Project 3

Sheuly Debnath

December 13, 2024

Abstract

This project explores and compares various methods for classifying the Pima Indians Diabetes dataset, predicting whether a patient has diabetes based on eight features.

The study begins with tree-based methods, including Decision Trees with cost-complexity pruning and ensemble techniques like Random Forest, AdaBoost and XGBoost. We analyze their performance as the number of trees in the ensemble increases and compare the results to scikit-learn implementations. Variable importance from the Random Forest model is also used for feature selection.

Next, we implement a Feed-Forward Neural Network with customizable hidden layers and a Logistic Regression model trained using Stochastic Minibatch Gradient Descent. Hyperparameters such as learning rate, hidden layers, neurons, and epochs are tuned for optimal performance.

Finally, we introduce XGBoost, a powerful gradient boosting method known for its performance in classification tasks. XGBoost was implemented and compared alongside the other models, achieving the best accuracy of 100%.

All methods are evaluated using metrics like accuracy, precision, recall, specificity, F-score, and AUC. Among the methods, Random Forest achieved an accuracy of 0.915584, while Logistic Regression demonstrated the best recall of 0.901961. However, XGBoost outperformed all methods in terms of accuracy, achieving a perfect 1.00, making it the strongest model for predictive tasks in this analysis.

1 Introduction

Diabetes is a serious, chronic disease that occurs either when the pancreas does not produce enough insulin (a hormone that regulates blood glucose), or when the body cannot effectively use the insulin it produces. Raised blood glucose, a common effect of uncontrolled diabetes, may, over time, lead to serious damage to the heart, blood vessels, eyes, kidneys and nerves. More than 400 million people live with diabetes [1]. Early diagnosis is critical to reducing medical costs and preventing more serious conditions.

This project uses the Pima Indian Diabetes dataset, containing data from 768 female patients aged 21 and older, to predict diabetes based on health and physical characteristics like BMI and glucose levels. The dataset is particularly significant as the Pima Indian population has a high incidence of diabetes.Pima Indians are a Native American group that lives in Mexico and Arizona, USA [8]: this minority group was discovered

to present a high incidence rate of diabetes mellitus. Hence, research around them was thought to be significant.

We implement and compare several machine learning methods for classification, including tree-based approaches (Decision Tree, Random Forest, and AdaBoost,), Logistic Regression, and Feed-Forward Neural Networks. Models are evaluated using performance metrics such as accuracy, precision, recall, specificity, F-score, and AUC. Interpretability is also considered, as it is essential in healthcare applications. In determining which method would be best in analysing our dataset we also look at interpretability of its results: in fact, the ability to explain the reasonings behind a machine learning prediction is particularly crucial in the healthcare field, where mistakes could be fatal [2]

The report is organized as follows: Section 2 describes the methods used, Section 3 presents the analysis and results, and Section 4 provides a comparison of the models. The project code is available at: [GitHub Repository Link].

2 Method

2.1 Decision trees

A Decision Tree is a supervised learning algorithm commonly used for classification and regression tasks. It is structured as a tree-like model of decisions, where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents an outcome (class label or continuous value). With this basic algorithm we can in turn build more complex networks, spanning from homogeneous and heterogenous forests (bagging, random forests and more) to one of the most popular supervised algorithms nowadays, the extreme gradient boosting, or just XGBoost [6] . But let us start with the simplest possible ingredient.A decision tree is typically divided into a root node, the interior nodes, and the final leaf nodes or just leaves. These entities are then connected by so-called branches.

To classify an instance, start at the root node, test the attribute specified, then follow the branch corresponding to the attribute's value. This process repeats for the next node until a leaf is reached, providing the classification.

When considering a classification problem, the prediction are made as follows:

We define a PDF p_{mk} that represents the number of observations of a class k in a region R_m with N_m observations. We represent this likelihood function in terms of the proportion $I(y_i = k)$ of observations of this class in the region R_m as

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k).$$

We let p_{mk} represent the majority class of observations in region m. The three most common ways of splitting a node are given by

* Misclassification error :

$$p_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i \neq k) = 1 - p_{mk}.$$

* Gini index g:

$$g = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}).$$

* Information entropy or just entropy s

$$s = -\sum_{k=1}^{K} p_{mk} \log p_{mk}.$$

2.1.1 Pruning the tree: Cost complexity pruning

Decision tree often leads to overfitting and unnecessarily large and complicated trees.

The basic idea is to grow a large tree T_0 and then prune it back in order to obtain a subtree. A smaller tree with fewer splits (fewer regions) can lead to smaller variance and better interpretation at the cost of a little more bias.

The so-called Cost complexity pruning algorithm gives us a way to do just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α.

For each value of α there corresponds a subtree $T \in T_0$ such that

$$\sum_{m=1}^{\overline{T}} \sum_{i:x_i \in R_m} (y_i - \overline{y}_{R_m})^2 + \alpha \overline{T},$$

Cost Complexity Pruning aims to find the smallest tree that balances complexity and fit to the training data. The tuning parameter α controls the trade-off between the tree's complexity (number of terminal nodes) and its accuracy.

When $\alpha = 0$, the tree is fully grown, and the focus is on minimizing the training error. As α increases, the penalty for having more terminal nodes grows, leading to pruning of branches and resulting in a simpler tree. The pruning process removes branches in a nested and predictable way as α increases. By adjusting α, we can obtain a sequence of subtrees. The optimal α can be selected using a validation set or cross-validation, after which the corresponding subtree is chosen based on the full dataset [6].

2.1.2 Ensemble Methods: From Single Trees to Many Trees

A single decision tree often overfits the training data, resulting in high variance. Ensemble methods address this issue by combining multiple models to reduce variance and improve performance. These methods can use either homogeneous or heterogeneous models, forming "forests" or "jungles" of trees. Key ensemble techniques include [7]:

Voting Classifiers: Combine predictions from multiple models for a final decision.

Bagging and Pasting: Build models using different subsets of data, with or without replacement.

Random Forests: An ensemble of decision trees built using bagging and random feature selection.

Boosting Methods: Sequentially improve weak learners, including Adaptive Boosting (AdaBoost) and Extreme Gradient Boosting (XGBoost).

2.1.3 Random Forest

Random Forest is an ensemble method. The idea behind it is to reduce the high variance associated to large decision trees by building a large collection of decorrelated trees, and averaging their results. Random forest improves over bagging methods by not only reducing correlation by training trees on different bootstrap samples but also by considering just a subset of variables at each split [7].

Random Forest Algorithm The algorithm described here can be applied to both classification and regression problems.

We will grow of forest of say B trees. 1. For $b = 1 : B$

* Draw a bootstrap sample from the training data organized in our $\boldsymbol{X}$ matrix.

* We grow then a random forest tree T_b based on the bootstrapped data by repeating the steps outlined till we reach the maximum node size is reached

1. we select $m \leq p$ variables at random from the p predictors/features

2. pick the best split point among the m features using for example the CART algorithm and create a new node

3. split the node into daughter nodes

4. Output then the ensemble of trees $\{T_b\}_1^B$ and make predictions for either a regression type of problem or a classification type of problem.

The number m of variables considered at each split regulates the model's complexity and its ability to reduce correlation. For m = n the model is equivalent to bagging. The suggested default value for m is p for classification and

$$p/3$$

for regression. However, as argued by Hastie et al [5] , it could still be beneficial to tune this parameter to obtain the best results.

2.1.4 Variable importance

Variable importance plots can also be generated for random forests. During each split in every tree, the importance of the variable used for splitting is measured by the improvement in the loss function resulting from that split. This measure is then accumulated across all trees in the forest for each variable individually. To compute the importance of each variable ℓ in each tree T, we assess the change in the model's performance due to the split on ℓ.

The importance measure for a single tree T is given by:

$$I_\ell^2(T) = \sum_{t=1}^{J-1} \hat{\imath}_\ell^2 \mathbf{1}(v(t) = \ell),$$

where:

- t is an internal node, $t = 1, \ldots, J-1$;
- $v(t)$ is the variable selected for the partition at node t;
- $\hat{\imath}_\ell^2$ is the estimated improvement in loss due to the split.

The extension to a random forest is:

$$I_\ell^2 = \frac{1}{B}\sum_{b=1}^{B} I_\ell^2(T_b),$$

where B is the total number of trees in the forest.

This measure is much more reliable due to the stabilizing effect of averaging. The importance values are scaled to sum to 1.

2.1.5 AdaBoost

Boosting is recognized as one of the most powerful learning techniques of recent decades. Initially introduced by Freund and Schapire in 1997, it was designed for classification problems and later adapted for regression tasks. The main objective of boosting is to iteratively apply "weak" learners to successively modified versions of the dataset, thereby generating a sequence of classifiers $G_m(x)$, $m = 1, 2, \ldots, M$. These classifiers are then combined using a weighted majority vote to produce a final, more accurate prediction.

