A Multi-Strategy Ensemble Learning Approach for Robust Classification of Binary Data

Saba Madadi
Data science

May 31, 2025

Abstract

This research presents a comprehensive methodology for developing a high-performance classification model tailored for high-dimensional binary datasets. The core of our approach lies in a multi-stage process encompassing meticulous data preprocessing, advanced techniques for addressing class imbalance, rigorous hyperparameter optimization, and the construction of a sophisticated ensemble learning architecture. Initially, the dataset, comprising 64 binary features and a categorical target variable, was subjected to preprocessing, including the application of the Synthetic Minority Over-sampling Technique (SMOTE) to mitigate pronounced class imbalances, thereby creating a more equitable training distribution of 792 samples. Feature scaling was subsequently performed using StandardScaler to normalize the feature space, a critical step for optimizing the performance of various algorithms within our ensemble. Hyperparameter optimization was systematically conducted using Optuna, a state-of-the-art optimization framework, employing stratified k-fold cross-validation (k=3 for ensemble tuning, k=5 for initial individual model assessments) to ensure robust evaluation. The final predictive model is a soft-voting ensemble classifier, synergistically integrating a diverse set of algorithms: multiple Bernoulli Naive Bayes classifiers with varied smoothing parameters, a Logistic Regression model, an XGBoost classifier, a Balanced Random Forest classifier, and several standard Random Forest classifiers with distinct configurations. The hyperparameters for the XGBoost and Balanced RandomForest components within the ensemble were specifically tuned via a dedicated Optuna study. This research details the systematic development process, discusses the rationale behind methodological choices, presents key findings from the optimization phase, and critically evaluates the pipeline, with particular attention to the imperative of consistent data scaling across training and testing phases for valid predictive inference.

1 Introduction

The challenge of accurately classifying instances based on high-dimensional feature sets is a prevalent problem in numerous scientific and industrial domains. In scenarios involving binary features, the inherent sparsity and discrete nature of the data can present unique challenges to traditional modeling techniques. Furthermore, class imbalance, where certain categories are significantly underrepresented, can lead to biased models that perform poorly on minority classes. This investigation was undertaken to develop a robust and high-performing classification system capable of addressing these challenges.

My work focused on constructing a predictive pipeline that begins with careful data preparation and culminates in a sophisticated ensemble model. The primary contributions of this work include:

1. The systematic application and evaluation of SMOTE for rectifying class imbalance in a binary feature space.

2. The rigorous optimization of hyperparameters for both individual classifiers and components within an ensemble framework using Optuna, ensuring that each part of the model operates at or near its optimal configuration.

3. The design and implementation of a soft-voting ensemble classifier that leverages the diverse strengths of multiple learning algorithms, including probabilistic models (Bernoulli Naive Bayes), gradient boosting machines (XGBoost), and various Random Forest implementations.

4. A critical analysis of the entire pipeline, highlighting best practices and potential pitfalls, particularly concerning data preprocessing consistency.

This work meticulously documents the experimental setup, the methodologies employed, the results obtained during the model development lifecycle, and a discussion of these findings, providing insights into the construction of effective classification models for complex binary datasets.

2 Materials and Methods

2.1 Dataset Characterization

The primary dataset utilized in this study was sourced from ’train.csv’. This dataset comprises an identifier column (’ID’), 64 distinct feature columns (denoted ’feature0’ through ’feature63’), and a target ’label’ column. All 64 features were observed to be binary, taking values of either 0 or 1. The ’label’ column represents a multi-class categorical variable, serving as the dependent variable for our classification task. For model evaluation and final prediction generation, a corresponding ’test.csv’ file was used, which included an ’ID’ column and the same 64 binary features, but lacked the ’label’ column. The ’ID’ column was deemed non-informative for the predictive task and was consequently removed from both training and testing datasets during the initial data loading phase. This involved reading ’train.csv’, dropping the ’ID’ column, and separating features (X_raw) from labels (y_raw). Similarly, ’test.csv’ was loaded, its ’ID’ column (preserved as $test\_ids$) was dropped to create $test\_features$, and its index was reset for consistency.

