Machine Learning Approaches for Diabetes Prediction: A Comparative Study of Class Balancing Techniques and Model Performance

Alexandros Angelakis
Department of Computer Science
University of Crete
Heraklion, Crete
angelakis@csd.uoc.gr

Abstract

This study explores the end-to-end engineering process of machine learning model development, focusing on data preprocessing, model selection, hyperparameter tuning, and evaluation. I employ nested cross-validation to ensure robust performance estimation and compare multiple classification models on a real-world dataset. The results highlight the importance of proper data handling and evaluation in building reliable machine learning systems.

1 Introduction

Machine learning is increasingly applied across various domains, necessitating well-engineered pipelines for effective model development. This project implements a structured workflow encompassing data preprocessing, feature selection, model selection, and evaluation to ensure optimal performance on the Diabetes Prediction Dataset [1].

2 Problem Definition

The dataset consists of 100,000 observations, each representing a patient diagnosed with or without diabetes. Each observation includes eight features, such as gender, age, heart disease, smoking history, HbA1c level, and blood glucose level, along with a diagnosis indicating whether the patient has diabetes. This dataset serves as a valuable resource for researchers, students, and data scientists interested in developing machine learning models for the early detection of diabetes, exploring feature engineering techniques, and conducting classification tasks, given the large sample size available for experimentation.

2.1 Data Processing and Preprocessing

Preprocessing is a critical step in ensuring model reliability. As previously mentioned, the dataset comprises **100,000 entries** with **no missing values**. It contains an equal proportion of **categorical and numerical features**. All categorical features, except **smoking history**, were **one-hot encoded**, while the **smoking history** feature was **frequency encoded**. Additionally, all numerical features were **standardized to have zero mean and unit variance** to ensure uniform scaling.

A key challenge in this dataset is the severe **class imbalance**, with a ratio of

$$\frac{\text{class } 0}{\text{class } 1} \approx 10.82$$

[1] https://www.kaggle.com/datasets/iammustafatz/diabetes-prediction-dataset/data

Specifically, out of **100,000 samples**, **91,544** belong to **class 0** (non-diabetic patients), while only **8,456** belong to **class 1** (diabetic patients). To address this imbalance, I employ two **class distribution balancing techniques**:

1. **Random undersampling** of the majority class to match the sample size of the minority class.
2. **Class weighting** in the models to assign higher predictive weight to the minority class, ensuring better representation during training.

I also attempted to use the **Synthetic Minority Oversampling Technique (SMOTE)**, but due to the large dataset size, it was computationally expensive, and I lacked the necessary resources to implement it effectively. As a result, I decided to exclude it from my approach.

For comparison, I also evaluate my **machine learning pipeline without any class balancing** to assess its impact on model performance.

3 Machine Learning Pipeline

3.1 Feature Selection

Before training any model, I tested two different feature selection algorithms: **LASSO** and **Backward Elimination**.

For the **LASSO** algorithm, I performed cross-validation to determine the optimal value of α (the regularization term). The best value found for α was consistently **0.0001**, and the algorithm selected all features of the dataset, using an **importance threshold of 0.005**. Figure 1 illustrates the relative importance of each feature.

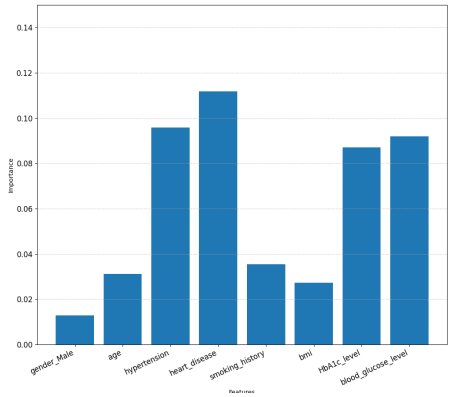

Figure 1: Feature importance from LASSO feature selection

Algorithm 1 Backward Selection

Require: FS Selected Variables S

while S changes **do**
  // Identify V^* with the maximum p-value given $S \setminus V$
  $V^* \leftarrow \arg\max_{V \in S} \text{pvalue}(T; V | S \setminus V)$
  if $\text{pvalue}(T; V^* | S \setminus V) > a$ **then**
    $S \leftarrow S \setminus V^*$
  end if
end while
return S

The **Backward Elimination** algorithm produced the same results, as all **p-values** in each iteration were **very close to zero**, preventing the removal of any features. Therefore, all features were deemed important for our classification task.

3.2 Model Selection and Training

I evaluate multiple machine learning models to determine the most effective classifier:

Model	Hyperparameters
Logistic Regression	C: [0.001, 0.01, 0.1, 1, 10, 100] Penalty: ['l1', 'l2'] Solver: ['liblinear', 'saga']
Support Vector Classifier (SVC)	C: [0.1, 1, 10, 100] Kernel: ['linear', 'rbf', 'poly', 'sigmoid'] Gamma: ['scale', 'auto']
Random Forest Classifier	n_estimators: [50, 100, 200] Max_depth: [None, 5, 10, 20]
XGBoost Classifier	Learning Rate: [0.01, 0.1, 0.2] n_estimators: [50, 100, 200] Max_depth: [3, 5, 7]
Decision Tree Classifier	Max_depth: [None, 5, 10, 20] Min_samples_split: [2, 5, 10]
K-Nearest Neighbors (KNN)	n_neighbors: [3, 5, 7, 9] Weights: ['uniform', 'distance'] Metric: ['euclidean', 'manhattan', 'minkowski']
Gaussian Naïve Bayes (GNB)	Var_smoothing: [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]

Table 1: Machine Learning Models and Hyperparameter Configurations

Each model was implemented using scikit-learn or XGBoost, with hyperparameters optimized through cross-validation. The dataset was split into 90% training samples and 10% holdout testing samples. The total number of models trained using nested cross-validation to select the best one was 6,800.

3.3 Cross-Validation and Hyperparameter Tuning

Nested cross-validation is a robust model evaluation technique used to obtain an unbiased estimate of model performance while also tuning hyperparameters. The procedure consists of two nested loops: an outer loop for model evaluation and an inner loop for hyperparameter tuning. The outer loop performs stratified 10-fold cross-validation on the dataset, splitting the data into training and testing subsets in each iteration. The inner loop performs another level of stratified 5-fold cross-validation exclusively on the training data of the corresponding outer fold. This step is responsible for hyperparameter tuning. For the inner loop, I used multiple evaluation metrics such as ROC-AUC, F1-score, precision, recall, accuracy, balanced accuracy and many more.

The model with the best hyperparameters from the inner loop is used to make predictions on the outer test set. This process is repeated 10 times, producing 10 model evaluations and from these, the best overall model is selected based on its average AUC score across all outer folds. The final model is retrained on the entier (training) dataset using the best hyperparameters.

This procedure ensures that the final model generalizes well to unseen data while optimizing hyperparameters effectively.

4 Experiments and Evaluation

I conducted three different types of experiments: one without balancing the dataset, one by adjusting the model weights, and one using random undersampling.

4.1 Without balancing the dataset

This experiment took approximately **1 hour and 40 minutes** to complete. The best configuration found through **nested cross-validation** is presented in Table 2.

Parameter	Value
Model	XGBoost Classifier
Learning Rate	0.2
Max Depth	3
Number of Estimators	100
Average AUC	0.9829

Table 2: Best Model Configuration from Nested Cross-Validation

After evaluating this configuration on the **holdout test set**, we obtain the **confusion matrix** and the **ROC curve**, shown in Figure 2, as well as the performance metrics summarized in Table 3.

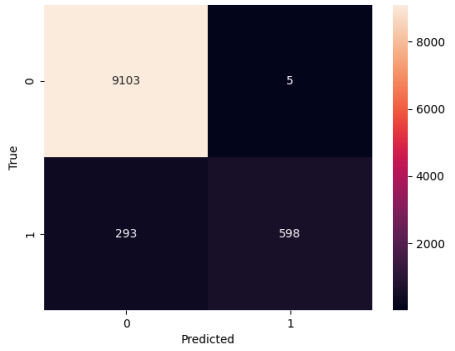

Figure 2: Confusion matrix

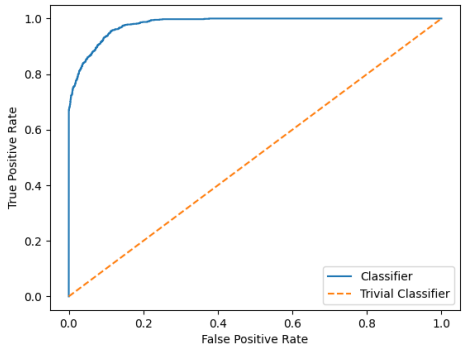

Figure 3: ROC curve

Metric	Value
ROC AUC	0.9806
Average Precision	0.8915
F1 Score	0.8005
F1 Macro	0.8922
Precision	0.9917
Recall	0.6712
Balanced Accuracy	0.8353
Accuracy	0.9702
Matthews Correlation Coefficient	0.8026

Table 3: Performance Metrics of the Best Model

From the **confusion matrix** and the **metrics**, we observe that the number of **false negatives (Type II errors)** is significantly high, resulting in a **low recall (0.6712)**. While a **high precision** (low false positives) is desirable, in **medical applications**, recall is often prioritized to **avoid missing critical cases**.

In particular, for **diabetes detection**, failing to identify diabetic patients (**high FN rate**) may result in these individuals continuing unhealthy habits, leading to a deterioration in their condition.

This issue arises due to the **imbalance between class 0 (non-diabetic) and class 1 (diabetic)**. The model is more sensitive to the **majority class (non-diabetic)** because of the **large class difference**, leading to a bias where the model frequently predicts **class 0 (non-diabetic)** even when the true label is **class 1 (diabetic)**.

It is also important to determine which features have greater predictive power than others. Figure 4 shows that the **HbA1c level**, which measures a person's average blood sugar level over the past 2–3 months, the **blood glucose level**, which refers to the amount of glucose in the bloodstream at a given time, **hypertension** and **age** contribute the most to the classification of diabetes, as expected.

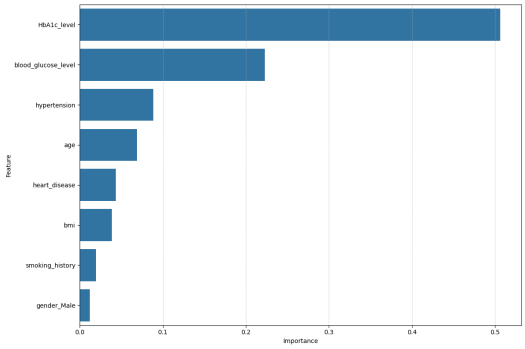

Figure 4: Feature contribution to prediction

4.2 Balancing the dataset

The **random undersampling** experiment took only **5 minutes** to complete because the majority class (**90,000 training samples**) was undersampled to approximately **8,000** to match the minority class. This significant reduction in data size made the entire process much faster.

On the other hand, **weight balancing** for each model took considerably longer than training without any balancing, with an execution time of approximately **2 hours and 40 minutes**.

The best configuration found through **nested cross-validation** was the same for both experiments and is presented in **Table 4**.

Parameter	**Value**
Model	XGBoost Classifier
Learning Rate	0.1
Max Depth	3
Number of Estimators	200
Average AUC	0.9831

Table 4: Best Model Configuration from Nested Cross-Validation

After evaluating these configurations on the **holdout test set**, we obtain the **confusion matrices** and the **ROC curves**, shown in Figures 5, 6, 7, 8 as well as the performance metrics summarized in Tables 5.

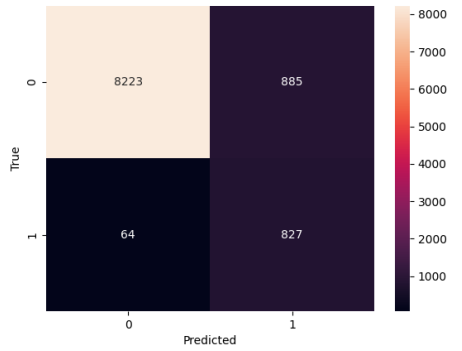

Figure 5: Confusion matrix undersampling

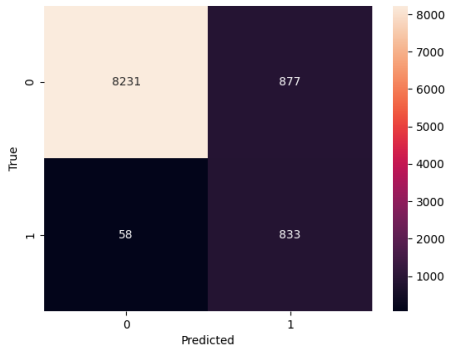

Figure 6: Confusion matrix weight balancing

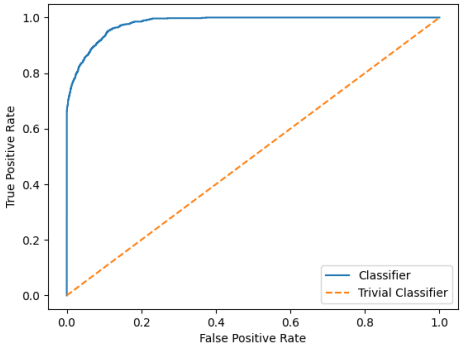

Figure 7: ROC undersampling

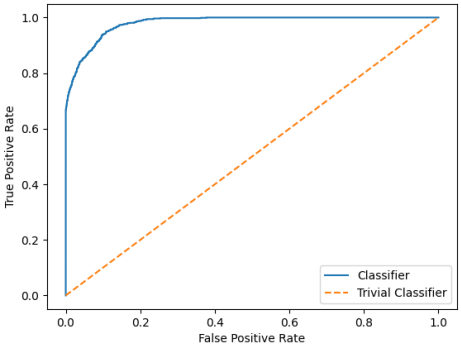

Figure 8: ROC weight balancing

Metric	Random Undersampling	Weight Balancing
ROC AUC	0.9805	0.9805
Average Precision	0.8916	0.8913
F1 Score	0.6354	0.6405
F1 Macro	0.7904	0.7934
Precision	0.4831	0.4871
Recall	0.9282	0.9349
Balanced Accuracy	0.9155	0.9193
Accuracy	0.9051	0.9065
Matthews Correlation Coefficient	0.6285	0.6345

Table 5: Comparison of Performance Metrics Between Random Undersampling and Weight Balancing

Both experiments exhibit similar metrics and behavior. Therefore, it is more efficient to use **random undersampling** to reduce execution time and lower the computational cost of the algorithms.

The recall has significantly increased, approaching a value of **0.93**. In contrast, precision has decreased considerably, nearing **0.48**, which can also be observed in the confusion matrices. This occurs because the **minority class** is now receiving more attention from the models, allowing for better classification. Additionally, **balanced accuracy** has increased by almost **10%**, further indicating improved classification performance.

Finally, we can also see in Figures 9 and 10 how the features contribute to classification in these experiments. As expected, the features most relevant to diabetes are the ones that contribute the most.

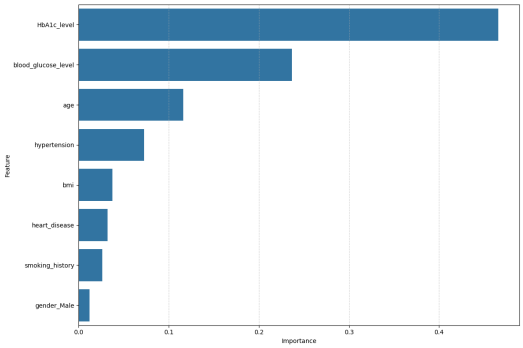

Figure 9: Feature contribution to prediction undersampling

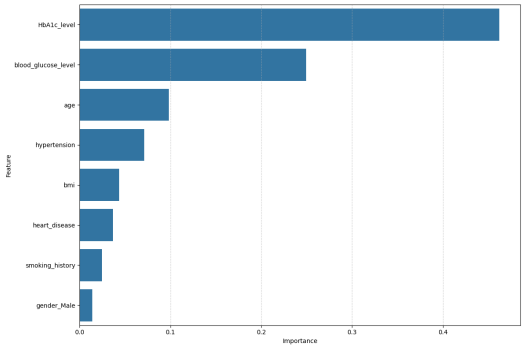

Figure 10: Feature contribution to prediction weight balancing

5 AutoML - JADBio

For comparison purposes, I also used an AutoML software called JADBio. This software allows you to input your dataset, and it automatically runs a machine learning pipeline to find the best model configuration.

JADBio's AI system estimated the out-of-sample performance of the models produced by each configuration using a 90% - 10% hold-out method. Overall, 171 models were trained. The best configuration JADBio found is as follows:

For preprocessing, it used Mean Imputation, Mode Imputation, Constant Removal, and Standardization. For feature selection, it utilized the Epilogi algorithm with hyper-parameters `equivAlpha` = 0.01 and a `stopping criterion` of the Independence Test with a threshold of 0.01. Finally, for the predictive algorithm, it employed a Classification Decision Tree with a Deviance splitting criterion and hyper-parameters: `minimum leaf size` = 4, and pruning parameter `alpha` = 0.05.

The metrics of this configuration is shown in Table 6

JADBio achieved high precision along with high recall but did not manage to attain high balanced accuracy. There is a trade-off between balanced accuracy and high precision.

Finally, for the feature selection part, JADBio also selected all the features for classification. The performance achieved by adding each feature in sequence to the model relative to the performance of the final model with all selected features is shown below in Figure 11. The features are added in order of importance:

Some features may not seem to add predictive performance to the model; however, the feature selection algorithms include them as an effort to make the final model more robust to noise.

