Biostats625_Final_Report

Yulin Shao

2024-12-17

Logistics Regression

Logistic Regression serves as an interpretable baseline model, estimating diabetes probability through the logistic function:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^{p} \beta_i X_i)}} \tag{1}$$

To handle potential overfitting with our high-dimensional health indicators, we incorporated elastic net regularization:

$$\min_{\beta_0, \beta} \left[-\sum_{i=1}^{n} \log(P(y_i|x_i)) + \lambda(\alpha||\beta||_1 + \frac{1-\alpha}{2}||\beta||_2^2) \right] \tag{2}$$

The computational intensity of cross-validation with our large dataset led us to implement two versions of logistic regression. The standard implementation processes cross-validation folds sequentially, while our optimized version leverages the doParallel package to distribute fold computations across available CPU cores. This parallel implementation maintains identical model parameters ($\alpha = 0.5$, $\lambda \in [0.001, 0.1]$) while significantly reducing computation time.

XGBoost

XGBoost (eXtreme Gradient Boosting) extends traditional gradient boosting by employing a more regularized model formalization to prevent overfitting while maintaining high predictive accuracy. The algorithm operates by iteratively adding trees through gradient boosting. At each step, XGBoost calculates the prediction error and generates a new tree that minimizes this error, expressed through the objective function:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}i) + \sum k = 1^K \Omega(f_k) \tag{3}$$

Here, the first term represents the training loss (measuring how well the model fits the data), while the second term penalizes model complexity to prevent overfitting. The regularization term is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{4}$$

where T represents the number of leaves in the tree, w_j represents leaf weights, γ controls the minimum loss reduction for node splitting, and λ controls the L2 regularization on leaf weights.

Given the dataset's size, we implemented four distinct versions of XGBoost to address memory usage and computational efficiency. The first key optimization involves data structure: we compared dense matrix format (standard but memory-intensive) versus sparse matrix format (memory-efficient for datasets with potential sparsity). The second optimization dimension involves processing strategy: sequential

versus parallel execution. This resulted in four implementations: dense-sequential, dense-parallel, sparse-sequential, and sparse-parallel. All versions maintain consistent model parameters (maximum tree depth of 6, subsample ratio of 0.8, column sampling ratio of 0.8) and utilize grid search over learning rates $0.01, 0.1, 0.3$.

The sparse matrix implementation reduces memory usage by storing only non-zero elements, particularly beneficial for categorical variables encoded as dummy variables. The parallel implementations leverage multi-threading capabilities across CPU cores, distributing the computational load of tree construction and prediction. Early stopping at 10 rounds with a maximum of 100 rounds was implemented across all versions to optimize training efficiency.

Our comprehensive approach to optimization addresses both memory constraints and computational efficiency while maintaining model performance. The parallel implementations significantly reduce training time for both algorithms, while the sparse matrix format in XGBoost provides memory efficiency without compromising predictive accuracy. All implementations were evaluated using 5-fold cross-validation with Area Under the Curve (AUC) as our primary metric, chosen for its robustness to class imbalance in medical diagnosis scenarios.