Biostats625_Final_Report

Team: - Yulin Shao - Tong Liu - Yichen Zhao - Yana Xu

2024-12-17

Introduction

Diabetes is a significant public health concern, affecting millions of individuals worldwide and leading to high healthcare costs. Early detection is critical for improving patient outcomes and reducing complications, which makes accurate and computationally efficient predictive models highly relevant. The goal of this project was to build and optimize machine learning models that can predict diabetes using demographic and health-related indicators.

We began with a smaller Heart Disease dataset (around 900 entries) to design our initial pipelines. However, the dataset's small size did not expose the computational challenges encountered in large-scale machine learning. To better reflect real-world conditions, we switched to the **CDC Diabetes Health Indicators dataset**, which contains over **250,000 records**. Handling such a large dataset required us to implement **parallel computing**, memory-efficient strategies, and algorithm-specific optimizations. This report describes the methods we used, the optimizations implemented, and their effects on training time and accuracy.

Methods

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is simple and non-parametric, but it can be very slow with huge datasets because it calculates distances to all training points. To speed things up, we applied **PCA** for numeric features and **MCA** for binary ones, cutting the dimensionality from 20 down to 9. We then rewrote the distance function in **C++** and used nth_element for **partial sorting** instead of sorting every distance. Lastly, we **parallelized** the predictions with parLapply by splitting the test data into chunks of 1,000 records each. These steps made KNN run much faster on the 250k-row dataset.

Next, we optimized distance calculations by rewriting the distance function in **C++**. Instead of sorting all distances, we used **partial sorting** with the `nth_element` function to retrieve only the top k-nearest neighbors. Finally, we implemented **parallel processing** with `parLapply` to divide the test set into smaller chunks of 1,000 samples each, which were processed concurrently.

Logistic Regression

Logistic Regression is widely used for binary classification tasks due to its simplicity and interpretability. The model maps the input features to the probability of having diabetes using the logistic function:

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

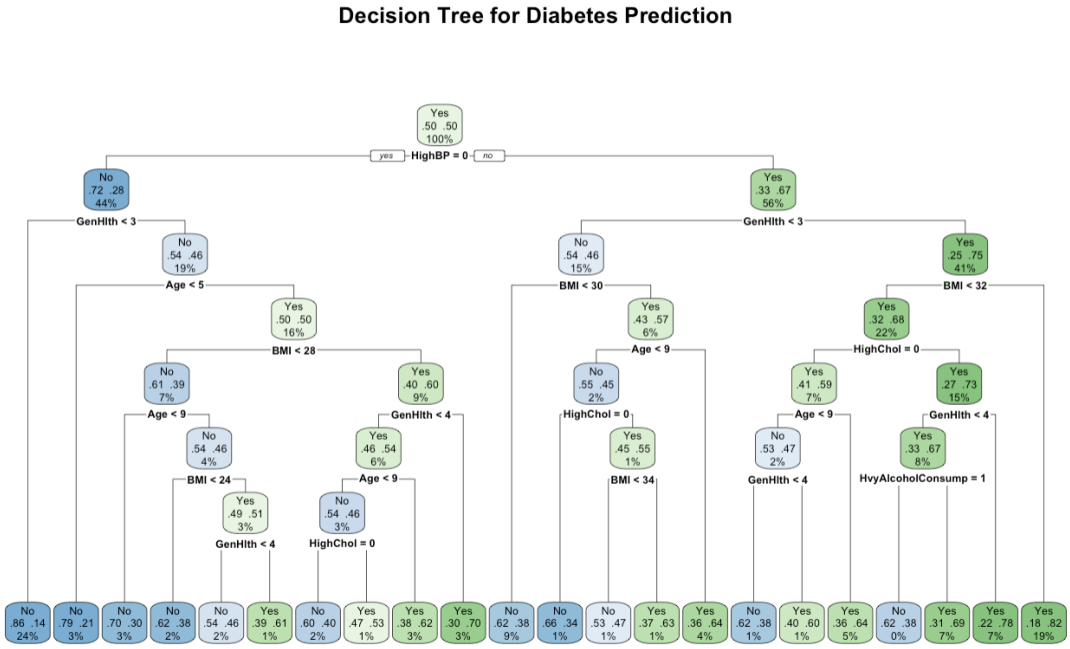
Given the large dataset, we incorporated **elastic net regularization** to handle multicollinearity and prevent overfitting:

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

Cross-validation was computationally expensive due to repeated model training. To address this, we parallelized the cross-validation process using the **doParallel** package, which enabled the division of folds across CPU cores, reducing training time significantly.

Decision Tree

A single Decision Tree (rpart) was our baseline model. Decision trees split the data on the best features at each node and are known for being easy to interpret. We performed a **grid search** over the complexity parameter (cp) in the range [0.0005, 0.02], and used parallel backends to speed up the 5-fold cross-validation. Despite being straightforward, the Decision Tree offered decent performance with minimal computation cost.



