Python for Data Science

Churn Prediction

Hae In Keum
Julius Enderwitz

Project overview

Predicting User Churn from Behavioral Logs

- **<u>Goal:</u>** Predict user churn in a 10-day horizon using event-level log data
  - Transform the clickstream logs into user-level features without leakage → observe windows
- **<u>Our approach:</u>**
  - Window-based feature construction to capture recent behavior of individual users
  - Multiple backward-sliding snapshots per user to maximize training data usage
  - Use tree-based ensemble models (LGBM) to capture non-linearities and interactions
  - Explicit threshold tuning on the validation set to handle class imbalance
- **<u>Idea:</u>** Churn is not driven by one action, but gradual disengagement patterns

Window Design and Dataset Construction

- **Backward-sliding windows:**
  - Avoid time leakage, capture behavioral dynamics instead of full-sample averages
- **Windows design:**
  - Observation window (OBS) of 14 days, recent subwindow (SUB_OBS) of 7 days
  - Prediction window (PRED) of 10 days
  - Creates 2 backward-sliding snapshots per user
- **Resulting data:**
  - Users who churned during observation are omitted since since they display churn signals and would not have been flagged in PRED (hidden churn signals – *alive_users*)
  - User-level data: one observation is one tuple of observation window features and a binary churn flag

Feature Engineering Philosophy

Activity Volume & Frequency

User Feedback & Interaction

Advertising Exposure

Temporal & Recency Signals

- Features encode changes in behavior, not only levels.
- Zero-imputation when no data is available: NaN songs listened to means zero songs, this is a meaningful signal that should not be dropped.

Model Comparison

- <u>Models evaluated:</u>
  - Logistic Regression, Random Forest, AdaBoost, LightGBM
- <u>Advantage of boosted tree models:</u>
  - Learns sequentially from errors, captures weak but consistent churn signals
  - Thus, more sample-efficient than Random Forest
- <u>Tradeoff:</u>
  - High flexibility can lead to overfitting
  - Feature selection, regularization, evaluation on a held-out validation set