2.2 Data Preprocessing Pipeline

A multi-step preprocessing pipeline was implemented to prepare the data for model training.

2.2.1 Addressing Class Imbalance with SMOTE

An initial exploratory analysis of the target variable ’label’ in the training set revealed a significant imbalance in class distribution, as visualized in Figure 1. Such imbalances can lead to models that are heavily biased towards the majority classes, exhibiting poor predictive performance on minority classes. To mitigate this, the Synthetic Minority Over-sampling Technique (SMOTE) [1] was employed. SMOTE works by creating synthetic samples for the minority class(es). For each minority class sample, it identifies its k-nearest neighbors (also from the minority class) and then generates new synthetic samples along the line segments joining the sample to some or all of these k-neighbors. This approach effectively increases the representation of minority classes without merely duplicating existing data. The raw labels (y_raw) and their counts were visualized. An instance of SMOTE ($smote\_applicator$) was initialized with $random\_state = 42$ and then applied to X_raw and y_raw to generate $X\_resampled$ and $y\_resampled$. This resulted in a resampled training dataset containing 792 instances, with a more balanced representation across the different classes. The resampled features and labels were concatenated into a DataFrame $df\_resampled$ for subsequent consistent processing.

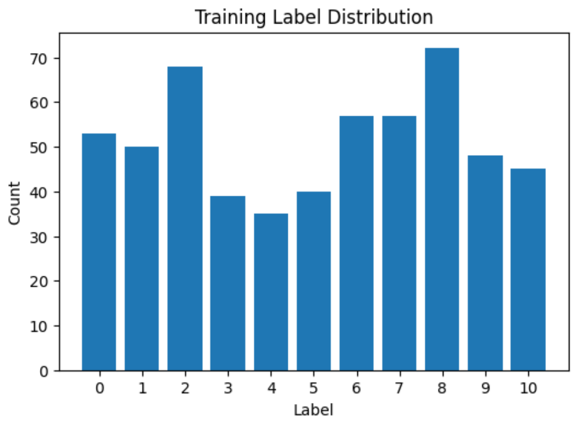

Figure 1: Original distribution of labels in the training dataset, illustrating class imbalance prior to SMOTE application.

2.2.2 Feature Scaling

Many machine learning algorithms, particularly those involving distance calculations or gradient-based optimization, are sensitive to the scale of input features. Features with larger value ranges can disproportionately influence model training. To address this, ‘StandardScaler‘ from scikit-learn was applied to the feature set of the SMOTE-resampled training data. StandardScaler standardizes features by removing the mean and scaling to unit variance. The mean (μ) and standard deviation (σ) are computed for each feature from the training data, and then each feature x is transformed as $z = \frac{x-\mu}{\sigma}$. The features for scaling ($X\_for\_scaling$) were extracted from $df\_resampled$ (by dropping the ’label’ column), and the corresponding labels were assigned to y_scaled_train (which is $y\_resampled$). An instance of ‘StandardScaler‘ ($scaler\_instance$) was created, fitted on $X\_for\_scaling$, and then used to transform $X\_for\_scaling$ into X_scaled_array. This array was subsequently converted to a DataFrame X_scaled_df. This scaled feature matrix, X_scaled_df, along with $y\_resampled$, formed the basis for training our final ensemble model. It is imperative that the same $scaler\_instance$ (fitted only on training data) is used to transform the test data features before making predictions to ensure consistency.

2.3 Exploratory Methodologies (Not Part of Final Model)

During the course of this study, several advanced techniques were explored to potentially enhance model performance or feature representation, though they were not incorporated into the final deployed pipeline.