A notable example of a boosting method is **AdaBoost**. In the context of a two-class classification problem, where the output variable Y is coded as $Y \in \{-1, 1\}$ and is based on a vector of predictor variables X, AdaBoost follows the steps outlined in the algorithm below.

Basic Steps of AdaBoost The basic idea is to set up weights which will be used to scale the correctly classified and the misclassified cases [6] . 1. We start by initializing all weights to $w_i = 1/n$, with $i = 0, 1, 2, \ldots n-1$. It is easy to see that we must have $\sum_{i=0}^{n-1} w_i = 1$.

2. We rewrite the misclassification error as

$$\overline{\mathrm{err}}_m = \frac{\sum_{i=0}^{n-1} w_i^m I(y_i \neq G(x_i))}{\sum_{i=0}^{n-1} w_i},$$

1. Then we start looping over all attempts at classifying, namely we start an iterative process for $m = 1 : M$, where M is the final number of classifications. Our given classifier could for example be a plain decision tree.

a. Fit then a given classifier to the training set using the weights w_i.

b. Compute then err and figure out which events are classified properly and which are classified wrongly.

c. Define a quantity $\alpha_m = \log\left(1 - \overline{\mathrm{err}}_m\right)/\overline{\mathrm{err}}_m$

d. Set the new weights to $w_i = w_i \times \exp\left(\alpha_m I(y_i \neq G(x_i)\right.$.

5. Compute the new classifier $G(x) = \sum_{i=0}^{n-1} \alpha_m I(y_i \neq G(x_i)$.

For the iterations with $m \leq 2$ the weights are modified individually at each steps. The observations which were misclassified at iteration $m-1$ have a weight which is larger than those which were classified properly. As this proceeds, the observations which were difficult to classifiy correctly are given a larger influence. Each new classification step m is then forced to concentrate on those observations that are missed in the previous iterations .

2.2 XGBoost: Extreme Gradient Boosting

XGBoost or Extreme Gradient Boosting, is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable [7] . It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way for more [3] . By minimizing the residual errors iteratively, it enhances the predictive power of the model.

2.3 Logistic Regression

Logistic regression is used for classification problems to estimate the probability that a given datapoint $\mathbf{x} = (x_1, \ldots, x_N)$ leads to outcome y_i for $i = 1, \ldots, k$, where k is the number of possible labels for the outcome classes. We assume now that we have two classes with y_i either 0 or 1. By using the sigmoid function, we can write the probabilities of each of the two outcomes as:

$$p(y_i = 1 \mid \mathbf{x}_i, \boldsymbol{\beta}) = \frac{e^{\boldsymbol{\beta}^T \mathbf{x}_i}}{1 + e^{\boldsymbol{\beta}^T \mathbf{x}_i}}$$

$$p(y_i = 0 \mid \mathbf{x}_i, \boldsymbol{\beta}) = 1 - p(y_i = 1 \mid \mathbf{x}_i, \boldsymbol{\beta})$$

where $\boldsymbol{\beta}$ is the vector of the parameters of the model which we want to estimate. To simplify the notation, we will call $p(y_i = 1 \mid \mathbf{x}_i, \boldsymbol{\beta}) = p_1(\mathbf{x}_i, \boldsymbol{\beta})$ and $p(y_i = 0 \mid \mathbf{x}_i, \boldsymbol{\beta}) = p_0(\mathbf{x}_i, \boldsymbol{\beta})$. Furthermore, our results were obtained by introducing an L_2 regularization term λ. The cost function we seek to minimize is:

$$C(\boldsymbol{\beta}) = -\sum_{i=1}^{n} \left(y_i \boldsymbol{\beta}^T \mathbf{x}_i - \log\left(1 + e^{\boldsymbol{\beta}^T \mathbf{x}_i}\right)\right) + \lambda \|\boldsymbol{\beta}\|_2^2$$

In order to obtain the optimal vector of parameters $\boldsymbol{\beta}$, we will use the mini-batch stochastic gradient descent algorithm. for more information from my project 2-[4]

2.3.1 Feed-Forward Neural Network

Neural Networks (NNs) are computational learning systems based on a collection of connected units or nodes called neurons, which reproduce the activity of neurons in the brain. This project focuses on a specific NN model called the Multilayer Perceptron (MLP): all nodes of a layer will be connected to every node in the subsequent layer, making it a fully connected network. Each connection in the network is associated with a weight, and each node has an intrinsic bias parameter and an activation function, which returns the output of the node. Hence, the output node y is produced by the activation function f:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b_i\right) = f(z)$$

where w_i are the weights, b_i are the biases, and x_i are the inputs, which correspond to the outputs of the nodes in the previous layer.

Figure 1 shows a visualization of this process.

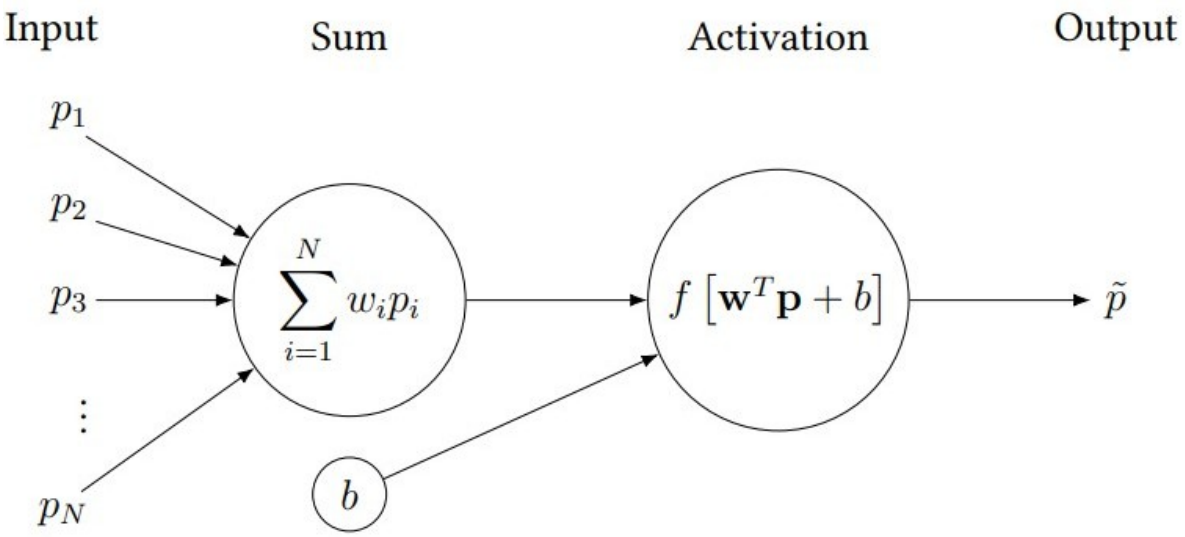

Figure 1: The feed-forward algorithm: the input values p_i, for $i = 1, \ldots, N$, from the first layer are multiplied by the corresponding weights w_i, for $i = 1, \ldots, N$, and summed. Then, the activation function f is applied to $\mathbf{w}^T\mathbf{p} + \mathbf{b}$, where $\mathbf{b}$ is a vector of dimension $N \times 1$. The output then becomes the input for the neurons in the next layer.

By repeating the feed forward and back propagation algorithms over a certain number of epochs, we hope to move in the coefficient space towards a minimum for the cost function. In our case, to reduce the computational cost of the process we will use mini-batch SGD [4]

3 Results and Discussion

3.1 Pima Indians Diabetes Dataset

The dataset we have considered in this analysis is been vastly used in classification settings since its publication in 1988. This dataset is originally from the US National Institute of Diabetes and Digestive and Kidney Diseases and its objective is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements. The data is selected from a larger diabetes database by imposing several constraints: in particular, all patients considered are females at least 21 years old of Pima Indian heritage.

Feature	Description	Data type
Preg	Number of times pregnant	Integer
Glucose	Plasma glucose concentration at 2 Hours (GTIT)	Continuous
Press	Diastolic Blood Pressure (mm Hg)	Continuous
Skin	Triceps skin fold thickness (mm)	Continuous
Insulin	2-Hour Serum insulin (lh/ml)	Continuous
BMI	Body mass index [weight in kg/(Height in m)]	Continuous
DPF	Diabetes pedigree function	Continuous
Age	Age (years)	Integer
Outcome	Binary value indicating non-diabetic/diabetic	Factor

Table 1: Summary of the Pima Indians Database dataset variables.

The dataset contains information about 768 patients and has no null values and no missing values. However, as noted in 5, some attributes present inconsistent values: glucose concentration (Gluc), blood pressure (BP), skin fold thickness (Skin), insulin and BMI, contain zero values which are not within the normal range and are therefore inaccurate. Since the dataset is not very large, we prefer avoiding removing rows unnecessarily: for this reason, the zero values in the cited columns have been replaced by the column mean. The resulting distributions for all variables are presented in Figure 2.

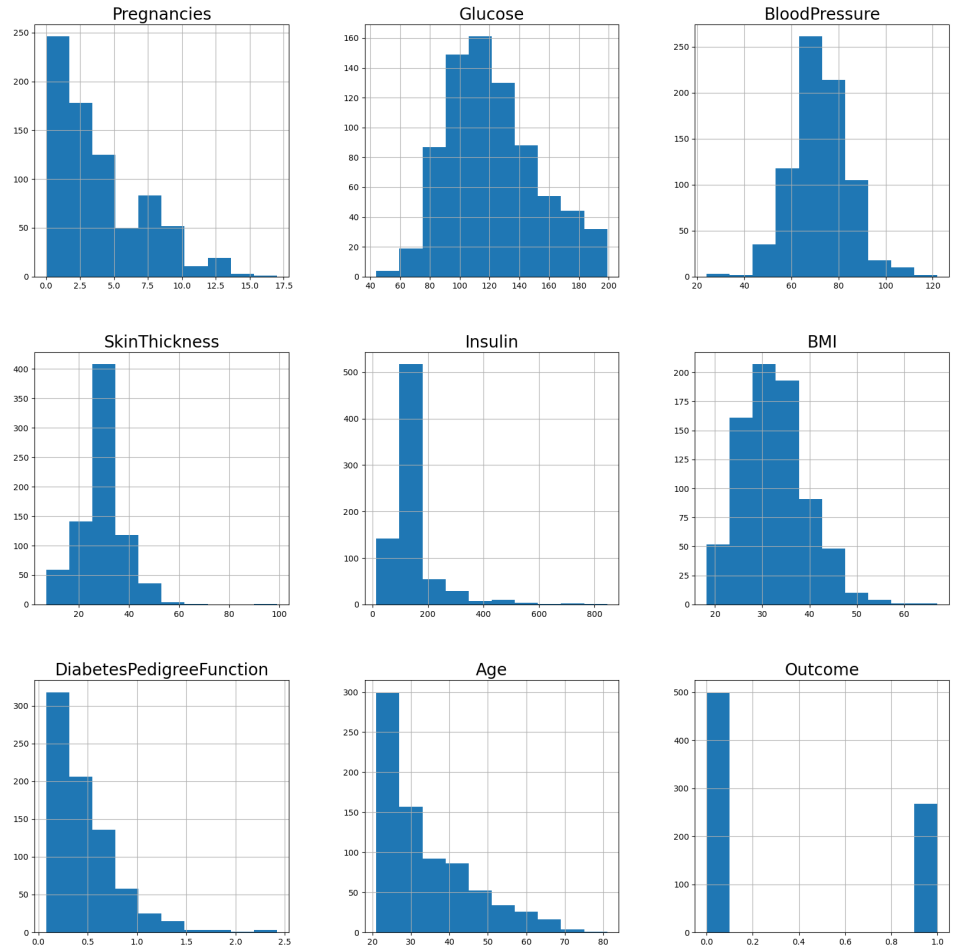

Figure 2: Histograms for the 8 predictors in the Pima Indians Diabetes dataset and the Outcome variable after data cleaning.

It's interesting to observe that the dataset is imbalanced, with nearly twice as many cases belonging to class 0 compared to class 1. This will be an important factor to consider when comparing the performance of the different methods.

We can also examine the correlation between the predictors and the outcome, as shown in Figure 3. From the results, we see that the variables most strongly linked to the outcome are Glucose and BMI. Other pairs of predictors that are highly correlated include BMI and Skin Thickness, as well as Age and Pregnancies.

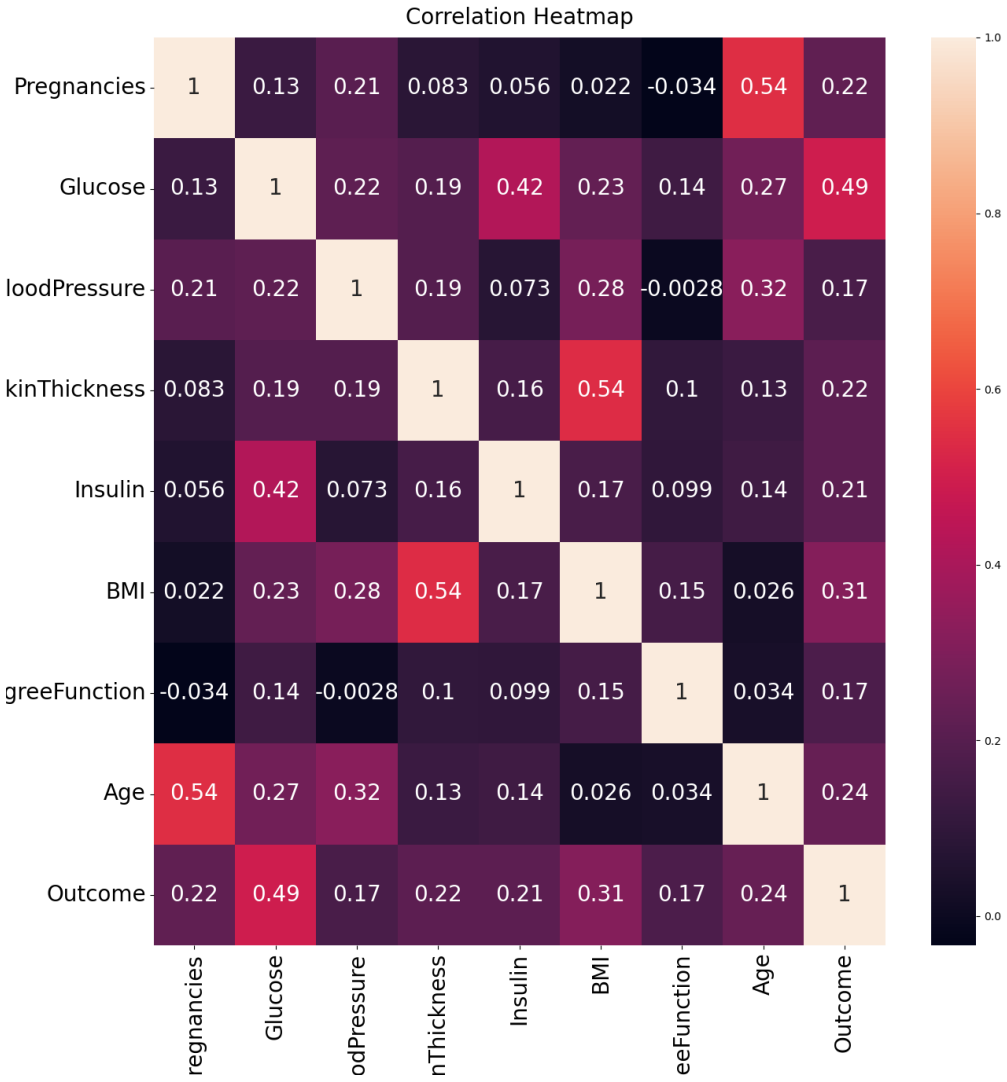

Figure 3: Correlation matrix for the 8 predictors in the Pima Indians Diabetes dataset and the Outcome variable.

3.2 Implementation

To better understand the methods used in our analysis, we tried creating our own implementations.

For tree-based methods, we designed the implementation using object-oriented programming. The base class, 'Node', represents a single node in a classification tree. If the node isn't a leaf, it stores the feature being split, the threshold value, and links to its left and right child nodes. If it is a leaf, it only holds the predicted value for that part of the feature space. The main class, 'DecisionTreeClass', builds a full decision tree for classification. It includes methods to train the tree to a specified depth, using either the Gini index or cross-entropy as the loss function, and to make predictions on test data.

We also created additional classes, 'RandomForest' and 'AdaBoost', to implement these statistical methods. Table 2 compares the accuracy of our tree-based implementations with those from scikit-learn. The results are similar, showing that our methods work as intended.

However, for the rest of the analysis, we used scikit-learn methods. This decision was based on computational efficiency. While our implementations achieved good accuracy, they took much longer to train, likely due to the recursive nature of the algorithms.

Table 2: Accuracy comparison for methods and our implementation

Method	**scikit-learn Accuracy**	**Our Implementation Accuracy**
Decision Tree Classifier	0.7857	0.7078
Random Forest	0.8831	0.9026
AdaBoost	0.8571	0.8961
Gradient Boosting(Xgboost)	1.0000	N/A

Table 2: Accuracy comparison between scikit-learn methods and our implementation.

The Feed-Forward Neural Network implementation was done in an object-oriented manner, creating both a Layer class and a Model class to be able to train the Neural Network through feed-forward and back-propagation functions. Finally, Logistic Regression with mini-batch Stochastic Gradient Descent was implemented.

3.3 Classification metrics

Classification metrics are used to assess the performance of a classification model by comparing its predictions with the actual labels. Below are some commonly used metrics:

1. Accuracy

Accuracy is the ratio of correctly predicted instances to the total number of instances:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

While accuracy is easy to understand, it may not always be a good metric, especially for imbalanced datasets.

2. Precision

Precision measures how many of the predicted positive instances are actually positive:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

It is particularly important in scenarios where false positives are costly.

3. Recall (Sensitivity)

Recall, or sensitivity, measures how many of the actual positive instances were correctly predicted:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

It is crucial in cases where false negatives are more problematic than false positives.

4. F1 Score

The F1 Score is the harmonic mean of precision and recall, providing a balance between the two:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

5. Confusion Matrix

The confusion matrix provides a complete picture of model performance by showing the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). A typical confusion matrix looks like this:

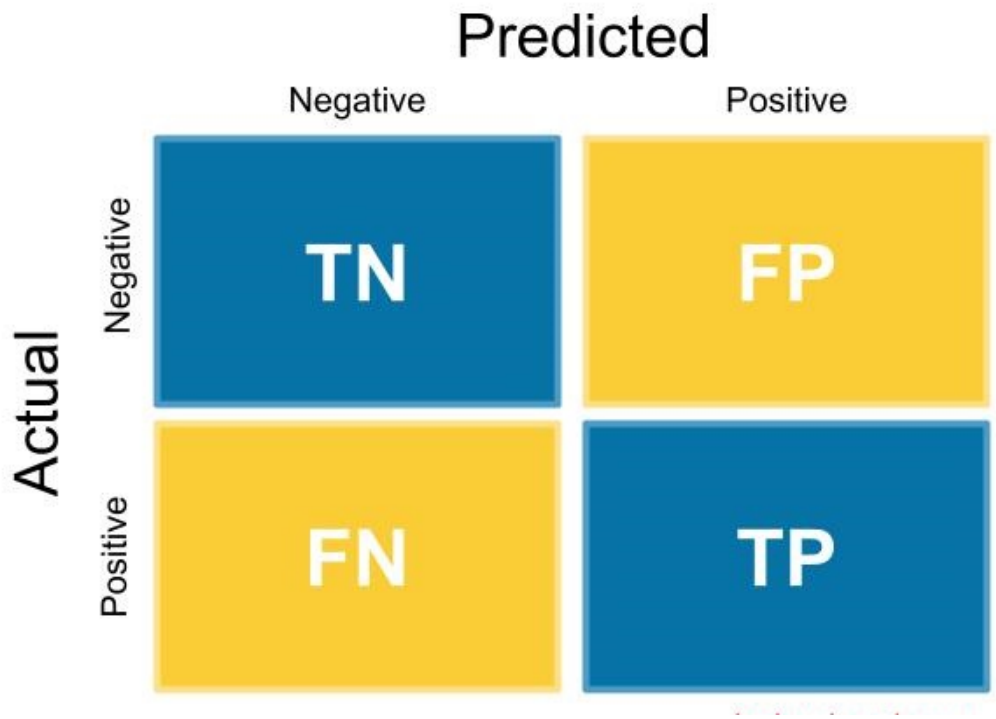

Figure 4: Example of confusion matrix

6. ROC-AUC Score

The Receiver Operating Characteristic (ROC) curve plots the true positive rate (TPR) against the false positive rate (FPR) at various thresholds. The Area Under the Curve (AUC) summarizes the model's ability to distinguish between classes.

3.4 Tree-based methods

3.4.1 Decision Tree Classifier

The first method we explored for classification was the Decision Tree classifier. One of the key parameters for this model is the depth of the tree, which controls its complexity.

To manage the depth, we built a complete tree and then applied cost-complexity pruning to simplify it. The results of this process are shown in Figure 5. For $\alpha = 0$, we used the full tree, which perfectly fits the training data (accuracy = 1). However, this model overfits and performs poorly on unseen data.

The best test accuracy was achieved with $\alpha = 0.006$, where the tree was pruned to balance complexity and generalization. The resulting pruned tree decision path is represented in Figure 6.

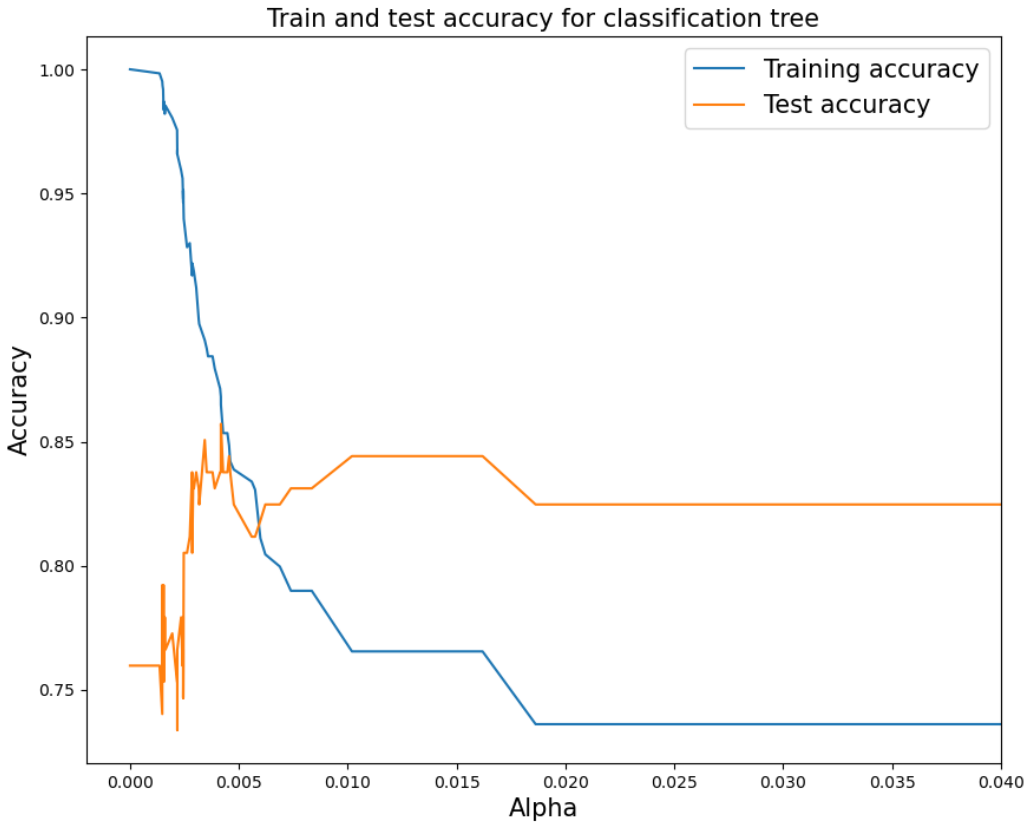

Figure 5: Accuracy on Training and Test Sets for Different Values of α.

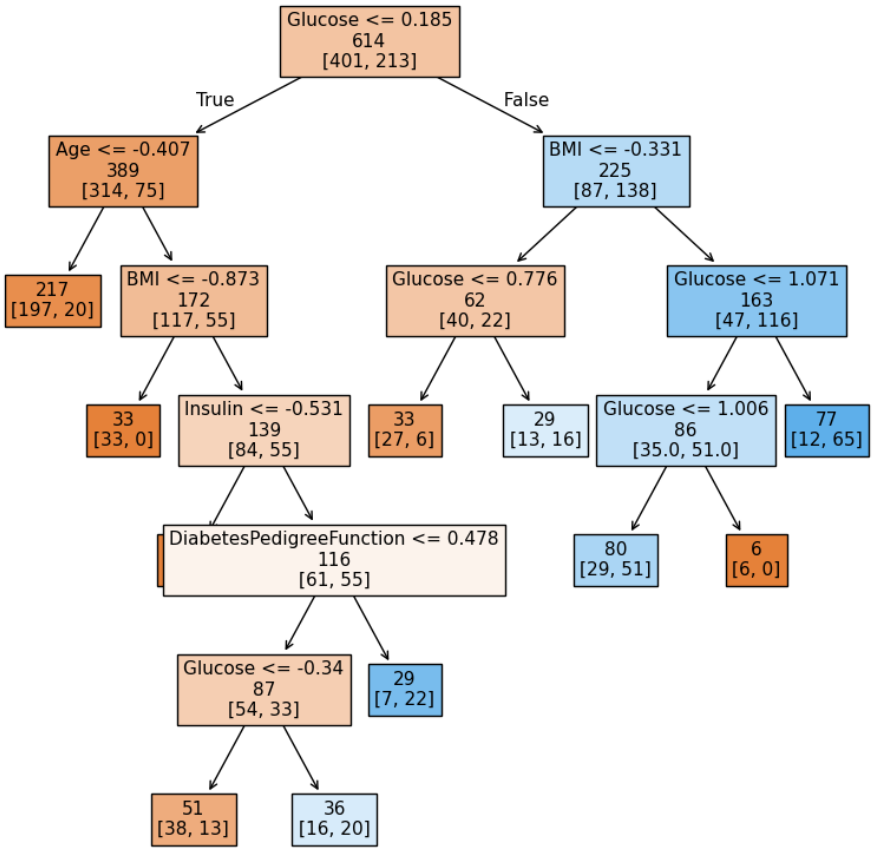

Figure 6: Decision path for the selected decision tree. The coloring visually represents the majority class in each node.

