Hospital Readmission Prediction Documentation

Data Preprocessing

1. **Importing the Two Sheets into Pandas DataFrames**
   Load the two datasets into pandas DataFrames for processing.
2. **Handling Missing Values**
   Replace all occurrences of '?' in the dataset with None (or NaN) to standardize missing values. Use the isnull() function to display the number of null values in each column.
3. **Dropping Columns with High Missing Value Percentage**
   Drop any column where the percentage of null values is greater than or equal to 90%.
4. **Displaying Null Counts in Remaining Columns**
   After dropping high-missing columns, display the number of null values in the remaining columns.
5. **Creating Encoding Dictionaries**
   Store all categorical encoding mappings in dictionaries for consistent encoding and decoding of columns.
6. **Decoding Columns Using One-Hot Encoding**
   Apply one-hot encoding to the relevant categorical columns to convert them into binary indicator variables.
7. **Formatting the Age Column**
   Clean the age data by removing characters such as "[]" and "()". Extract the numeric range from the cell and calculate the average to represent age more accurately.
8. **Converting Categorical Data into Numerical Data**
   Transform categorical variables into numeric codes to facilitate model training.
9. **Removing Low-Correlation Features**
   Remove any feature that has a correlation less than 0.3 with the target feature to reduce noise in the dataset.
10. **Detecting and Removing Outliers**
    Detect outlier rows in each column using masking techniques. Remove these rows using boolean indexing, processing one column at a time.

.

Models:

Random Forest Algorithm Tuning

1. **Number of Trees (n_estimators):**
   After performing GridSearch across multiple hyperparameter values, the optimal number of trees was found to be approximately 400.
2. **Tree Depth (max_depth):**
   The most promising maximum depth ranged between 4 and 7. A depth of 6 was selected as it provided a well-balanced recall, which was prioritized for this model.
3. **Class Weight (class_weight):**
   The balanced class weight was used to address the data imbalance, as over 90% of the samples belong to the 'No' class.
4. **Evaluation Metrics:**
   The focus was on maintaining relatively high accuracy while maximizing recall for the 'Yes' (readmitted) class to reduce false negatives.
5. **Hyperparameter Selection Criteria:**
   Recall was used as the scoring metric during GridSearchCV to prioritize correctly identifying patients likely to be readmitted within 30 days during hyperparameter tuning.

XGBoost Model Training

1. **Train-Test Split:**
   Split the data into training and testing sets with an 80% / 20% ratio. Use stratified sampling to maintain similar class distributions in both sets.
2. **Handling Class Imbalance:**
   Apply SMOTE (Synthetic Minority Over-sampling Technique) **only** to the training data to balance the minority class without affecting the test set distribution.
3. **Initialize Model:**
   Initialize the XGBoost classifier using default hyperparameters.
4. **Hyperparameter Grid Definition:**
   Define a grid of hyperparameters to tune, including max_depth, n_estimators, learning_rate, subsample, and colsample_bytree.
5. **Hyperparameter Tuning:**
   Use GridSearchCV with 3-fold cross-validation to search for the best hyperparameter combination. Optimize for the F1 score to balance precision and recall.
6. **Model Training:**
   Train the XGBoost model on the SMOTE-balanced training dataset using the best hyperparameters found.
7. **Model Evaluation:**
   Evaluate the final model on the original (unbalanced) test data. Report results using confusion matrix, classification report (precision, recall, F1-score), and accuracy score.

Logistic Regression Model Training

1. **Train-Test Split**
   Split the dataset into training and testing sets using stratified sampling to maintain the original class distribution.
2. **SMOTE Oversampling**
   Apply SMOTE (Synthetic Minority Over-sampling Technique) to the training data to balance the class distribution by generating synthetic samples for the minority class.
3. **Feature Scaling**
   Normalize feature distributions using StandardScaler, which is essential for the effective performance of Logistic Regression.
4. **Model Initialization**
   Initialize the Logistic Regression model with the saga solver (supports both L1 and L2 regularization) and set class_weight='balanced' to handle class imbalance.
5. **Hyperparameter Search Space**
   Define a grid for regularization strength (C) and penalty type (l1, l2) for tuning using RandomizedSearchCV.
6. **Hyperparameter Tuning**
   Use RandomizedSearchCV with cross-validation to find the best hyperparameters, optimizing for the F1 score.
7. **Model Training**
   Train the Logistic Regression model on the SMOTE-resampled and scaled training dataset using the best hyperparameters found.

Support Vector Machine (SVM) Model Training

1. **Data Splitting**
   The dataset is split into features (X) and labels (y). Using train_test_split, the data is divided into a training set (60%) and a testing set (40%), with a fixed random state to ensure reproducibility.
2. **Feature Scaling**
   StandardScaler is applied to standardize features so that each contributes equally to the model. The scaler is fit on the training data and then applied to both training and testing sets.
3. **Model Initialization and Training**
   An SVM classifier with a linear kernel (svm.SVC(kernel='linear')) is initialized and trained on the scaled training data. This model is well-suited for linearly separable data and supports both binary and multiclass classification.
4. **Cross-Validation**
   Five-fold cross-validation is performed on the training set to evaluate the model's robustness. The average cross-validation score estimates the model's generalization capability.
5. **Model Evaluation**
   The trained model predicts labels for the test set. Performance is assessed using:
   - classification_report for precision, recall, and F1-score breakdown
   - accuracy_score for overall accuracy
   - Manual inspection of the first 10 predictions to get a quick qualitative sense of model output.

K-Nearest Neighbors (KNN) Model Training

1. **Data Splitting**
   The dataset is divided into features (X) and labels (y). A 60%-40% train-test split is performed using a fixed random state to ensure reproducibility of results.
2. **Feature Scaling**
   Features are standardized using StandardScaler. This transformation to zero mean and unit variance is crucial for distance-based algorithms like KNN, ensuring all features contribute equally.
3. **Hyperparameter Grid Definition**
   A grid for hyperparameter tuning is defined for GridSearchCV, including:
   - n_neighbors: Number of nearest neighbors used for prediction
   - weights: Using 'distance' weighting, where closer neighbors have greater influence
   - metric: Euclidean distance as the similarity measure between points
4. **Model Initialization and Grid Search**
   A KNeighborsClassifier is instantiated, and grid search with 5-fold cross-validation is conducted over the hyperparameter grid. Accuracy is used as the scoring metric.
5. **Model Selection and Prediction**
   The best hyperparameter combination found is used to train the final KNN model, which then predicts the test set labels.
6. **Model Evaluation**
   Model performance is evaluated using:
   - Best parameters and cross-validation score from the grid search
   - Accuracy score on the test set
   - Confusion matrix for class-wise performance
   - Classification report detailing precision, recall, F1-score, and support

(EDA) Insights

1. Missing Value Handling

- Replaced all ? with NaN.
- Dropped columns with ≥90% missing data to avoid data sparsity.
- Remaining columns were cleaned or imputed appropriately.

2. Data Cleaning & Transformation

- **Age** column was reformatted by removing brackets and averaging numeric ranges to convert it into a usable numerical feature.
- **Categorical variables** were encoded via:
    - One-hot encoding (for nominal features).
    - Label encoding (for ordinal categories).
- Encoding dictionaries were created for consistency across transformations.

3. Feature Reduction

- Features with a **correlation < 0.3** to the target (readmitted) were removed.
- This helped reduce noise and improved model performance.

4. Outlier Detection

- Masking techniques were used to identify and drop outlier rows.
- This ensured more stable and reliable model training.

5. Class Distribution

- The dataset was heavily imbalanced (~90% "No", ~10% "Yes" for readmission).

- Addressed using **SMOTE** (for oversampling) and **class weighting** in models.

Feature Impact Summary

Top Predictive Features (across all models):

Feature	Insight
number_inpatient	Frequent inpatient visits strongly correlated with higher readmission.
number_emergency	High emergency visits often signal poor patient condition.
time_in_hospital	Longer stays can indicate severe cases, increasing risk.
diag_1, diag_2	Certain diagnosis codes (e.g., heart failure, diabetes) were high risk.
discharge_disposition	Discharge to nursing/recovery facilities correlated with readmission.
insulin	Changes in insulin prescription showed significant impact.
age	Older patients had a higher readmission rate.

Model Comparison & Tuning Strategies

Model	Strengths	Hyperparameter Tuning	Evaluation Focus
Random Forest	Robust to outliers, handles mixed data types well.	GridSearch on n_estimators, max_depth, etc.	Prioritized **Recall**
XGBoost	High performance, good with imbalanced data + SMOTE.	GridSearchCV on multiple params.	Prioritized **F1 Score**
Logistic Regression	Interpretable, works well with scaled data + SMOTE.	RandomizedSearchCV (C, penalty).	Balanced metrics
SVM (Linear)	Effective for linearly separable cases; sensitive to feature scaling.	Linear kernel + Cross-validation	Accuracy & F1 Score
KNN	Simple, intuitive; effective on small datasets with proper scaling.	GridSearch on n_neighbors, weights.	Accuracy & Distance

Tuning Highlights:

- **Random Forest**: Best performance with n_estimators=400, max_depth=6, class_weight='balanced'.
- **XGBoost**: Tuned max_depth, n_estimators, learning_rate, subsample – optimized for F1 on SMOTE-balanced data.
- **Logistic Regression**: Used saga solver, tuned C and penalty (l1, l2) with SMOTE and StandardScaler.
- **SVM**: Scaled features, used linear kernel with 5-fold CV.
- **KNN**: Grid-searched n_neighbors, weights='distance', used Euclidean metric, relied on scaling.

Outcomes and Insights

Key Outcomes:

- **Best Performing Model**: **XGBoost** — delivered the best **F1-score** and balanced **precision/recall**, making it most reliable for identifying readmitted patients.
- **Random Forest** came close in performance, with better interpretability and solid recall.
- **Logistic Regression** provided a baseline with high interpretability and reasonably good recall.
- **SVM** and **KNN** performed adequately but were sensitive to feature scaling and didn’t match ensemble methods on recall or F1.

Strategic Insights:

- Class imbalance handling was crucial models trained without SMOTE or class weights underperformed.
- Feature selection based on correlation and outlier removal improved generalization.
- Recall-focused tuning reduced false negatives, which is vital in healthcare settings to avoid missing high-risk patients.
- Diagnosis-related and treatment-related features (like number of procedures, insulin changes) were among the most influential in readmission prediction.

Team Members:

بيتر نادر وهبه مكرم 2023170146

بلال خالد محمد العسكري احمد 2023170141

كيفن هاني عياد عبد الله 2023170452

رامي مدحت حلمي طوبيا 2023170215

ماريو نادر سامي عزيز 2023170467