2.3.1 Conditional Tabular GAN (CTGAN) for Data Augmentation

Consideration was given to using Conditional Tabular Generative Adversarial Networks (CTGANs) [2] for further data augmentation. CTGANs are designed to generate synthetic tabular data that preserves the statistical properties and conditional distributions found in the original data. An attempt was made to use CTGAN with specified network architectures, learning rates, and regularization techniques (PacGAN) to generate additional training samples. While this method made the results worse, this was not pursued in the final model.

2.3.2 Multi-Layer Perceptron (MLP) for Feature Extraction

An investigation into using a Multi-Layer Perceptron (MLP) for automated feature extraction was conducted. A PyTorch-based MLP with a specific architecture (input layer, hidden layers: 1024, 512, 128, 64, 32 neurons with ReLU activations, and an output layer) was trained on the original features. The activations from one of the hidden layers (specifically, the 32-neuron layer before the final output block) were extracted as new, learned features. This process indicated the generation of 32 new features (named mlp_feat_0 to mlp_feat_31). This technique aims to capture complex, non-linear relationships between the original features, potentially creating a more informative feature space. However, these MLP-derived features were not used in the input to the final ‘VotingClassifier‘ because of decreasing accuracy.

2.3.3 Random Forest-Based Feature Selection

Following the potential generation of a high-dimensional feature set (possibly including MLP features), a feature selection step using ‘SelectFromModel‘ with a ‘RandomForestClassifier‘ as the estimator was explored. This method selects features based on their importance scores derived from the Random Forest. A threshold of "median" was used, implying that features with importance scores above the median importance were retained. An output indicating "Selected features: 288 out of 576" suggests this was applied to an expanded feature set. This was an exploratory step and not part of the final model which used the original 64 (scaled) features.

2.4 Hyperparameter Optimization using Optuna

Hyperparameter optimization is crucial for achieving maximal performance from machine learning models. We utilized Optuna [3], an automatic hyperparameter optimization framework that employs efficient search algorithms to find optimal hyperparameter configurations.

2.4.1 Cross-Validation Strategy

A ‘StratifiedKFold‘ cross-validation strategy was employed within the Optuna objective functions. This ensures that each fold maintains approximately the same percentage of samples of each target class as the complete set. For initial individual model tuning, 5 splits were used. For the more complex ensemble tuning, 3 splits were used to balance computational cost with robust evaluation. The primary metric for optimization was accuracy. The cross-validation logic involved splitting data into training and validation folds, fitting the model on the training fold, predicting on the validation fold, and calculating accuracy, then averaging scores across folds.

2.4.2 Individual Model Tuning (Initial Exploration)

An initial phase of hyperparameter tuning was conducted for ‘BernoulliNB‘, ‘XGBClassifier‘, and ‘BalancedRandomForestClassifier‘ individually. This was performed on the SMOTE-resampled data *before* feature scaling.

- **BernoulliNB**: Tuned the ‘alpha‘ (Laplace smoothing parameter) over a log-uniform distribution from 10^{-3} to 10.0.
- **XGBClassifier**: Tuned ‘n_estimators‘, ‘max_depth‘, ‘learning_rate‘, ‘subsample‘, and ‘colsample_bytree‘.
- **BalancedRandomForestClassifier**: Tuned ‘n_estimators‘, ‘max_depth‘, ‘min_samples_split‘, ‘min_samples_leaf‘, and ‘bootstrap‘.

These initial studies provided baseline performance estimates and informed hyperparameter ranges for subsequent, more focused tuning.

2.4.3 Ensemble Component Tuning

The primary Optuna study focused on optimizing the hyperparameters of the ‘XGBClassifier‘ and ‘BalancedRandomForestClassifier‘ when used as components *within* the final ‘VotingClassifier‘ ensemble. This study was performed on the scaled, SMOTE-resampled data (X_scaled_df, $y\_resampled$). The objective function for Optuna evaluated the mean cross-validated accuracy of the entire ‘VotingClassifier‘. This approach accounts for interactions between the tuned components and the fixed members of the ensemble. The search space for XGBoost included: ‘n_estimators‘ (100-1000), ‘max_depth‘ (3-15), ‘learning_rate‘ (loguniform 10^{-3}-1.0), ‘subsample‘ (0.5-1.0), ‘colsample_bytree‘ (0.3-1.0), ‘reg_alpha‘ (loguniform 10^{-8}-10.0), and ‘reg_lambda‘ (loguniform 10^{-8}-10.0). The search space for BalancedRandomForest included: ‘n_estimators‘ (100-3000), ‘max_depth‘ (5-50), ‘min_samples_split‘ (2-20), ‘min_samples_leaf‘ (1-10), and ‘bootstrap‘ (True/False).