3.4.2 Random Forest Analysis

Next, we analyze the data using the Random Forest method. One of the most important parameters for this method is the number of predictors considered by each tree at each split. Increasing this parameter m results in higher model complexity and causes greater correlation between the trees in the ensemble.

For example, choosing $m = n$, where $n = 8$ is the total number of predictors, means selecting every variable in our model at each split, which introduces no randomization. In this case, the model becomes equivalent to bagging.

We proceed to tune the value of m. As shown in Figure 7, the best performance is obtained for $m = 2$, which corresponds to the recommended value for classification, $\lfloor \sqrt{8} \rfloor = 2$.

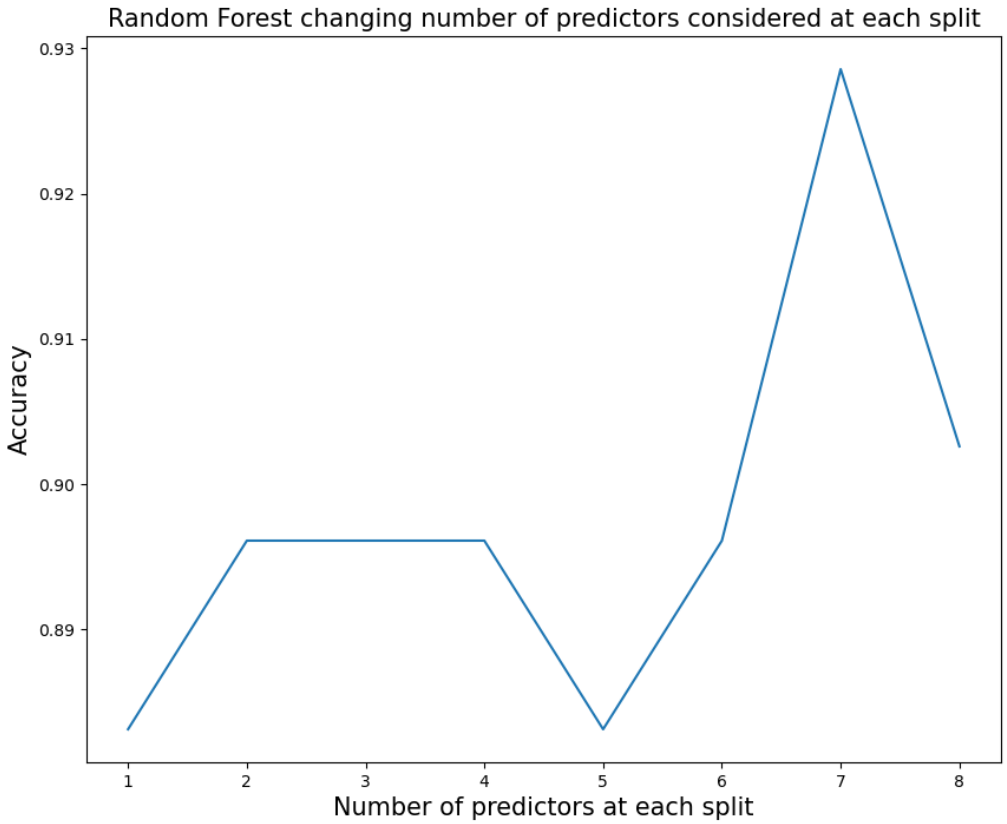

Figure 7: Test accuracy for Random Forest with changing number of predictors considered at each split.

XGBoost starts with high accuracy that tends to increase as the number of trees increases. It stabilizes at a high level, indicating that it effectively learns from the data and benefits from the additional trees due to its gradient boosting nature.

AdaBoost shows an initial increase in accuracy but levels off at a lower value compared to XGBoost. This suggests that while AdaBoost improves performance with more trees, it may not achieve the same level of accuracy as XGBoost in this scenario.

Random Forest exhibits a steady increase in accuracy and eventually stabilizes at a high level. It combines the strengths of multiple decision trees, benefiting from averaging predictions to reduce overfitting.

The **unpruned trees** serve as a baseline. If they show lower accuracy than the ensemble methods, it indicates that pruning and ensemble techniques are effectively managing overfitting and improving generalization.

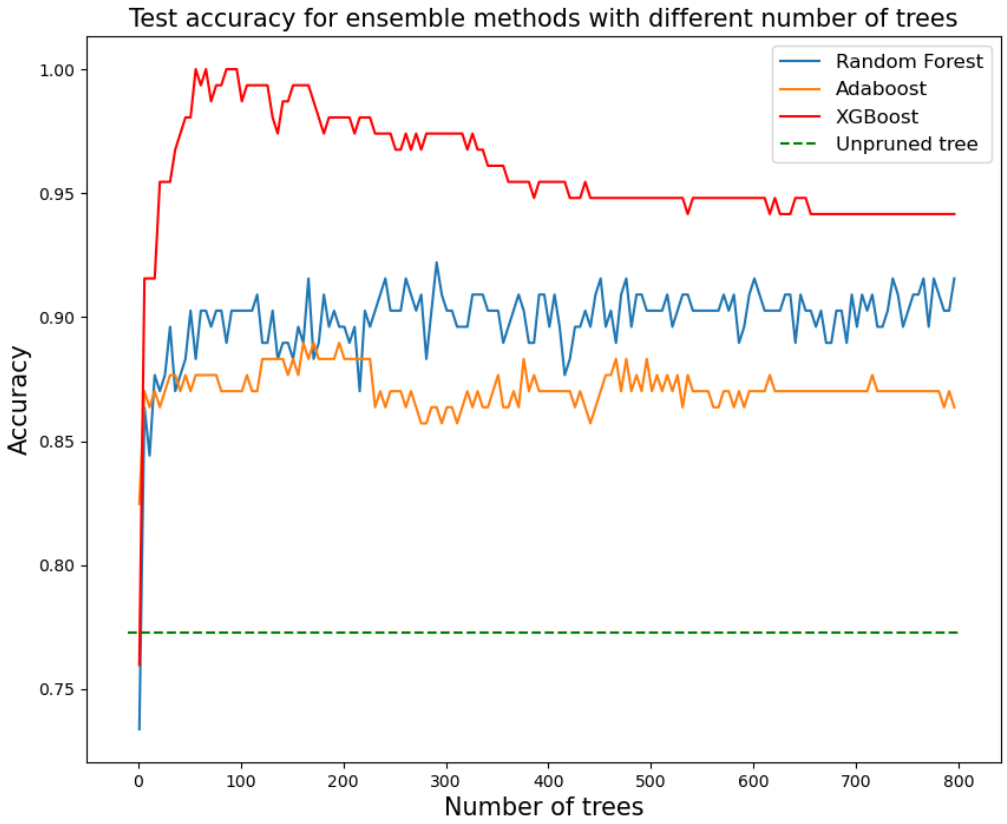

Figure 8: Comparing ensemble methods accuracy for increasing number of trees. The green dotted line shows the result for a single unpruned tree.

Figure 9 displays the ROC curves for the three classification methods we've examined so far. The curves are quite similar, and the AUC (Area Under the Curve) values are also comparable across all methods.

Table 3 summarizes the performance of these methods using various metrics. Interestingly, the pruned decision tree outperforms the other methods in all areas. Considering its better interpretability and portability, the decision tree seems to be the most suitable option for classifying our data at this stage.

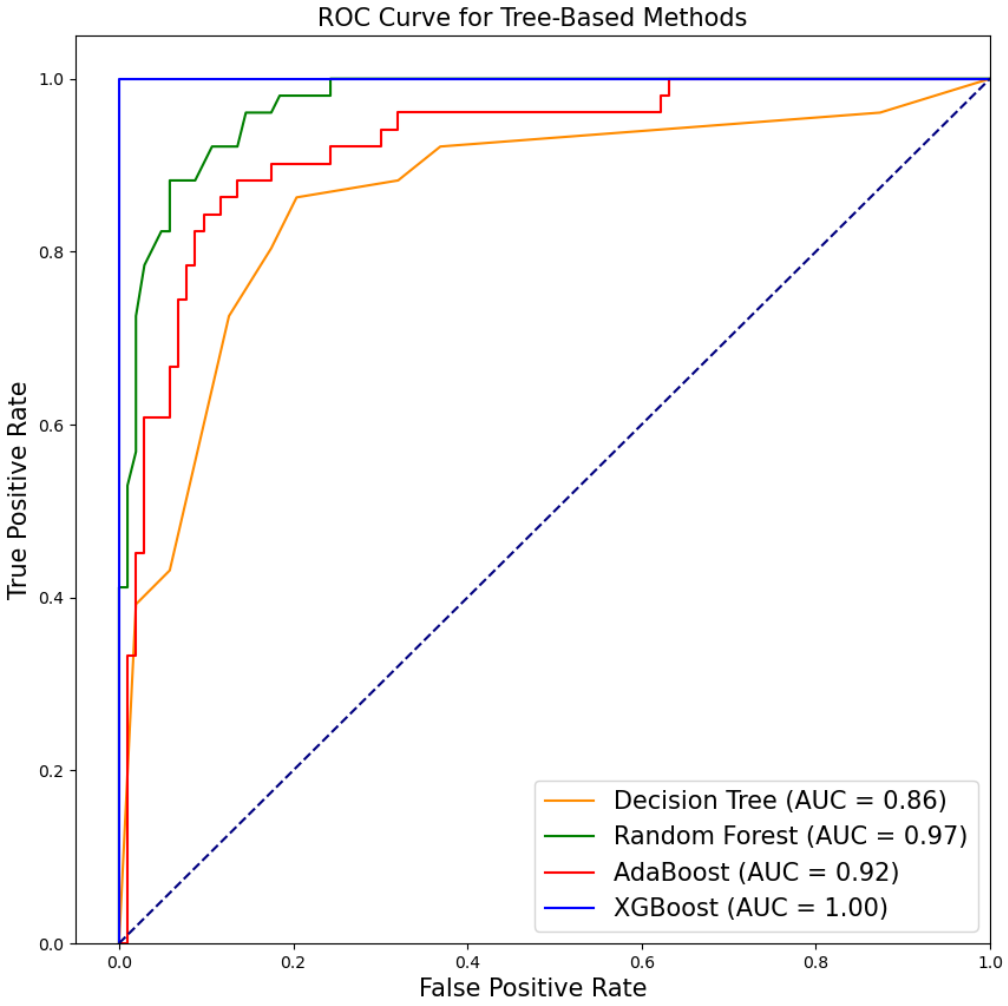

Figure 9: ROC curves plot comparison for Decision Tree, Random Forest, AdaBoost and XGBoost.

Table 3: Metrics comparison for tree-based methods.

Method	**Accuracy**	**Precision**	**Recall**	**Specificity**	**F-Score**	**AUC**
Decision Tree	0.818	0.677	0.863	0.796	0.759	0.865
Random Forest	0.916	0.880	0.863	0.942	0.871	0.972
AdaBoost	0.877	0.796	0.843	0.893	0.819	0.923
XGBoost	1.000	1.000	1.000	1.000	1.000	1.000

Table 3: Performance metrics for different classification methods.

3.4.3 Variable importance analysis

In this section, we analyse a possible variation on our model through variable selection. As shown in the Random Forest can be used naturally to obtain a measure of variable

importance. Figure 10 shows this analysis: as we also noticed when considering correlation between variables and outcome, Glucose and BMI are clearly the two crucial variables in determining the classification

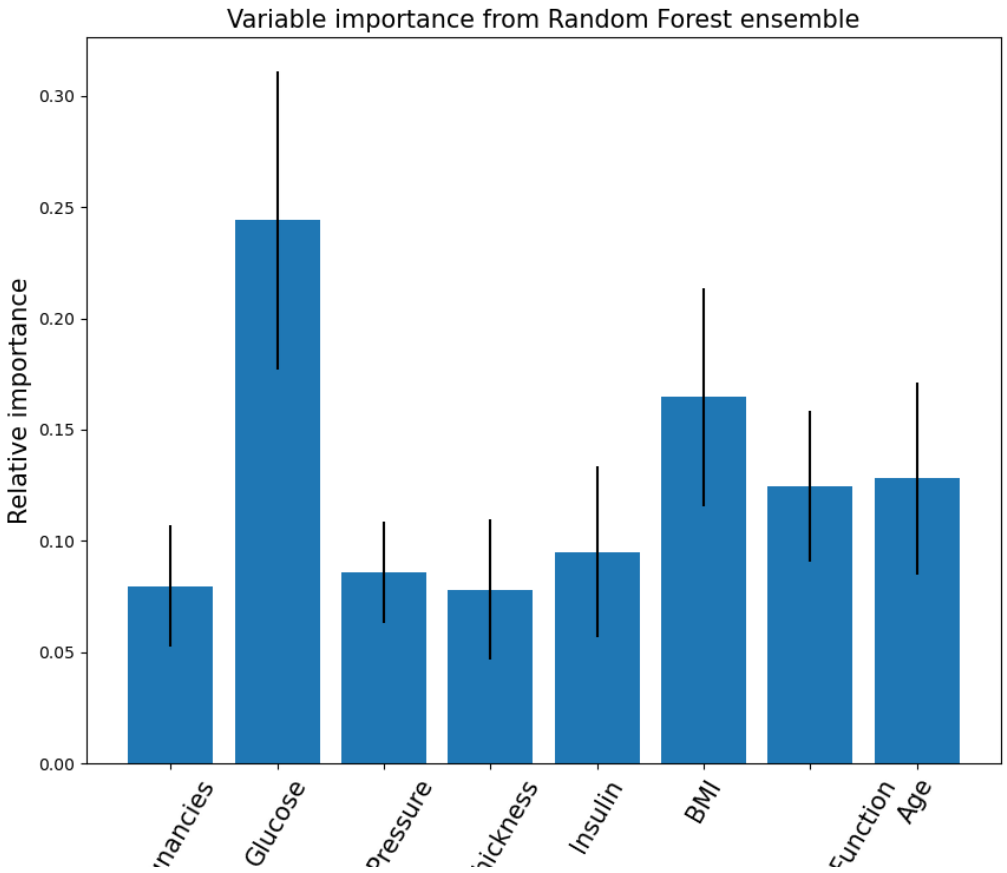

Figure 10: Relative importance plot of the 8 explanatory variables from a Random Forest ensemble.

3.4.4 Logistic Regression and Neural Networks

The heatmap provides a visual representation of accuracy across different combinations of learning rates and regularization parameters. The darkest or most intense colors indicate higher accuracy, suggesting optimal hyperparameter pairs. As seen, The best results are obtained for $\mu = 0.01$ and $\lambda = 0.01$. tends to yield better accuracy. This suggests that a flexible model is better suited to this dataset, but care must be taken to avoid overfitting.

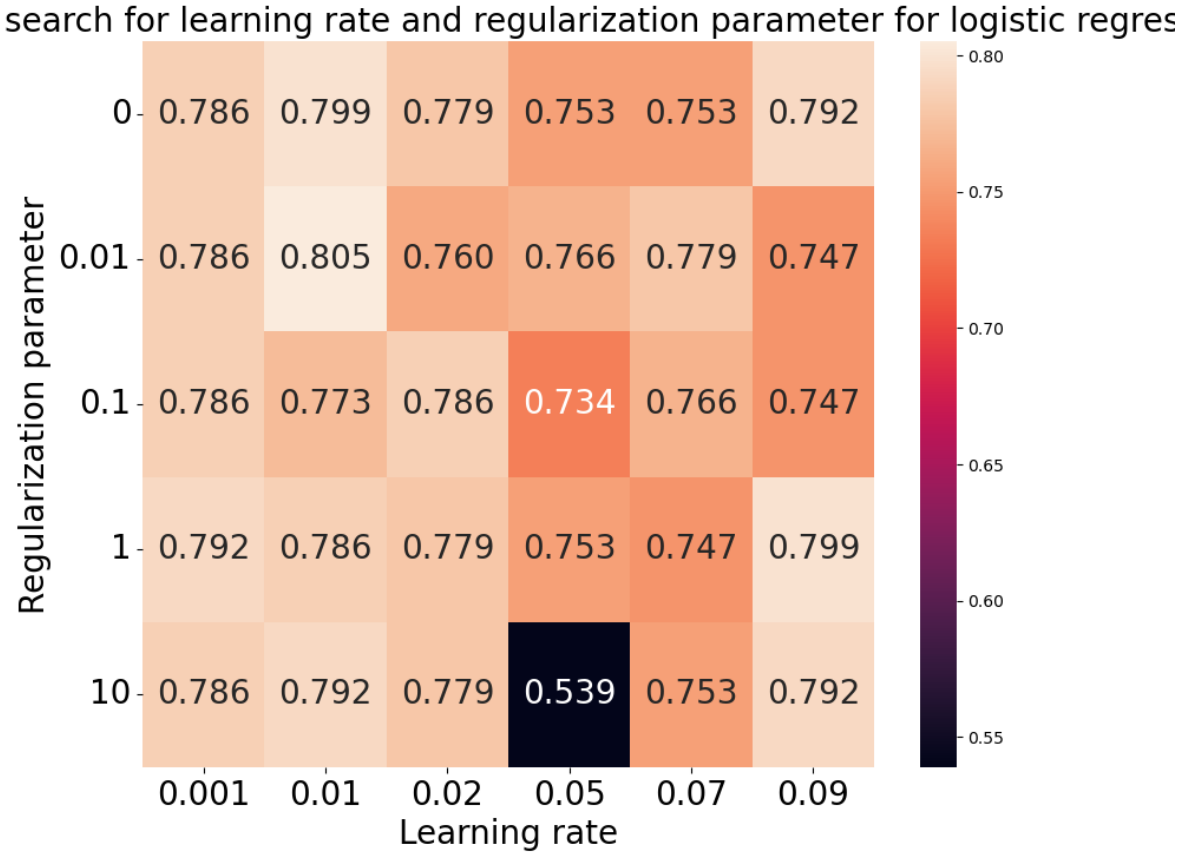

Figure 11: Grid search for the optimal accuracy as function of the learning rate and regularization parameter.

