Term: Fall 2025 **Subject:** Industrial Engineering (IEE) **Number:** 520

Course Title: Statistical **L**earning for **D**ata **M**ining (CSE 512)

Final Project Report

Project Title:
Binary Classification with XGBoost

Author:
Abhi Sachdeva (1221508080)

Date:
December 4, 2025

Introduction

This report presents the development and evaluation of a binary classification model. After thoroughly testing multiple classification algorithms, such as Random Forest, AdaBoost, and XGBoost, the final selected model is an XGBoost Classifier optimized using a randomized hyperparameter search. The model achieves a Balanced Error Rate (BER) of 0.3405 on a 5-fold cross-validation. This report outlines the following steps: data preparation/preprocessing, methodology, evaluation, and final model selection.

1. Data Preparation

1.1 Data Overview

There are two datasets: ProjectLABELED2025.csv, containing the labeled training data, and ProjectNOTLABELED2025.csv, containing the unlabeled data for prediction. The target variable is a binary class label. Additionally, the datasets consist of 21 feature attributes organized into three categories:

1. **Numerical Features (7 attributes):**
   $x15,\ x16,\ x17,\ x18,\ x19,\ x20,\ x21$

2. **Categorical Features (3 attributes):**
   $x2,\ x3,\ x4$

3. **Binary Features (11 attributes):**
   $x1,\ x5,\ x6,\ x7,\ x8,\ x9,\ x10,\ x11,\ x12,\ x13,\ x14$

1.2 Missing Values

Dropping rows with missing values can significantly reduce the training data; thus, we adopt an imputation strategy to avoid dropping any rows. Imputation statistics were computed exclusively from the labeled training data and then applied consistently to both the training data and the unlabeled datasets. This prevents any data leakage from the test set.

1. **Numerical Features:** Missing values are imputed using the median value computed from the labeled training data.

2. **Binary and Categorical Features:** Missing values are imputed using the mode (most frequent value) from the labeled training data

1.3 Class Imbalancing

The dataset shows an imbalance between the two target classes. To address the imbalance, we compute the scaled weight parameter, which is the ratio of the majority class count to the minority class count. This parameter is passed to the XGBoost Classifier, which uses it to adjust the gradient computations during training. This effectively gives higher importance to the minority class samples without artificially modifying the training data distribution.

1.4 Additional Feature Engineering

Minimal Feature Engineering was applied to the final selected model. Tree-based models, such as XGBoost, can effectively handle categorical features natively. Thus, one-hot-encoding was not utilized. I explored other methods, such as polynomial features and interaction features, but neither significantly improved the model's BER. Pseudo-labeling with an imbalanced class allowed more confirmation bias, thus causing no significant improvement in BER.

2. Methods

I tested a variety of classification models such as Random Forest, AdaBoost, and XGBoost. For each, I aimed to find the optimal hyperparameters using either RandomizedSearchCV or GridSearchCV from the scikit-learn library for 100 iterations. For the final selected model (XGBoost), I used RandomizedSearchCV due to the high dimensionality of the parameter space (9 hyperparameters) for 100 iterations, sampling from the following parameters:

Parameter	Distribution	Description
n_estimators	$randint(100,\ 350)$	Number of boosting rounds
max_depth	$randint(3,\ 10)$	Maximum tree depth
learning_rate	$Uniform(0.02,\ 0.18)$	Step size
subsample	$Uniform(0.7,\ 0.3)$	Row subsampling ratio
colsample_bytree	$Uniform(0.7,\ 0.3)$	Column subsampling ratio
min_child_weight	$randint(1,\ 8)$	Minimum sum of instance weight
gamma	$Uniform(0,\ 0.3)$	Minimum loss reduction for split
reg_alpha	$Uniform(0,\ 1)$	L_1 regularization
reg_lambda	$Uniform(0.5,\ 1)$	L_2 regularization

2.1 Model's Explored

1. **Random Forest Classifier:** An ensemble method using bagging with decision trees.

2. **AdaBoost Classifier:** An ensemble method using boosting to sequentially train weak learners. Tested with a varying number of estimators

3. **XGBoost Classifier (Selected Model):** An ensemble method using an optimized gradient boosting algorithm with regularization.

3. Evaluate

3.1 Cross-Validation

The current implementation utilizes a 5-fold cross-validation strategy, which partitions the training data into 5 equal folds while maintaining the class distribution in each fold. This is crucial for imbalanced datasets, such as ours, as it ensures each fold has representative samples from both classes. Additionally, I employed a stratified approach to ensure that each fold does not contain predominantly one class.

