WSDM - KKBox's Customer Churn

Big Data Analytics

Ehtisham Ul Haq & Ijaz Khan

27th November, 2025

Phase 1: Problem Identification

Use case: “WSDM - KKBox's Customer Churn”

Scope: I will build a binary classification model to predict whether a subscriber will fail to renew their membership within 30 days of expiration, utilizing temporal transaction history to flag churn risks.

Phase 2: Data Sourcing

Dataset link: https://www.kaggle.com/competitions/kkbox-churn-prediction-challenge/data

Description: This is a relational dataset. It is not a single CSV file; it consists of four distinct tables linked by a unique User ID (msno).

This is a large-scale "Big Data" classification problem where we must predict if a subscriber will churn (fail to renew their membership) within 30 days of expiration. Unlike standard flat-file datasets, this requires engineering features from a relational database consisting of **~6.7 million users, ~21 million financial transactions, and ~400 million daily listening logs (30GB).**

Phase 3: Pipeline design

Since the user_logs file is ~30GB and 400M rows, we cannot load it all into memory at once. We need a **Batch Processing Architecture.**

1. **High-Level Architectural Diagram**
    - **Data Ingestion (Batch Layer):**
        Tool: Apache Spark (PySpark).
        Strategy: We treat the CSV files as the "Raw Zone." Since this is historical data, we do not need Kafka (Real-time). We will ingest the raw CSVs directly into Spark DataFrames.
    - **Storage (Data Lake):**
        Format: Convert the raw CSVs immediately into Parquet format.
        Why? CSV is row-based and slow. Parquet is columnar and compressed. Searching 400M rows in Parquet is 10-50x faster than CSV.
        Location: In a cloud scenario: AWS S3 or Google Cloud Storage. For your local case study: A dedicated local directory acting as the HDFS.
    - **Processing (ETL & Aggregation):**
        Tool: PySpark.
        Task: The user_logs table must be aggregated before machine learning. We cannot feed 400M rows to a model. We use Spark to group by msno (User ID) and calculate summaries (e.g., Average listening time per week).
        Output: A "Gold Table" (Structured Feature Set) that is much smaller (e.g., 6.7M rows, one per user) which can then be saved to CSV/Parquet.

- **Analytics & Modeling:**
  Tools: XGBoost (Python) or LightGBM.
  Strategy: Once Spark has reduced the data size from 30GB to ~1GB (by aggregating logs), we can switch to Python/Pandas for the actual model training on a single powerful machine.
- **Visualization:**
  Tool: Tableau or PowerBI.
  Usage: Connect the BI tool to the final predictions to show "Projected Churn Revenue Risk" to stakeholders.

Phase 4: Machine Learning Methodology

1. **Pre-Processing Strategy:**
   - **Handling Missing Values:**
     Gender: Contains many nulls. Strategy: Create a new category called "Unknown". Do not drop these rows; the fact that a user didn't fill out their profile is a predictive signal in itself.
     Age (bd): Contains outliers (e.g., -5, 0, or 200). Strategy: Filter valid ages (e.g., 10 to 80). For invalid/missing, use Median Imputation (Median is robust to outliers; Mean is not).
   - **Handling Categorical Variables:**
     City / Registered_via: Nominal variables. Strategy: Use Label Encoding (converting categories to integers 1, 2, 3...).
     Why? Tree-based models (XGBoost/LightGBM) handle Label Encoding very well. One-Hot Encoding would create too many columns (sparsity) for variables with many categories.
2. **Feature Engineering (The Winning Factor):**
   This is where the model is made or broken. You need to turn raw logs into behavioral insights.
   Recency (RFM): Days since the last user_log entry. (Inactive users likely churn).
   Frequency: Average days active per week over the last month.
   - **Engagement Quality:**
     %_songs_completed: (num_100 / total_songs). High completion rates = Happy user.
     %_songs_skipped: (num_25 / total_songs). High skip rates = Dissatisfied user.
   - **Financial Behavior:**
     price_per_day: Did the user downgrade their plan recently?
     auto_renew_change: Did they recently toggle auto-renew off? (Strongest predictor).
3. **Scaling:**

- **Recommendation:** No Scaling needed (mostly).
- **Why:** I am recommending Tree-based algorithms (XGBoost). These algorithms rely on splitting rules (e.g., "If Age < 30"), so they are invariant to the scale of the data.
- **Note:** If you were using Logistic Regression or Neural Networks, you would strictly need Standardization (Z-Score).

4. **Algorithm Recommendation:**
    - **Recommendation:** LightGBM (Light Gradient Boosting Machine).
    - **Justification:**
        - **Speed:** It is significantly faster than XGBoost on large datasets (millions of rows).
        - **Handling Nulls:** It handles missing values natively (you don't strictly have to impute everything).
        - **Imbalance:** It has built-in parameters (is_unbalance=True or scale_pos_weight) to handle the churn imbalance.
        - **Accuracy:** It currently dominates tabular data competitions on Kaggle.
5. **Dataset Analysis:**
    - **Imbalanced:** Yes. The churn rate is likely roughly 5-10%. We have far more "Non-Churners" than "Churners."
    - **Temporal:** Yes. User behavior changes over time. A user who listened to 50 songs in Jan but 0 in Feb is highly likely to churn in March.
    - **High-Dimensional:** Originally high, but after feature engineering, we aim for ~50-100 strong features.

Phase 5: Implementation Plan

1. **Library Selection:**
    - **Data Processing:** pyspark (Crucial for the 400M logs), pandas (For the final training set).
    - **Modeling:** lightgbm (or xgboost), scikit-learn (for splitting and metrics).
    - **Visualization:** matplotlib, seaborn.
2. **Logic Flow (Pseudo-Code):**

```
# --- STEP 1: BIG DATA PROCESSING (PySpark) ---
def process_logs_spark():
    logs = spark.read.csv("user_logs.csv")

    # Feature Engineering on Big Data
    user_features = logs.groupBy("msno").agg(
        avg("total_secs").alias("avg_listen_time"),
        sum("num_unq").alias("total_unique_songs"),
```

```
        max("date").alias("last_seen_date")
    )
    return user_features

# --- STEP 2: MERGE & PREPARE (Pandas) ---
def prepare_training_data():
    members = pd.read_csv("members.csv")
    train_labels = pd.read_csv("train.csv")
    transactions = pd.read_csv("transactions.csv")

    # Merge everything onto the Train Labels
    df = train_labels.merge(members, on="msno", how="left")
    df = df.merge(process_logs_spark().toPandas(), on="msno", how="left") # Import
Spark result

    return df

# --- STEP 3: MODELING (LightGBM) ---
def train_model(df):
    X = df.drop(["is_churn", "msno"], axis=1)
    y = df["is_churn"]

    # Split Data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

    # Initialize LightGBM with Imbalance handling
    model = lgb.LGBMClassifier(
        n_estimators=1000,
        learning_rate=0.05,
        scale_pos_weight=9  # Assuming 1:9 ratio of Churn:Non-Churn
    )

    model.fit(X_train, y_train)

    return model
```

3. **Evaluation Metrics:**
    - **Primary Metric:** Log Loss (Logarithmic Loss).
    - **Why?** This is the official metric for the WSDM competition. It measures the uncertainty of your probabilities. Being 100% wrong is penalized heavily.

- **Business Metric: Recall (Sensitivity).**
- **Why?** In Churn prediction, False Negatives are expensive (We predicted they would stay, but they left, so we lost revenue). We prefer False Positives (We thought they would leave, sent them a coupon, but they were going to stay anyway). High Recall ensures we catch the churners.