Metric	Mean Estimate	CI
ROC AUC	0.971	[0.963, 0.978]
Mean Average Precision	0.928	[0.915, 0.941]
F1 Score	0.983	[0.981, 0.986]
F2 Score	0.991	[0.989, 0.992]
F0.5 Score	0.975	[0.972, 0.979]
Accuracy	0.970	[0.966, 0.975]
Balanced Accuracy	0.825	[0.805, 0.846]
Matthews Correlation Coefficient	0.780	[0.752, 0.809]
Precision	0.970	[0.966, 0.975]
True Positive Rate (Sensitivity, Recall, Hit Rate)	0.996	[0.994, 0.998]
Specificity	0.673	[0.635, 0.715]
True Positive Ratio	0.911	[0.904, 0.919]
True Negative Ratio	0.057	[0.052, 0.063]
False Positive Ratio	0.028	[0.023, 0.032]
False Negative Ratio	0.004	[0.002, 0.005]

Table 6: Performance Metrics with Confidence Intervals

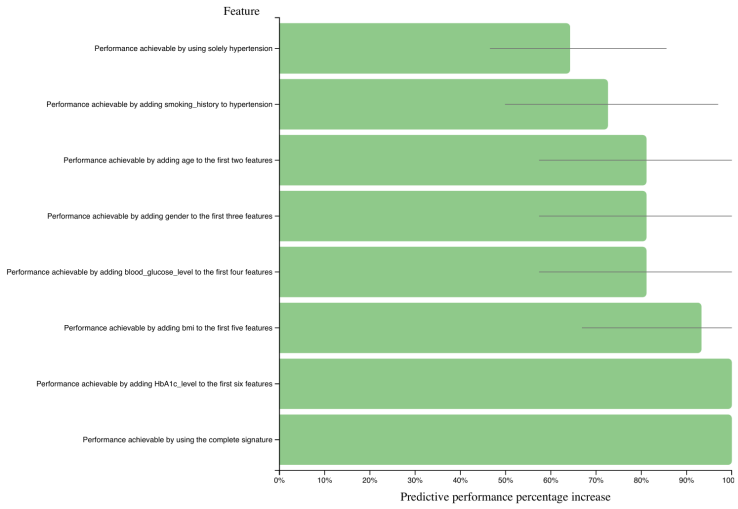

Figure 11: Feature contribution to predictive performance

6 Conclusion

In this study, I explored the complete machine learning pipeline for **diabetes prediction**, focusing on **data preprocessing, feature selection, model selection, and evaluation**. I employed **nested cross-validation** to ensure robust performance estimation and examined multiple classification models to determine the most effective approach.

Our experiments highlight the significant impact of **class imbalance** on model performance. Initially, without any balancing techniques, the model exhibited **low recall (0.6712)**, indicating a high false negative rate. To address this issue, I experimented with **random undersampling** and **weight balancing** techniques.

The results demonstrated that **both techniques improved recall** by prioritizing the minority class, with recall increasing to **0.93**. However, this came at the cost of **lower precision (0.48)**. Despite

this trade-off, **balanced accuracy improved by nearly 10%**, indicating better overall classification performance. Overall, **tree-based models, particularly XGBoost, performed the best across all experiments as well as AutoML**.

Furthermore, **random undersampling** significantly reduced computational time, completing in just **5 minutes**, compared to **2 hours and 40 minutes** for weight balancing. Since both techniques produced similar performance metrics, I conclude that **random undersampling** is the preferred approach due to its efficiency in terms of **execution time and computational cost**.

This study underscores the importance of **handling class imbalance** in medical machine learning applications, where **recall is prioritized over precision** to minimize false negatives. Future work could focus on finding a balance between precision and recall, rather than considering only recall, and explore **advanced resampling techniques**, such as **SMOTE** or **generative adversarial networks (GANs)**, to improve class balance while preserving the original data distribution.

Special thanks to my professor, Ioannis Tsamardinos, for introducing me to the beauty of Machine Learning and for granting me access to his AutoML software.

References

[1] Kavakiotis, I., Tsave, O., Salifoglou, A., Maglaveras, N., Vlahavas, I., & Chouvarda, I. (2017). "Machine learning and data mining methods in diabetes research." Computational and Structural Biotechnology Journal, 15, 104-116.

[2] He, H., & Garcia, E. A. (2009). "Learning from imbalanced data." IEEE Transactions on Knowledge and Data Engineering, 21(9), 1263-1284.

[3] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research, 16, 321-357.

[4] Chen, T., & Guestrin, C. (2016). "XGBoost: A scalable tree boosting system." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

[5] Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). "Efficient and robust automated machine learning." Advances in Neural Information Processing Systems (NIPS), 28, 2962-2970.

[6] Kohavi, R. (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection." Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), 2, 1137-1145.