Table of Contents

---

1. Project Setup and Initialization

This initial step configures the computing environment and imports necessary libraries.

Library Imports: The necessary Python libraries are imported: `os` and `pandas` for data manipulation.

Define File Paths: Local paths are defined for the main training data (`train (6).csv`) and the churn label file (`train_churn_labels.csv`).

2. Data Loading and Initial Exploration

The primary feature dataset is loaded and summarized to understand its structure and statistical properties.

Load Data: The `train (6).csv` file is read into a pandas DataFrame named `data`.

Descriptive Statistics: The `.describe()` method is executed on the DataFrame, showing count, mean, standard deviation, and quartile information for the numeric columns. This summary is also saved to a file named `summary_features.csv`. The data contains 192 columns in the summary, with varying counts of non-missing values.

3. Target Label Analysis

The dataset containing the target variable is loaded to examine the class distribution.

Load Labels: The `train_churn_labels.csv` file is loaded into a DataFrame named `labels`.

Label Values: The unique values in the `Label` column are identified as -1 and 1.

Class Imbalance Check: The value counts for the target variable show a significant class imbalance: 46,328 instances with label -1 and only 3,672 instances with label 1.

4. Feature Selection by Missing Values

Columns are filtered based on the percentage of missing values to remove those with excessive sparsity.

Missing Value Calculation: The percentage of missing values for every column is calculated.

Filtering Criterion: Columns with less than 30% missing values are selected for the downstream analysis.

Result: This filtering step results in 67 columns being retained, creating a new DataFrame called `data_filtered_missing`.

5. Feature Engineering and Preprocessing

The selected features are prepared for model training by separating them by type and establishing a preprocessing pipeline.

5.1. Separating Feature Types

The 67 retained columns are separated into categorical (`object`, `category` dtypes) and numerical (`int64`, `float64` dtypes) feature lists.

- **Categorical features:** Variables like `Var192` through `Var228`.

- **Numerical features:** Variables like `Var6`, `Var7`, `Var13`, `Var21`, etc.

5.2. Creating Preprocessing Pipelines

- **Missing Indicator:** Flags columns with missing values.

- **Numerical Pipeline:**

    - Imputes missing values with the mean.

    - Applies `MinMaxScaler` for feature scaling.

- **Categorical Pipeline:**

    - Imputes missing values with the most frequent value.

    - Applies `labelEncoder` for categorical encoding.

- **Column Transformer:** Applies the numerical pipeline to numerical features and the categorical pipeline to categorical features.

6. Model Training and Evaluation (Baseline: Logistic Regression)

The first model, Logistic Regression, is trained as a baseline to establish initial performance metrics.

6.1. Data Splitting and Baseline Training

- Data split into training and testing sets (70/30 split) using `train_test_split`.
- Preliminary Pipeline combines preprocessing steps with a Logistic Regression model (`max_iter=5000`).
- Pipeline is fit on the training data (`X_train`, `y_train`).

6.2. Model Evaluation

- Generates probability predictions (`y_proba_lr`) on the test set.
- Classification threshold of 0.1 is applied to convert probabilities into binary predictions (`y_pred_lr`).
- Evaluated using a Classification Report and ROC-AUC score.

7. Handling Class Imbalance with SMOTE

To address the severe class imbalance identified in Step 3, the SMOTE technique is applied.

7.1. Applying Synthetic Minority Over-sampling Technique (SMOTE)

- SMOTE object is initialized and applied to the training data (`X_train`, `y_train`) to generate synthetic samples for the minority class (Label 1).
- Result is a new, balanced dataset (`X_resampled`, `y_resampled`).

7.2. Training and Evaluation with SMOTE

- Second Logistic Regression pipeline is trained using the SMOTE-resampled training data.
- Same threshold of 0.1 used for evaluation on the test set.
- Performance evaluated using Classification Report and ROC-AUC score, compared against the baseline model.

8. Advanced Modeling and Hyperparameter Tuning

An advanced gradient boosting model is introduced, and its hyperparameters are optimized.

8.1. Hyperparameter Search with RandomizedSearchCV

- Final pipeline constructed using the `ColumnTransformer` (preprocessing) and a CatBoost Classifier.
- Parameter grid (`param_grid_cat`) defined for tuning the CatBoost model.
- `RandomizedSearchCV` used to efficiently search the hyperparameter space (`n_iter=10`, `cv=3`).
- Search fitted on the original training data.

8.2. Model Evaluation

- Best estimator (`best_cat`) from randomized search is used to make probability predictions on the test set.
- Same threshold (0.1) applied to generate binary predictions (`y_pred_cat`).

- Best hyperparameters printed, performance presented via Classification Report and ROC-AUC score.

8.3. Precision vs. Threshold Analysis

- Precision-Recall Curve calculated using predicted probabilities and true test labels.
- Line plot visualizes the relationship between Precision and classification Threshold, allowing optimal threshold selection beyond the fixed 0.1 used in evaluations.
-

9. Results

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits


XGBoost Evaluation (Threshold=0.65):

Best XGB Params: {'subsample': 0.7, 'reg_lambda': 5, 'reg_alpha': 1,
'n_estimators': 500, 'max_depth': 4, 'learning_rate': 0.01, 'colsample_bytree':
0.8}

              precision    recall  f1-score   support


           0       0.94      0.92      0.93      9266

           1       0.20      0.24      0.22       734


    accuracy                           0.87     10000

   macro avg       0.57      0.58      0.57     10000

weighted avg       0.88      0.87      0.88     10000


ROC-AUC: 0.7048997212862823

Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Random Forest Evaluation (Threshold=0.65):

Best RF Params: {'n_estimators': 200, 'max_features': 'log2', 'max_depth': None, 'class_weight': 'balanced'}

	precision	recall	f1-score	support
0	0.93	1.00	0.96	9266
1	0.50	0.00	0.01	734
accuracy			0.93	10000
macro avg	0.71	0.50	0.48	10000
weighted avg	0.90	0.93	0.89	10000

ROC-AUC: 0.666311633577622

Fitting 3 folds for each of 10 candidates, totalling 30 fits

c:\users\rajud\anaconda3\lib\site-packages\catboost\core.py:1411: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.

self._init_pool(data, label, cat_features, text_features, embedding_features, embedding_features_data, pairs, weight,

CatBoost Evaluation (Threshold=0.65):

Best CatBoost Params: {'learning_rate': 0.01, 'l2_leaf_reg': 7, 'iterations': 600, 'depth': 5}

	precision	recall	f1-score	support
0	0.94	0.93	0.93	9266
1	0.20	0.23	0.21	734

accuracy			0.88	10000
macro avg	0.57	0.58	0.57	10000
weighted avg	0.88	0.88	0.88	10000

ROC-AUC: 0.7024428178139175