Random Forest

Random Forest improves upon single Decision Trees by combining multiple bootstrapped trees to form a robust ensemble. Initially, we used the standard `randomForest` package but encountered significant runtime issues on such a large dataset. Switching to the **ranger** package, which supports **multi-threading**, dramatically improved efficiency. Ranger reduced the training time from **362 seconds to 3.42 seconds** while maintaining strong predictive performance.

XGBoost

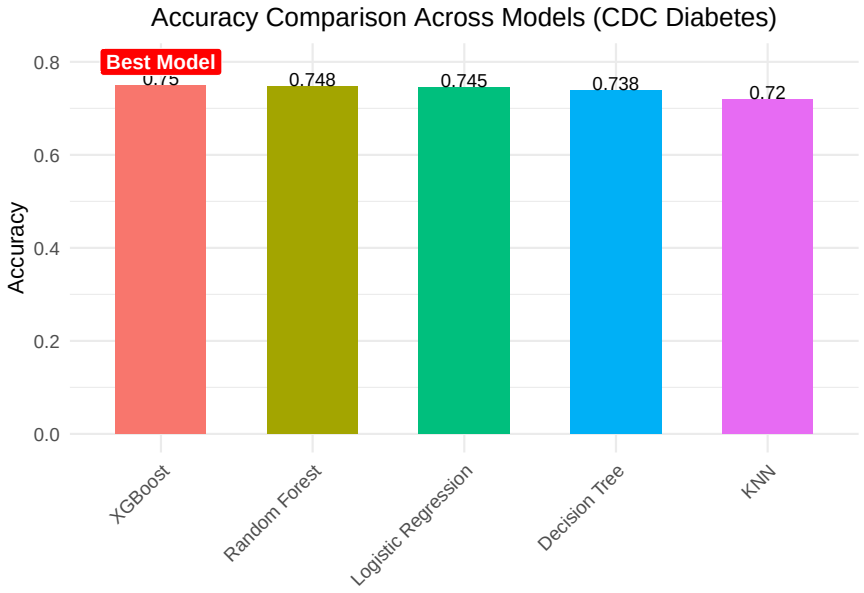
XGBoost is a gradient-boosted tree algorithm that includes parallel training and regularization. The objective function penalizes model complexity:

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

To optimize XGBoost, we tested both **dense** and **sparse matrix** formats. Sparse matrices provided significant memory savings by ignoring zero entries, which is particularly useful when dealing with categorical features. We further utilized **parallel tree building** through XGBoost's built-in multi-threading capabilities. Hyperparameter tuning was conducted over learning rates ($\eta = 0.01, 0.1, 0.3$) and tree depths, with **early stopping** at 10 rounds to prevent overfitting.

Results

Below is a bar chart comparing accuracy across all the methods we tried:



XGBoost ended up on top at **75% accuracy** (AUC around 0.82), followed by Random Forest at **74.81%** (AUC ~0.82 as well). Both ensemble methods were quite efficient after parallelization. The single Decision Tree gave us 73.76% accuracy, which was surprisingly high given how simple the model is. KNN reached **72%** accuracy after we did dimensionality reduction, rewrote distance calculations in C++, and parallelized predictions. That process cut KNN's runtime by about **62%** compared to a naive approach. Meanwhile, Logistic Regression ended up at **74.5%** accuracy and benefitted from parallel cross-validation, making it about **3x faster** to train than the naive version. Although Logistic Regression was the least accurate, it was the quickest model overall, showing that sometimes there's a trade-off between interpretability/speed and top-tier accuracy.

Conclusion

Our project highlights the importance of optimizing machine learning models for large-scale datasets, especially in critical applications like diabetes prediction. By implementing strategies such as dimensionality

reduction, sparse representations, parallel computing, and algorithm-specific optimizations, we achieved significant improvements in runtime while maintaining strong predictive accuracy. Even classic algorithms, like KNN or Logistic Regression, can work efficiently if you add dimensionality reduction, parallel computing, and specialized C++ routines. Random Forest (via ranger) and XGBoost, each near 75% accuracy and AUC ~0.82, gave the strongest performances across the board. The Decision Tree was simpler, but still got about 73.76%, which was pretty competitive given how easy it is to interpret.

Going forward, we'd like to compare runtime and memory usage in more detail, and maybe try further optimizations or ensemble methods for KNN or Logistic Regression. We believe these approaches can help identify people at high risk for diabetes sooner, potentially reducing medical costs and improving patient outcomes in real-world healthcare environments.

References

- Rios Burrows, N., Hora, I., Geiss, L. S., Gregg, E. W., & Albright, A. (2017). Incidence of End-Stage Renal Disease Attributed to Diabetes Among Persons with Diagnosed Diabetes—United States and Puerto Rico, 2000–2014. Morbidity and Mortality Weekly Report, 66(43), 1165–1170.
- Detrano, R., Jánosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology.