Recruitment Rate (RR) Prediction Using XGBoost

Project Overview

This project focuses on predicting the Recruitment Rate (RR) for clinical trials using machine learning techniques. By leveraging a subset of 450,000 clinical trial records from clinicaltrials.gov and employing XGBoost, a powerful gradient boosting algorithm, the model benchmarks recruitment rates to optimize clinical trial operations. The ultimate goal is to help researchers and organizations estimate recruitment efficiency effectively.

Key Features

1. **Data Analysis**
    - Conducted in-depth analysis of clinical trial data, focusing on variables that influence recruitment rates.
2. **Model Development**
    - Developed an XGBoost-based predictive model using historical clinical trial data.
3. **Benchmarking**
    - Delivered insights into recruitment efficiency and optimization strategies for clinical trial planning.

Technologies Used

- **Python**: Main programming language for all processes, including data preprocessing, modeling, and analysis.
- **XGBoost**: Machine learning algorithm for high-performance boosting on structured data.
- **Pandas**: Used for data manipulation and analysis.
- **NumPy**: Managed large multi-dimensional arrays and matrices.
- **Scikit-learn**: Utilized for building and evaluating the model.

- **Matplotlib & Seaborn**: Used for creating visualizations to explore data insights and performance metrics.

Installation

Prerequisites

Ensure the following Python libraries are installed:

```
pip install xgboost pandas numpy scikit-learn matplotlib seaborn
```

Data Preprocessing

1. **Data Cleaning**
    - Handled missing values by appropriate imputation or exclusion methods.
    - Removed outliers to ensure the model's robustness.
    - Normalized or standardized numerical features to align with model requirements.
2. **Feature Engineering**
    - Identified key features such as trial phase, country, start date, and target population.
    - Created derived features based on domain expertise, such as time elapsed since trial initiation.
3. **Data Splitting**
    - Divided the dataset into training and testing sets, using an 80-20 split.

Model Training

The core of the project is the XGBoost algorithm, a highly scalable and efficient model for tabular data.

Code Implementation

```
import pandas as pd
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Load the dataset
data = pd.read_csv("PS4_cleaned_clinical_trials.csv")

# Drop irrelevant columns or columns with excessive missing values
columns_to_drop = [
    "NCT Number", "Study Results", "Conditions", "Interventions",
    "Primary Outcome Measures", "Secondary Outcome Measures",
    "Sponsor", "Collaborators", "Other IDs", "Start Date",
    "Primary Completion Date", "Completion Date", "First Posted",
    "Results First Posted", "Last Update Posted"
]
data.drop(columns=columns_to_drop, axis=1, inplace=True)

# Handle missing values
data.fillna(data.median(numeric_only=True), inplace=True)

# Separate features and target
X = data.drop("Study Recruitment Rate", axis=1)
y = data["Study Recruitment Rate"]

# Identify categorical and numerical columns
categorical_columns = X.select_dtypes(include=["object"]).columns
numerical_columns = X.select_dtypes(include=["number"]).columns
```

```python
# Identify categorical and numerical columns
categorical_columns = X.select_dtypes(include=["object"]).columns
numerical_columns = X.select_dtypes(include=["number"]).columns

# Preprocessing for numerical and categorical data
categorical_transformer = OneHotEncoder(handle_unknown="ignore")
preprocessor = ColumnTransformer(
    transformers=[
        ("num", "passthrough", numerical_columns),
        ("cat", categorical_transformer, categorical_columns),
    ]
)

# Define the model
model = XGBRegressor(random_state=42)

# Create a pipeline
pipeline = Pipeline(steps=[("preprocessor", preprocessor), ("model", model)])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
pipeline.fit(X_train, y_train)

# Model evaluation
print("Training completed. The model is ready for predictions!")
```

Model Evaluation

After training the model, it was evaluated using the testing dataset. Key metrics include:

1. **Mean Absolute Error (MAE)**
    - Measures the average magnitude of prediction errors without considering direction.
2. **R-Squared (R^2)**
    - Indicates the proportion of variance in the dependent variable explained by the independent variables.
3. **Root Mean Squared Error (RMSE)**
    - Provides a measure of prediction error magnitude, sensitive to larger errors.

Results

- **R-Squared Value:** The model achieves an R^2 value of 0.XX, signifying the proportion of variance explained.
- **Mean Absolute Error**: The MAE is XX, reflecting the model's prediction error.

```
Confusion Matrix:
 [[   6  199    3]
 [   0 1778    0]
 [   2  380    5]]
Classification Report:
              precision    recall  f1-score   support

        High       0.75      0.03      0.06       208
         Low       0.75      1.00      0.86      1778
      Medium       0.62      0.01      0.03       387

    accuracy                           0.75      2373
   macro avg       0.71      0.35      0.31      2373
weighted avg       0.73      0.75      0.65      2373
```

Conclusion

This predictive model empowers clinical trial teams with actionable insights to understand recruitment dynamics and refine planning and execution. Incorporating more features, tuning parameters, and exploring alternative models can further enhance its predictive performance.

Future Improvements

1. **Hyperparameter Tuning**
    - Implement grid search or random search to optimize model parameters further.

2. **Feature Engineering**

    - Explore additional relevant features, such as historical trial data or geographic trends, to boost model accuracy.

3. **Model Optimization**

    - Compare XGBoost's performance with other algorithms, such as Random Forest, LightGBM, or Neural Networks.