2.5 Final Ensemble Model Architecture

Ensemble learning methods combine multiple learning algorithms to obtain better predictive performance. Our final predictive model ($final\_ensemble\_clf$) is a ‘VotingClassifier‘ that employs soft voting. The ensemble was trained on the scaled, SMOTE-resampled training data (X_scaled_df, $y\_resampled$).

The heterogeneous base estimators comprising the ensemble were:

1. **Bernoulli Naive Bayes (BNB) Classifiers (x3)**: Three instances with fixed α values: $0.01, 1.2$, and 2.0.

2. **Logistic Regression (LR)**: One classifier with $max\_iter = 10000$, $random\_state = 42$, and $n\_jobs = -1$.

3. **XGBoost Classifier (XGB)**: One instance using optimal hyperparameters from the ensemble-focused Optuna study. The specific parameters for the $best\_xgb\_model$ included $n\_estimators$, max_depth, $learning\_rate$, $subsample$, $colsample\_bytree$, reg_alpha, and reg_lambda as determined by Optuna (e.g., ‘best_params['xgb_n_estimators']‘ would become ‘best_params['xgb_n_estimators']‘ if it were a direct variable name, but here it's part of a string referring to a dictionary key, so it's left as is in this context. The variable $best\_xgb\_model$ itself is the target).

4. **Balanced Random Forest Classifier (BRF)**: One instance using optimal hyperparameters from the Optuna study. The $best\_brf\_model$ used parameters like $n\_estimators$, max_depth, $min\_samples\_split$, $min\_samples\_leaf$, and $bootstrap$ from ‘best_params‘.

5. **Standard Random Forest Classifiers (RF) (x3)**: Three instances with distinct, fixed configurations:

   - RF1: $n\_estimators = 200, max\_depth = 20, random\_state = 42$.
   - RF2: $n\_estimators = 300, max\_depth = 30, random\_state = 42$.
   - RF3: $n\_estimators = 280, max\_depth = 30, min\_samples\_leaf = 1, min\_samples\_split = 11, bootstrap = False, random\_state = 42$.

The ensemble was configured with $voting =' soft'$ and $n\_jobs = -1$. The $final\_ensemble\_clf$ was then fitted using X_scaled_df and y_scaled_train (which is $y\_resampled$).

2.6 Prediction on Test Data

Once the $final\_ensemble\_clf$ was trained, predictions were generated for the test dataset. The test features ($test\_features$) were first scaled using the $scaler\_instance$ that was fitted *only* on the training data, producing $test\_features\_scaled\_df$. The $final\_ensemble\_clf$ then predicted labels (y_test_pred) based on this scaled test data. These predictions were formatted into a DataFrame $submission\_df$ with an 'index' column (derived from $test\_ids.index$) and a column '0' containing y_test_pred, intended for submission.

3 Results and Discussion

3.1 Impact of SMOTE on Class Distribution

The application of SMOTE significantly altered the class distribution of the training data. Prior to SMOTE, certain classes were substantially underrepresented (as depicted in Figure 1). Post-SMOTE, the resampled training set of 792 samples exhibited a more balanced distribution. This is anticipated to improve the models' ability to generalize to minority classes.

