Step 2: Advanced Modeling & MLflow Experiment Tracking

University: Tek-Up
Group: ING-4-SDIA

Tutor: Haythem Ghazouani

1 Introduction

Building a single model is rarely sufficient. In this tutorial, we explore the **Ensemble Learning** strategies discussed in the course notes (`Ensemble-Learning.pdf`). We will compare simple models with advanced Voting, Stacking, and Boosting.

Implementation Note

The complete implementation of these strategies can be found in `code/modeling.py`.

2 Modeling Strategies

Strategy: 1. Simple Baselines

Start with classical models like **Logistic Regression** or **Decision Trees**. These provide a performance baseline. If an advanced ensemble doesn't beat these, the complexity is not justified.

Strategy: 2. Voting Classifier

Combines multiple models (e.g., Random Forest + XGBoost) and averages their predictions. **Soft Voting** uses probabilities, which often yields better results.

Strategy: 3. Stacking Classifier

Trains a set of base-models and uses their predictions as features for a **Meta-Learner** (usually Logistic Regression). This allows the model to learn which base-model is most reliable for different segments of data.

Strategy: 4. Boosting (XGBoost)

Sequential learning where each new model fixes the errors of the previous ones. This is the industry standard for high-performance tabular ML.

3 Optimization Handling Imbalance

Strategy: Class Imbalance: SMOTE

Our dataset is imbalanced (80% vs 20%). We use **SMOTE** (Synthetic Minority Over-sampling Technique) to generate synthetic examples for the minority class during training.

Implementation Note

Important: SMOTE must only be applied to the training set, never the test set. This is why we use `imblearn.pipeline` instead of the standard sklearn pipeline.

Strategy: Hyperparameter Tuning: GridSearchCV

Instead of manual values, we define a grid of parameters (e.g., `n_estimators`: [100, 200]) and let `GridSearchCV` find the optimal combination using Cross-Validation.

4 Tracking with MLflow

Every strategy mentioned above must be tracked as a separate "Run" in **MLflow**.

```
import mlflow
from sklearn.ensemble import VotingClassifier

mlflow.set_experiment("Bank\_Churn\_Ensembles")

# Define the ensemble
voting_model = VotingClassifier(estimators=[('rf', rf), ('xgb',
    xgb)], voting='soft')

with mlflow.start_run(run_name="Voting\_RF\_XGB"):
    pipeline.fit(X_train, y_train)
    mlflow.log_metric("accuracy", accuracy_score(y_test, y_pred))
    mlflow.sklearn.log_model(pipeline, "model")
```

5 Selecting the Best Model

After running your script, launch the UI:

```
mlflow ui
```

Sort your experiments by **F1-Score** or **Accuracy**. The model with the highest performance while maintaining low complexity will be selected for Step 3: Deployment.

Experiment Data

All experiment metadata, parameters, and logged models are stored in the local `mlruns/` directory. Do not delete this folder if you want to keep your history.

6 Exercise

1. Implement a **StackingClassifier** using at least three different base-models.
2. Use MLflow to identify which model handles the minority class (Churn=1) the best.
3. **Question:** Why might a Stacking model overfit more easily than a simple Voting model?