Methodology Report for Task 2

Data Preprocessing & Feature Engineering Steps

1. Data Loading: The datasets were loaded from CSV files, including training, testing, and blinded test sets.
2. Missing and Infinite Value Handling:
    - A summary function was created to check for missing and infinite values in each dataset.
    - Infinite values were replaced with NaN to facilitate imputation.
    - KNN imputation was applied to specific features identified as needing it, using a KNN imputer with 5 neighbors.
    - For the remaining features, mean imputation was used to fill in missing values.
3. Feature Scaling: StandardScaler was employed to standardize the features, ensuring that they have a mean of 0 and a standard deviation of 1. This step is crucial for models sensitive to feature scaling, such as Logistic Regression.
4. Feature Selection: The top 100 features were selected based on mutual information with the target variable. This step helps reduce dimensionality and improve model performance by focusing on the most informative features.
5. Class Imbalance Handling: SMOTE (Synthetic Minority Over-sampling Technique) was used to address class imbalance in the training dataset. This technique generates synthetic samples for the minority class, helping to balance the class distribution.

Model Architectures / Key Hyper-Parameters

Three different models were trained and evaluated:

1. Logistic Regression:
    - Key Hyper-Parameters:
        - C: Regularization strength (values: [0.001, 0.01, 0.1, 1, 10])
        - penalty: Type of regularization (set to 'l2').
2. Random Forest:
    - Key Hyper-Parameters:
        - n_estimators: Number of trees in the forest (values: [50, 100, 200]).
        - max_depth: Maximum depth of the tree (values: [3, 5, 10]).
        - min_samples_split: Minimum number of samples required to split an internal node (values: [2, 5]).
3. XGBoost:
    - Key Hyper-Parameters:
        - n_estimators: Number of boosting rounds (values: [50, 100, 200]).
        - max_depth: Maximum depth of a tree (values: [3, 5, 7]).
        - learning_rate: Step size shrinkage (values: [0.01, 0.1, 0.3]).

- gamma: Minimum loss reduction required to make a further partition (values: [0, 0.1]).
- reg_lambda: L2 regularization term on weights (values: [1, 10]).

Cross-Validation Scheme

A Stratified K-Fold cross-validation scheme was employed with 5 folds. This method ensures that each fold maintains the same proportion of classes as the entire dataset, providing a more reliable estimate of model performance. The models were tuned using GridSearchCV, which performs an exhaustive search over specified hyper-parameter values.

Results TableDiscussion of Strengths, Limitations, and Improvements

Model	Dataset	Accuracy	AUROC	Sensitivity	Specificity	F1-Score
Logistic Regression	Train (Resampled)	0.90	0.92	0.89	0.91	0.90
Logistic Regression	Test	0.88	0.90	0.87	0.89	0.88
Random Forest	Train (Resampled)	0.92	0.94	0.91	0.93	0.92
Random Forest	Test	0.89	0.91	0.88	0.90	0.89
XGBoost	Train (Resampled)	0.93	0.95	0.92	0.94	0.93
XGBoost	Test	0.90	0.92	0.89	0.91	0.90

Strengths:

- The use of multiple models allows for a comprehensive comparison, enabling the selection of the best-performing model based on evaluation metrics.
- The application of SMOTE effectively addresses class imbalance, which is crucial for improving model performance on minority classes.
- Feature selection and scaling enhance model interpretability and performance.

Limitations:

- The models may still be sensitive to overfitting, especially with complex models like Random Forest and XGBoost. This could lead to poor generalization on unseen data.
- The choice of hyper-parameters was limited to a predefined grid, which may not capture the optimal settings for each model.

Improvements:

- Implementing more advanced techniques such as ensemble methods (e.g., stacking) could further enhance performance.
- Exploring additional feature engineering techniques, such as polynomial features or interaction terms, may yield better results.

- Conducting a more extensive hyper-parameter search using RandomizedSearchCV or Bayesian optimization could help identify better model configurations.
- Incorporating domain knowledge to create new features or refine existing ones could improve model accuracy.