We also tried using a Feed-Forward Neural Network to fit our data. This network has adjustable layers, the number of neurons in each layer, and different activation functions. Since we are working with two classes (diabetes positive or negative), the output layer of the network has only one neuron with a sigmoid activation function. This means the output will be a number between 0 and 1, representing the probability that a data point belongs to class 1 (diabetes positive). We used cross-entropy as the cost function.

We tested different combinations of the number of neurons, layers, and activation functions. After tuning the model, we chose a final setup with two hidden layers, each containing 20 neurons. The activation function is sigmoid, and we used a learning rate of $\eta = 0.001$, with no regularization. The results of our tuning process are shown in figures 12, 13, 14, and 15.

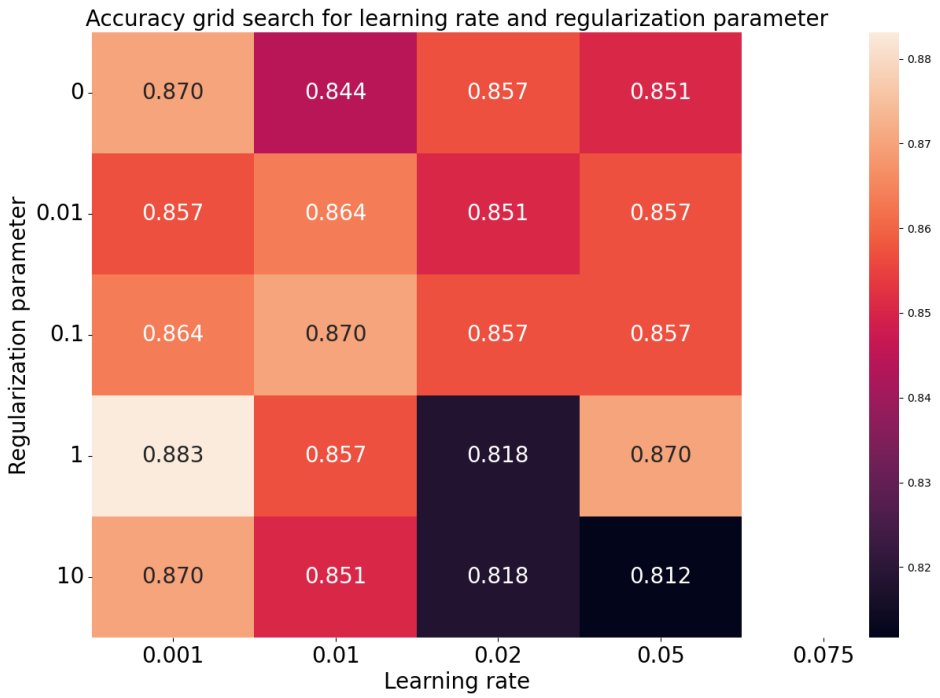

Figure 12: Grid search for the optimal accuracy as a function of the learning rate and regularization parameter. The chosen parameters are $\eta = 0.001$ and $\lambda = 0$.

Figure 13. the findings suggest that setting the learning rate to 0.05 and using 10 neurons results in optimal model performance. This highlights the importance of tuning hyperparameters in achieving the best outcomes in neural network training.

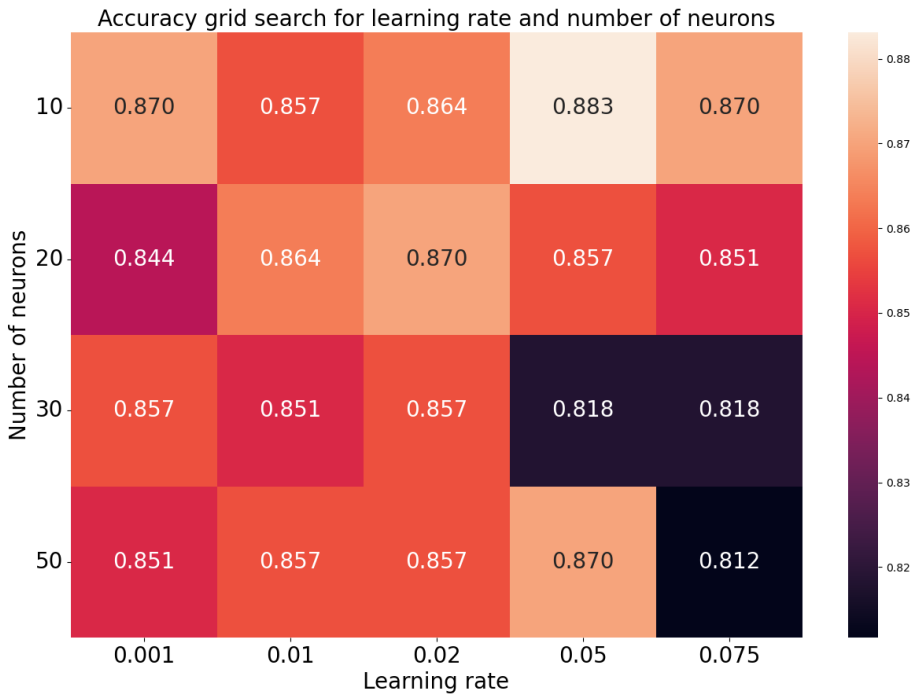

Figure 13: Grid search for the optimal accuracy as function of the learning rate and number of neurons in a single layer ($\eta = 0.05$ and number of neurons 10).

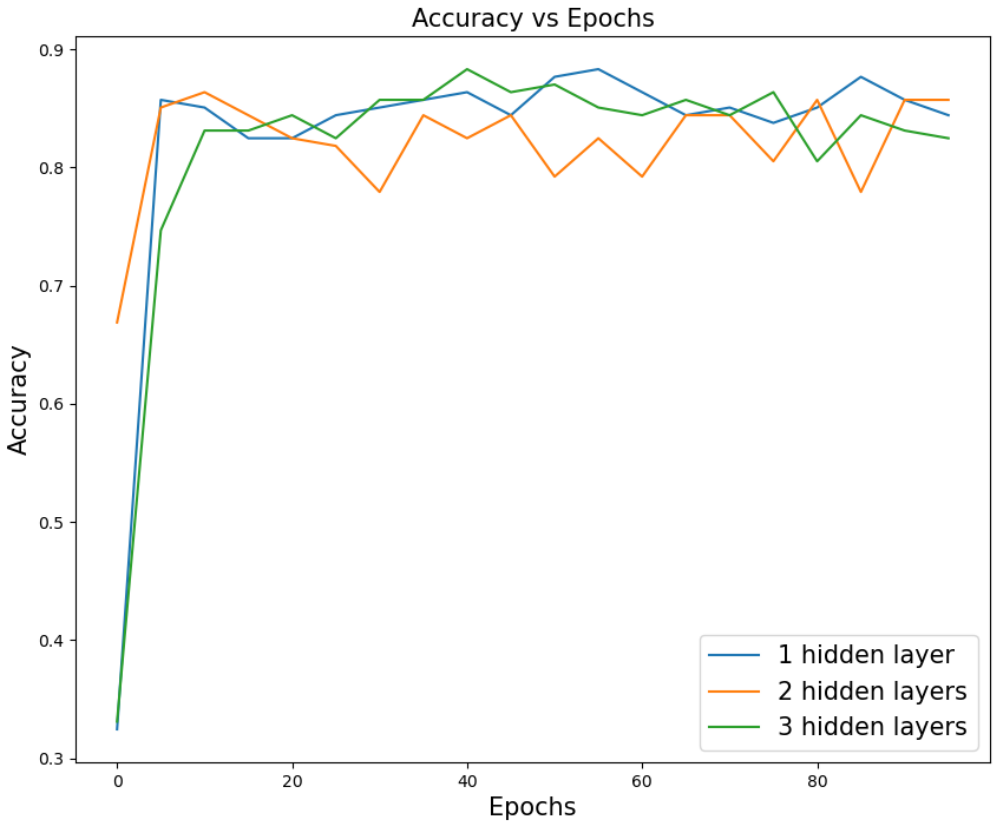

Figure 14: Test Accuracy vs. Epochs for Different Hidden Layer Configurations).

