Results & Metrics

Based on the hold-out evaluation in the notebooks, the models achieved the following performance:

- **User-level model**: 15-day MAE of **0.01681**, with a relative MAE (mean) of **3.36%**.
- **Cohort-level model**: 15-day MAE (mean) of **0.02084**, 15-day MAE (std) of **0.09662**, relative MAE (mean) of **4.12%**, and relative MAE (std) of **3.41%**.

These metrics reflect accuracy over the short-term window required by the case study and are used both for model selection and ongoing monitoring.

Contributing

Feel free to submit issues or pull requests. For new feature experiments (e.g., adding survival analysis or deep learning models), please document each run's hyperparameters and metrics, ideally using an experiment tracker like MLflow.

License