Pipeline

SelectKBest

LGBMClassifier

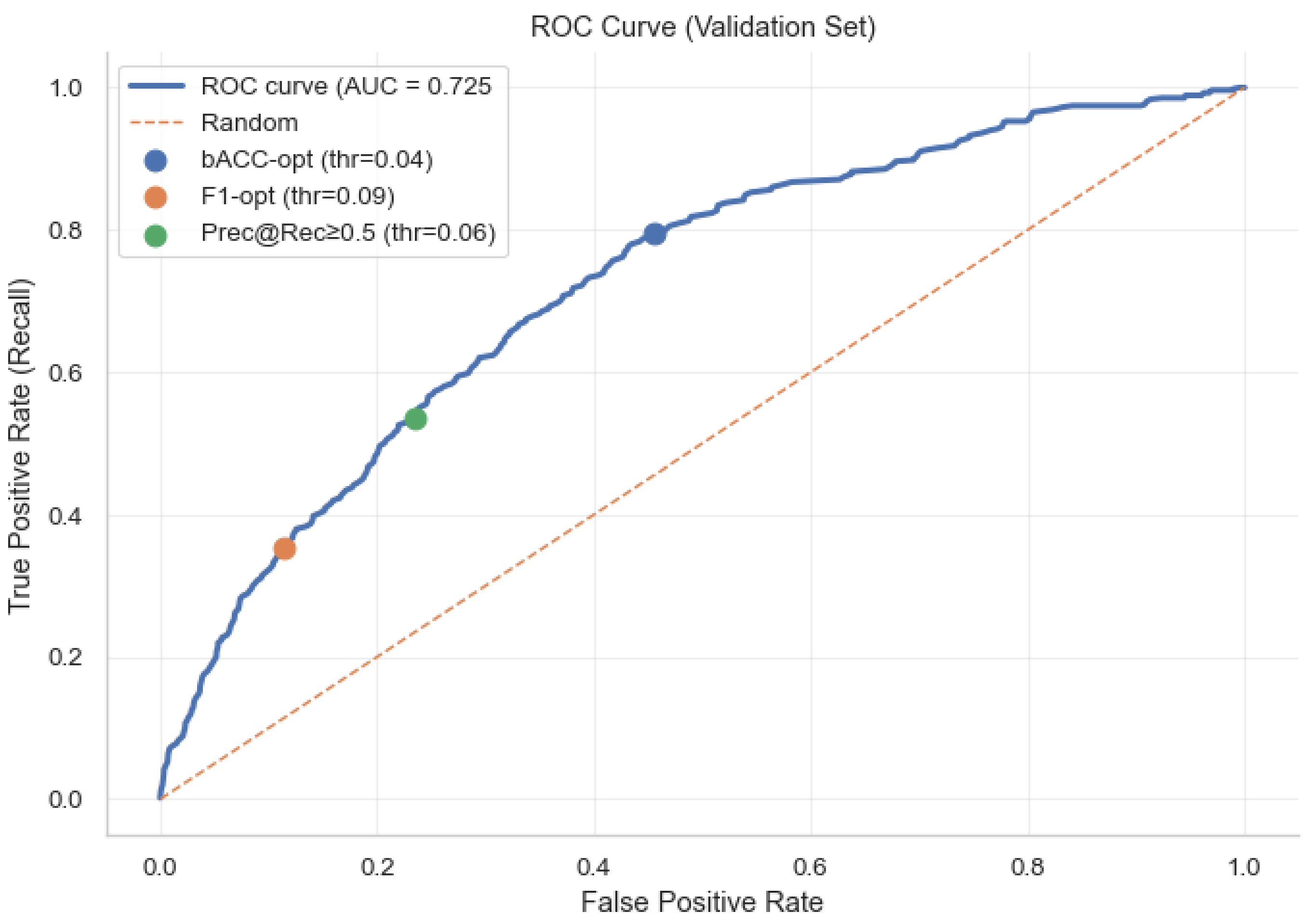
Parameters

boosting_type	'gbdt'
num_leaves	31
max_depth	5
learning_rate	0.03
n_estimators	5000
subsample_for_bin	200000
objective	'binary'
class_weight	None
min_split_gain	0.0
min_child_weight	0.001
min_child_samples	100
subsample	0.8
subsample_freq	1
colsample_bytree	0.6
reg_alpha	0.0
reg_lambda	10.0
random_state	42
n_jobs	-1
importance_type	'split'

Training, Validation , Threshold Tuning

- Training Strategy:
  - Stratified train / validation split (80/20)
  - RandomizedSearchCV with hyperparameter grid and k features
  - SelectKBest: selects k features with highest estimated mutual information with target
    - Reduction in uncertainty about the target after knowing the feature.
- Optimization metric: balanced accuracy
- Best model: LightGBMClassifier to predict validation set churn probabilities
- Threshold Tuning:
  - Default 0.5 is suboptimal for class-imbalanced churn data
  - Tested thresholds: [0.01, 0.95] in steps of 0.015, best threshold was **4%**
  - Balanced accuracy on the held-out validation set with $\tau = 0.04$ was **67.07%**
  - Threshold tuning is a business-aware decision: do not use raw accuracy, detect potential churners early for higher recall

Results and Model Behavior




```
[Threshold tuning by bAcc on held-out validation]
Best threshold: 0.04
Best Balanced Accuracy: 0.6707
              precision    recall  f1-score   support

           0       0.98      0.55      0.70      5102
           1       0.09      0.80      0.16       275

    accuracy                           0.56      5377
   macro avg       0.53      0.67      0.43      5377
weighted avg       0.93      0.56      0.67      5377
```

Performance metrics on the held-out validation set

- <u>Test prediction logic:</u>

Same feature pipeline, use the observation window ending at the cutoff to predict churn in the 10 days after

Shortcomings of this Model

- Symptoms:
  - Strong validation and public Kaggle performance
  - Weaker private Kaggle score → poor generalization
- Root Causes:
  - Independent feature and hyperparameter selection, iterations do not learn from each other
  - Performance is tied too strongly to validation set and public Kaggle score

- Attempts to fix it:
  - Reduce feature set: without uninformative features
  - There is no single underlying distribution: model users individually, specify user-specific churn models instead of a one-fits-all model
  - Use Optuna study with iterations learning from previous ones
  - Do not rely on the public Kaggle score as proxy for performance on unseen data