The 2 hidden layers configuration demonstrates the best performance in terms of test accuracy across epochs, making it the most effective choice for this model. The 1 hidden layer configuration is also robust, while the 3 hidden layers configuration does not provide a performance advantage and may indicate a need for careful tuning to avoid overfitting.

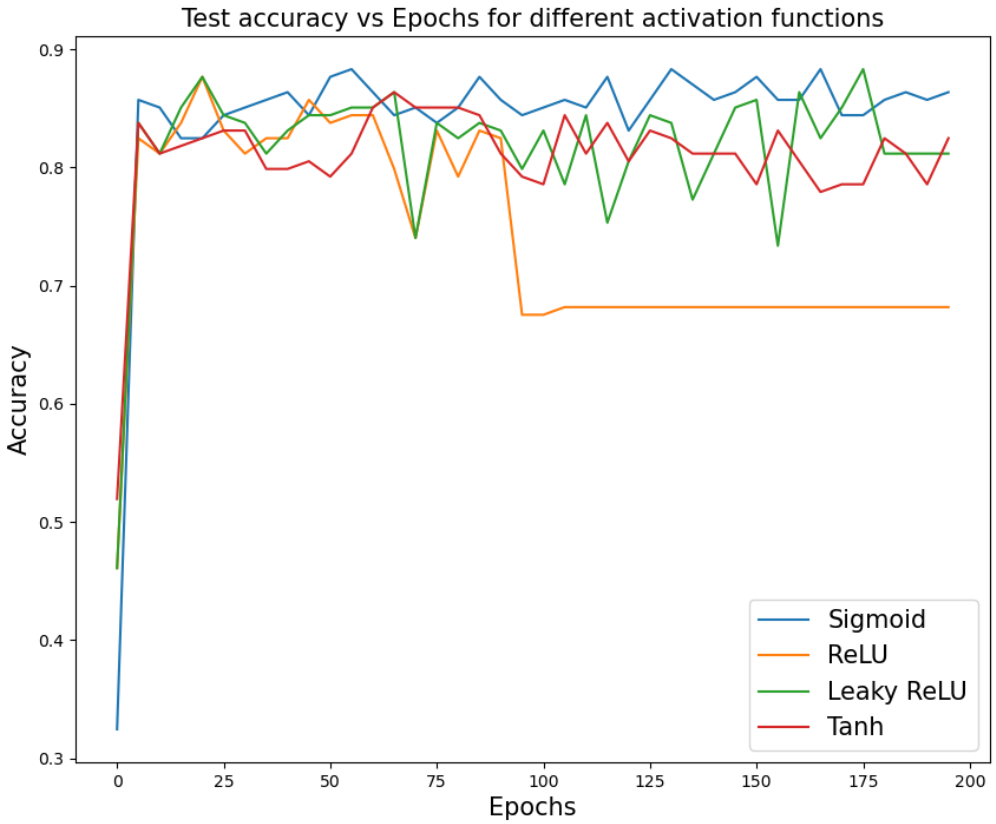

Figure 15: Analysis of Activation Functions on Test Accuracy).

In the figure 15,the Sigmoid activation function demonstrates the best performance in terms of test accuracy across epochs, making it the most effective choice for this model. Leaky ReLU follows as a viable alternative, while Tanh shows less effectiveness in this scenario. This analysis underscores the importance of selecting appropriate activation functions based on the specific characteristics of the dataset and model architecture.

Model	**Accuracy**	**Precision**	**Recall**	**Specificity**	**F-Score**	**AUC**
Decision Tree	0.818182	0.676923	0.862745	0.796117	0.758621	0.864839
Random Forest	0.915584	0.880000	0.862745	0.941748	0.871287	0.972111
AdaBoost	0.876623	0.796296	0.843137	0.893204	0.819048	0.922901
XGBoost	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
Logistic Regression	0.792208	0.630137	0.901961	0.737864	0.741935	0.929754
Neural Network	0.857143	0.822222	0.725490	0.922330	0.770833	0.922711

Table 4: Comparison of classification methods using different evaluation metrics.

Table 4 presents the performance of various classification methods across several metrics. XGBoost clearly stands out with a perfect score of 1.0000 across all metrics, including accuracy, precision, recall, specificity, F-Score, and AUC, indicating its

exceptional ability to correctly classify both classes.

In terms of accuracy, the pruned Decision Tree performs well with 81.8%, which is a solid result. However, XGBoost outperforms it significantly with an accuracy of 100

When considering precision, the Neural Network model excels with 82.22%, outperforming all other methods, indicating it is the most reliable when predicting positive cases. However, XGBoost still provides perfect precision.

For recall, Logistic Regression leads with 90.2%, demonstrating its strong ability to correctly identify positive cases (class 1), even though XGBoost provides perfect recall, capturing all true positives.

In terms of specificity, the Neural Network again performs the best with 92.23%, meaning it is most effective in identifying the negative class (class 0). This is a higher specificity than XGBoost and the other models.

The F-Score, which balances precision and recall, is best for XGBoost with a perfect score of 1.0000, but Logistic Regression comes close with 74.19

Finally, the AUC (Area Under the Curve), which assesses how well the model distinguishes between classes, is best for XGBoost, with a perfect score of 1.0000. This shows that XGBoost provides the most reliable separation between the classes.

While the pruned Decision Tree gives solid results, particularly in interpretability, XGBoost outperforms all other models across every metric. Logistic Regression, with its higher recall, could be preferred in situations where identifying positive cases is critical. However, considering all aspects, XGBoost offers the best overall performance in this study.

4 Conclusion

The aim of this project was to implement, study, and compare the performance of different statistical methods for classification on the Pima Indians Diabetes dataset. The methods were trained to predict whether a subject's diabetes mellitus diagnosis is positive based on 8 given attributes.

We began by implementing Decision Tree, Random Forest, AdaBoost classifiers and XGBoost and tested their performance compared to the corresponding scikit-learn methods. Additionally, we utilized previously implemented codes for a Feed-Forward Neural Network and logistic regression with mini-batch SGD.

Through our analysis, we also considered the possibility of restricting the number of variables in the model, selecting those deemed most crucial based on a variable importance analysis conducted with Random Forest. This showed that considering just the Glucose and BMI variables only slightly impacted the model's performance. Given that the full model only contains 8 variables and is still highly interpretable, we concluded that using the full model remains preferable.

It is important to highlight the noticeable imbalance between recall and specificity across the results of most methods. This discrepancy is likely due to the class imbalance in the dataset, where there are fewer positive diabetes cases compared to negative ones. As a result, many models tend to predict the negative class (0) more frequently, leading to high specificity (correctly identifying the negative cases) but lower recall (failing to identify enough of the positive cases).

In the context of medical screenings, recall is a crucial metric. We prefer predicting

false positives, which could be excluded with further medical testing, over false negatives. Given this observation, it could be argued that Logistic Regression is the best choice for our analysis, especially for identifying potential positive cases. However, for maximizing overall accuracy, XGBoost stands out as the most effective method.

Finally, while the Decision Tree offers the best interpretability, XGBoost's ability to achieve 100% accuracy suggests that it could be highly effective in practical applications where performance is paramount, and interpretability can be secondary.

For future work, addressing class imbalance, exploring additional features, refining hyperparameter tuning, and improving model interpretability are key areas for further research. Additionally, applying these models in real-world medical settings, particularly for diabetes detection, could provide valuable insights into their practical applications and effectiveness in clinical practice.

References

[1] World health organistion (2016) global report on diabetes, 2022. `https://www.who.int/publications/i/item/9789241565257`.

[2] V. Chang, J. Bailey, Q. A. Xu, et al. Pima indians diabetes mellitus classification based on machine learning (ml) algorithms. *Neural Computing and Applications*, 2022.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[4] Sheuly Debnath. Fys_stk4155 project 2, 2024. `https://github.com/Sheulydsp/FYS_STK4155`.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, New York, USA, 2001. Revised on Jan 13th 2017.

[6] Morten Hjorth-Jensen. Applied data analysis and machine learning, 2024. `https://github.com/CompPhysics/MachineLearning/blob/master/doc/pub/week46/ipynb/week46.ipynb`.

[7] Morten Hjorth-Jensen. Applied data analysis and machine learning, 2024. `https://github.com/CompPhysics/MachineLearning/blob/master/doc/pub/week47/ipynb/week47.ipynb`.

[8] L. O. Schulz, P. H. Bennett, E. Ravussin, J. R. Kidd, K. K. Kidd, J. Esparza, and M. E. Valencia. Effects of traditional and western environments on prevalence of type 2 diabetes in pima indians in mexico and the us. *Diabetes Care*, 29(8):1866–1871, 2006.