3.2 Performance Metric

The evaluation matric is Balanced Error Rate (BER): the average of the per-class error rates

$$BER = \frac{False\ Positive\ Rate + False\ Negative\ Rate}{2} \equiv 1 - balanced\ accuracy$$

A model predicting the majority class always might have high accuracy but fail completely on the minority class; BER accounts for both. It gives equal weighting to errors on all classes, preventing a good performance on a large class from masking poor performance masking poor performance on a smaller class.

3.3 Implementation

To ensure that we optimize for our target metric (BER), we have to maximize our balanced accuracy. During the hyperparameter tuning, I used balanced_accuracy_score ($1 - BER$) as the scoring function for RandomizedSearchCV. The process of velauating the BER is as follows:

1. Cross_val_predict to generate predictions for each sample when it is in the test fold

2. Construct a confusion matrix from these predictions

3. Compute the per-class error rates from the confusion matrix

4. Calculate the BER - the arithmetic mean of the per-class error rates

4. Selected Model

4.1 Model Specification

The final model selected is an XGBoost:

- **Algorithm:** XGBClassifier from the xgboost library
- **Objective:** Binary Classification (Predicting class labels with logistic loss)
- **Evaluation Metric:** logloss
- **Class Weighting:** scale_pos_weight (ratio of class counts)

The optimal hyperparameters found via the RandomizedSearchCV are:

Parameter	Value
n_estimators	246
max_depth	3
learning_rate	0.05
subsample	0.9661
colsample_bytree	0.8675
min_child_weight	3
gamma	0.2648
reg_alpha	0.7260
reg_lambda	1.3971

4.2 Results

The model achieved the following results on a 5-fold stratified cross validation:

- **Balanced Error Rate (BER):** 0.3405
- **Balanced Accuracy:** 0.6595
- **Class 0 Error Rate:** 0.2770
- **Class 1 Error Rate:** 0.4041

The BER of 0.3405 means that the model has approximately a 66% accuracy in predicting labels. Although this is not very accurate, other models that I hae explored have yielded very similar results ($\pm\ 0.01\ -\ 0.02$). Moreover, our BER score suggest that our model is learning some patterns as it is better than random guessing ($BER\ =\ 0.50$).

4.3 Selection Criteria

XGBoost was selected as the final model for the following reasons:

1. **Consistent Performance:** Across multiple experiments, XGBoost consistently provided the best BER score while also running quicker than other models (such as Random Forest)

2. **Imbalance Handling:** The scale_pos_weight parameter provided a native way to handle class balancing without data manipulation (i.e., dropping values)

3. **Regularization:** XGBoost offered built in regularization (reg_alpha and reg_lambda) to prevent overfitting

4. **Feature Handling:** Tree-based models (i.e., Random Forest or XGBoost) natively handle mixed feature types (i.e., binary, categorical, numerical)

5. Conclusion

5.1 Libraries Utilized (Python 3.11.5)

- pandas
- numpy
- scikit-learn
- xgboost
- scipy

5.2 Random Seeds

All randomized operation use a fixed random state (42) for reproducible results. This includes the RandomizedSearchCV, StratifiedKFold, and the XGBClassifier.

5.3 Code

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV,
cross_val_predict
from sklearn.metrics import balanced_accuracy_score, make_scorer, confusion_matrix
from scipy.stats import randint, uniform
import warnings

warnings.filterwarnings('ignore')
os.environ['PYTHONWARNINGS'] = 'ignore'
os.environ['XGB_VERBOSITY'] = '0'
import xgboost as xgb

# Configuration
labeled_path = 'ProjectLABELED2025.csv'
unlabeled_path = 'ProjectNotLABELED2025.csv'
output_path = 'ProjectPredictions2025AbhiSachdeva.csv'
target_column = 'label'

# Data Overview
binary_columns = ['x1', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13',
'x14']
categorical_columns = ['x2', 'x3', 'x4']
numerical_columns = ['x15', 'x16', 'x17', 'x18', 'x19', 'x20', 'x21']
```

```
columns = binary_columns + categorical_columns + numerical_columns

# Settings
N = 100
cv_folds = 5
rand_state = 42

def loadData():

   # Load datasets
   labeled = pd.read_csv(labeled_path)
   unlabeled = pd.read_csv(unlabeled_path)

   # Find the index column
   index_column = unlabeled.columns[0]
   if index_column not in columns + [target_column]:
       unlabeled_indices = unlabeled[index_column].values
   else:
       unlabeled_indices = np.arange(len(unlabeled))

   feature_columns = [col for col in columns if col in labeled.columns]
   x_labeled = labeled[feature_columns].copy()
   y_labeled = labeled[target_column].copy()
   x_unlabeled = unlabeled[feature_columns].copy()

   # Imputate using median value
   for col in numerical_columns:
       if col in x_labeled.columns:
           median = x_labeled[col].median()
           x_labeled[col].fillna(median, inplace=True)
           x_unlabeled[col].fillna(median, inplace=True)

