**Credit Risk Classification Pipeline**

**Detailed Technical Documentation**

**1. Business Problem**

The project aims to predict credit risk by classifying accounts as good (0) or bad (1) based on various account features. This binary classification problem focuses on identifying high-risk accounts while minimizing false positives.

**Key Business Metrics to Track:\_**

* Bad Flag Prediction Accuracy: 99%
* False Positive Rate: 60%

**2. Data Preprocessing Pipeline**

**2.1 Initial Data Cleaning**

* Original Dataset Dimensions: 96806 rows × 1216 columns
* Missing Value Threshold: 80%
* Columns Removed Due to Missing Values: ['bureau\_148', 'bureau\_436', 'bureau\_438', 'bureau\_444', 'bureau\_446', 'bureau\_447', 'bureau\_448', 'bureau\_449', 'onus\_attribute\_43', 'onus\_attribute\_44', 'onus\_attribute\_45', 'onus\_attribute\_46', 'onus\_attribute\_47', 'onus\_attribute\_48']
* Final Dataset Dimensions: 96806 rows × 1202 columns

**2.2 Missing Value Treatment**

Implementation Method: SimpleImputer

Strategy: Mean Imputation

Features Imputed: All of the remaining features with NaN values.

```

**2.3 Class Imbalance Details**

* Positive Class (Bad Flag = 1) Count: 1372
* Negative Class (Bad Flag = 0) Count: 95434
* Initial Imbalance Ratio: 69.55:1

**3. Feature Engineering & Selection**

**3.1 Random Forest Feature Selection**

* Initial Feature Count: 236
* Feature Importance Calculation Method: RandomForestClassifier
* Top Features Selected: \_\_\_\_\_\_\_
* Feature Selection Criteria: Importance threshold > \_\_\_\_\_\_\_

**3.2 Key Features Identified (Top 10):**

1. \_\_\_\_\_\_\_ (Importance Score: \_\_\_\_\_\_\_)

2. \_\_\_\_\_\_\_ (Importance Score: \_\_\_\_\_\_\_)

3. \_\_\_\_\_\_\_ (Importance Score: \_\_\_\_\_\_\_)

[Continue list...]

**4. Model Development**

**4.1 Random Search Optimization**

* Number of Trials: 50
* Parameter Ranges: RF Ratio: [1, 40], XGB Ratio: [1, 40], Feature Count: [50, 500]

**Best Configuration Found:**

- RF Ratio: 8.3092

- XGB Ratio: 33.0699

- Feature Count: 236

- Resulting Accuracy: 97%

**4.2 Final Model Architecture**

XGBoost Configuration:

scale\_pos\_weight: 33.0699

random\_state: 42

**5. Model Performance Metrics**

**5.1 Training Performance**

* Training Accuracy: \_\_\_\_\_\_\_%
* Training Precision: \_\_\_\_\_\_\_%
* Training Recall: \_\_\_\_\_\_\_%
* Training F1-Score: \_\_\_\_\_\_\_%

**5.2 Testing Performance**

* Test Accuracy: 97.26%
* Test Precision: 34.48%
* Test Recall: 100%
* Test F1-Score: 0.5128

**5.3 Confusion Matrix**

|  |  |  |
| --- | --- | --- |
|  | Predicted Negative | Predicted Positive |
| Actual Negative | 92827 | 2607 |
| Actual Positive | 0 | 1372 |

**7. Implementation Insights**

**7.1 Key Findings**

1. Class Imbalance Impact: The dataset given to us had only 1372 positive examples out og 96806 cases. Hence, there was a huge data imbalance between positive and negative examples. We tried using smote, and sampling of negative (majority) examples. Sampling gave us a much better accuracy and hence we stuck with that.
2. Feature Importance Patterns: We ran a random forest classifier to see which features have the greatest importance for classification. We then used ‘n’ of these top features in an XGBoost Classisifier.
3. Sample Size: The model tended to overfit to the positive class when the number of negative examples were similar to the number of positive examples. Hence, we also had to manipulate the number of negative examples we take in the sample as another hyperparamter.
4. Hyperparameter Tuning: We performed hyperparameter tuning on 3 parameters: RF-sample size, XGB-sample size, number of top features. We then picked 50 random points in this 3D space, fit the model, and took the model with the most Accuracy.

We observed that the size of data used in RF-feature extraction didn’t matter very much. However, the size of the data used in XGB mattered a lot. The accuracy peaked around 33-34k sample size. Also, the best value of number of features was in the range 300-350.