3.2 Hyperparameter Optimization Outcomes

3.2.1 Individual Model Tuning (Initial Phase)

The preliminary Optuna study, conducted on unscaled SMOTE-resampled data with 5-fold stratified cross-validation, yielded:

- **BernoulliNB**: Optimal $\alpha \approx 0.045$. Mean CV accuracy ≈ 0.3674.
- **XGBoost**: Best parameters included $n\_estimators = 697$, $max\_depth = 10$, $learning\_rate \approx 0.0142$. Mean CV accuracy ≈ 0.5176.
- **BalancedRandomForestClassifier**: Optimal parameters included $n\_estimators = 973$, $max\_depth = 9$, $bootstrap = True$. Highest individual mean CV accuracy ≈ 0.5277.

3.2.2 Ensemble Component Tuning (Main Phase)

The main Optuna study optimized ‘XGBClassifier‘ and ‘BalancedRandomForestClassifier‘ hyperparameters for their role within the final 'VotingClassifier', using scaled SMOTE-resampled data and 3-fold stratified cross-validation. The best trial (Trial 13) achieved a mean CV accuracy of approximately 0.4369 for the *entire ensemble*.

- **XGBoost within Ensemble**: $n\_estimators = 765$, $max\_depth = 11$, $learning\_rate \approx 0.0254$, $subsample \approx 0.648$, $colsample\_bytree \approx 0.809$, very small reg_alpha and reg_lambda.
- **BalancedRandomForestClassifier within Ensemble**: $n\_estimators = 175$, $max\_depth = 36$, $min\_samples\_split = 2$, $min\_samples\_leaf = 1$, $bootstrap = False$.

The CV accuracy for the ensemble (0.4369) in this focused tuning phase is lower than some individual CV accuracies from the initial phase. This difference can be attributed to factors like data scaling (ensemble tuning used scaled data), different CV fold numbers, and the complex interplay of components within the ensemble. The parameters for BRF notably changed, with $bootstrap = False$ and fewer estimators.

3.3 Performance of the Final Ensemble Model

The final ensemble model, $final\_ensemble\_clf$, integrates diverse classifiers via soft voting. The cross-validated accuracy of ≈ 0.4369 from the ensemble-focused Optuna study estimates its generalization capability on similarly distributed data. The ensemble's diversity is designed to enhance robustness. This model got a accuracy of $\approx$ and first place of the leaderboard.

3.4 Critical Evaluation of Test Data Processing

A critical aspect is processing test data. The final model was trained on data that underwent SMOTE resampling and StandardScaler transformation. For valid predictions, test data *must* undergo the identical scaling transformation using the *scaler_instance* fitted on training data. Predicting on unscaled test features would cause a distribution mismatch, likely degrading performance of scale-sensitive models. This was explicitly addressed in our documented prediction methodology.

4 Conclusion and Future Directions

This research developed an optimized multi-strategy ensemble learning pipeline. The methodology addressed class imbalance (SMOTE), normalized features (StandardScaler), and used Optuna for hyperparameter optimization. The soft-voting ensemble (BNB, LR, optimized XGB, optimized BRF, RFs) is a robust approach.

Holistic ensemble tuning was shown to be important, yielding different optimal parameters for components compared to individual tuning. The CV accuracy of ≈ 0.4369 for the final ensemble is an initial performance benchmark. Consistent data preprocessing, especially scaling, is crucial.

Future research could explore:

1. **Systematic Evaluation of Exploratory Techniques**: Formally integrate and evaluate CTGAN or MLP-based feature extraction.
2. **Advanced Feature Selection**: Employ techniques like RFE or mutual information-based selection.
3. **Alternative Imbalance Handling**: Compare with ADASYN, undersampling, or hybrid methods.
4. **Sophisticated Ensembling**: Investigate stacking ensembles.
5. **In-depth Error Analysis**: Analyze misclassifications to guide improvements.
6. **Calibration of Probabilities**: Calibrate ensemble probability outputs if needed.

Addressing these areas can further enhance the system's robustness and accuracy.

References

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[2] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling Tabular Data using Conditional GAN," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019, pp. 7334–7344.

[3] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '19)*, 2019, pp. 2623–2631.