   # Imputate using mode value
   for col in binary_columns + categorical_columns:
       if col in x_labeled.columns:
           mode = x_labeled[col].mode()[0]
           x_labeled[col].fillna(mode, inplace=True)
           x_unlabeled[col].fillna(mode, inplace=True)

   class_counts = y_labeled.value_counts()
   scale_pos_weight = class_counts[0] / class_counts[1]
   return x_labeled, y_labeled, x_unlabeled, unlabeled_indices, scale_pos_weight
```

```
def computeHyperparameters(x_labeled, y_labeled, scale_pos_weight):

    # Hyperparameter grid
    parameter_grid = {
        'n_estimators': randint(100, 350),
        'max_depth': randint(3, 10),
        'learning_rate': uniform(0.02, 0.18),
        'subsample': uniform(0.7, 0.3),
        'colsample_bytree': uniform(0.7, 0.3),
        'min_child_weight': randint(1, 8),
        'gamma': uniform(0, 0.3),
        'reg_alpha': uniform(0, 1),
        'reg_lambda': uniform(0.5, 1),
    }

    # XGBoost classifier model
    base_model = xgb.XGBClassifier(
        scale_pos_weight = scale_pos_weight,
        objective='binary:logistic',
        eval_metric='logloss',
        verbosity=0,
        random_state=rand_state,
        n_jobs=-1
    )

    # 5 fold stratified cross validation
    cv = StratifiedKFold(n_splits=cv_folds, shuffle=True, random_state=rand_state)
    scorer = make_scorer(balanced_accuracy_score)

    # Find optimal hyperparameters
    search = RandomizedSearchCV(
        estimator=base_model,
        param_distributions=parameter_grid,
        n_iter=N,
        scoring=scorer,
        cv=cv,
        random_state=rand_state,
        n_jobs=-1,
        verbose=0
    )
```

```
    search.fit(x_labeled, y_labeled)
    return search.best_estimator_, search.best_params_

def evaluateModel(model, x_labeled, y_labeled):
    cv = StratifiedKFold(n_splits=cv_folds, shuffle=True, random_state=rand_state)
    y_predicted_cv = cross_val_predict(model, x_labeled, y_labeled, cv=cv,
n_jobs=-1)

    # Generate a confusion matrix
    matrix = confusion_matrix(y_labeled, y_predicted_cv)

    classes = sorted(y_labeled.unique())
    error_rates = []

    # Calculate the error rates and the BER
    for cls in classes:
        mask = (y_labeled == cls)
        total = mask.sum()
        correct = ((y_labeled == cls) & (y_predicted_cv == cls)).sum()
        error_rate = (total - correct) / total
        error_rates.append(error_rate)

    ber = np.mean(error_rates)
    return ber, matrix, error_rates

def makePredictions(model, x_labeled, y_labeled, x_unlabeled, indices):

    # Make predictions on the unlabeled data
    model.fit(x_labeled, y_labeled)
    predictions = model.predict(x_unlabeled)

    # Generate a report with the predictions and the corresponding indices
    report = pd.DataFrame({
        'index': indices,
        'label': predictions.astype(int)
    })

    report.to_csv(output_path, index=False, header=False)
    return predictions

def main():
    x_labeled, y_labeled, x_unlabeled, indices, scale_pos_weight = loadData()
```

```
    model, parameters = computeHyperparameters(x_labeled, y_labeled,
scale_pos_weight)
    ber, matrix, error_rates = evaluateModel(model, x_labeled, y_labeled)
    predictions = makePredictions(model, x_labeled, y_labeled, x_unlabeled, indices)

    print("RESULTS: ")

    # Confusion Matrix
    print("\nConfusion Matrix:")
    print(matrix)
    print(f"\n  Class 0 error rate: {error_rates[0]:.4f}")
    print(f"  Class 1 error rate: {error_rates[1]:.4f}")

    # BER
    print(f"\n*** Balanced Error Rate (BER): {ber:.4f} ***")
    print(f"*** Balanced Accuracy: {1-ber:.4f} ***")

    # Selected Model and Hyperparameters
    print()
    print("MODEL: XGBoost")
    print("\nOptimal Hyperparameters:")
    for param, value in sorted(parameters.items()):
        if isinstance(value, float):
            print(f"  {param}: {value:.4f}")
        else:
            print(f"  {param}: {value}")

    return model, ber

if __name__ == "__main__":
    model, ber = main()
```

5.4 Conclusion

In conclusion, this project developed a binary classification model to predict class labels on an unlabeled dataset. By training the our selected XGBoost model with the pre-proccessed labeled training data and optimal hyperparameters, the model was able to achieve a BER of 0.3405 (~ 66% accuracy).