MSc Computer Engineering, Cybersecurity and Artificial Intelligence

University of Cagliari

Department of Electrical and Electronic Engineering

Predicting Diabetes: Random Forest vs MLP

Professors:

Ambra Demontis

Giorgio Fumera

Author:

Gianmarco Palmas (70/90/00685)

January 15, 2026

Contents

1 Introduction

This project focuses on comparing the effectiveness of two distinct machine learning approaches: Random Forest and Multi-Layer Perceptrons (MLP).

I use a Diabetes Dataset with clinical data collected from a hospital in Frankfurt, Germany. It contains measurements of patients including Pregnancies, Glucose, Blood Pressure, Skin Thickness, Insulin and other relevant factors. The goal is to use these traits to guess if a patient has diabetes or not.

The data is split into training (80%) and testing (20%) sets. Hyperparameters for Random Forest and MLP are optimized using 5-fold stratified cross-validation on the training set, while the final performance is evaluated on the held-out test set.

The performance of the models will be compared using metrics like accuracy, precision, recall, F1-score, and the ROC curve, highlighting the strengths and limitations of each approach.

2 Dataset Analysis & Preprocessing

This section provides an overview of the dataset used in the experiments. It includes a description of its structure, the features and target variable, and the overall class distribution. This provides the necessary context for the modeling and evaluation phases that follow.

2.1 Data Description

The dataset utilized in this project originates from clinical data collected in the Hospital of Frankfurt, Germany. It comprises a total of **2000 clinical samples**, each described by 8 numeric features and one binary target variable indicating the diagnostic status.

Regarding the data types, the dataset consists of a mix of integer and floating-point values:

- **Integer (`int64`)**: Used for discrete variables such as *Pregnancies*, *Glucose*, *BloodPressure*, *SkinThickness*, *Insulin*, *Age*, and the target *Outcome*.
- **Float (`float64`)**: Used for continuous variables, specifically *BMI* and *DiabetesPedigreeFunction*.

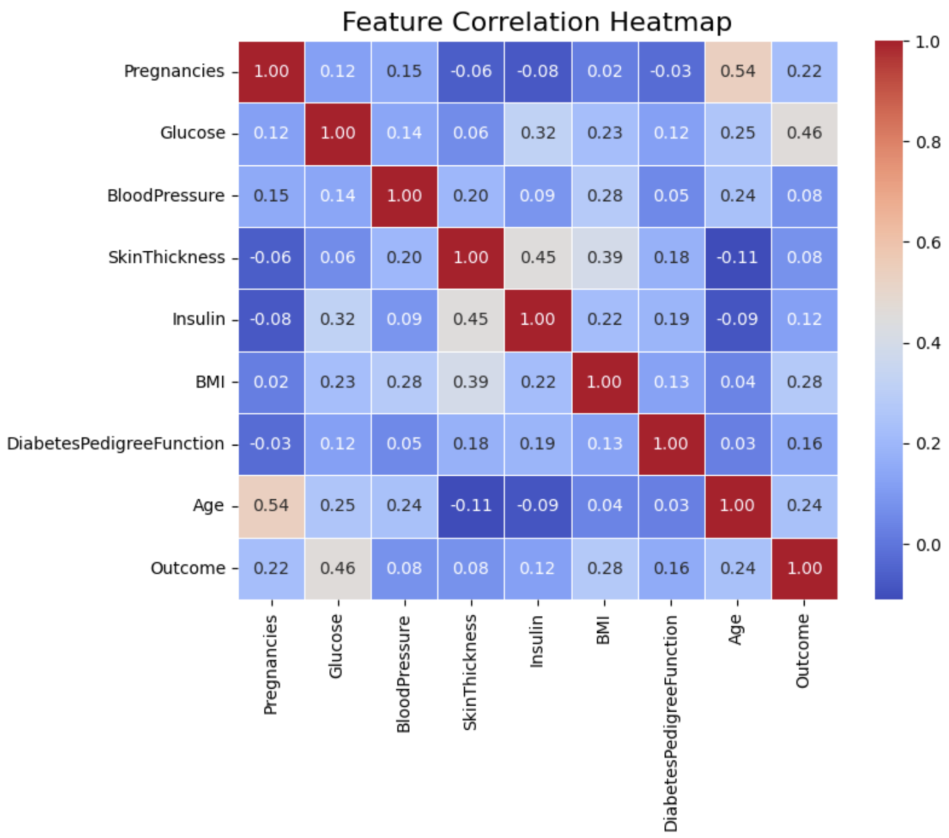
The features consist of:

- **Pregnancies**: Number of times pregnant
- **Glucose**: The amount of glucose in the plasma
- **BloodPressure**: The pressure of the bloodstream (mm Hg)
- **SkinThickness**: How thick the triceps skin fold is (mm)
- **Insulin**: Serum insulin level at 2 hours ($\mu U/ml$)
- **BMI**: Body mass index (kilograms divided by the height in meters squared)
- **DiabetesPedigreeFunction**: A function that evaluates the probability of diabetes based on familial history
- **Age**: Years of life

The outcome variable is binary: 0 for absence of diabetes and 1 for indicate diabetes.

This dataset was downloaded from Kaggle, where it is commonly used in machine learning tutorials and experiments.

As a key part of exploratory data analysis, I generated a correlation heatmap to understand the dataset's internal structure (Figure 1). It allows us to quickly identify the linear relationships between the different variables. It helps reveal how the features relate to each other.



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.00	0.12	0.15	-0.06	-0.08	0.02	-0.03	0.54	0.22
Glucose	0.12	1.00	0.14	0.06	0.32	0.23	0.12	0.25	0.46
BloodPressure	0.15	0.14	1.00	0.20	0.09	0.28	0.05	0.24	0.08
SkinThickness	-0.06	0.06	0.20	1.00	0.45	0.39	0.18	-0.11	0.08
Insulin	-0.08	0.32	0.09	0.45	1.00	0.22	0.19	-0.09	0.12
BMI	0.02	0.23	0.28	0.39	0.22	1.00	0.13	0.04	0.28
DiabetesPedigreeFunction	-0.03	0.12	0.05	0.18	0.19	0.13	1.00	0.03	0.16
Age	0.54	0.25	0.24	-0.11	-0.09	0.04	0.03	1.00	0.24
Outcome	0.22	0.46	0.08	0.08	0.12	0.28	0.16	0.24	1.00

Figure 1: Correlation heatmap between the features of the dataset. The more intense colors indicate a stronger correlation.

2.2 Data Preparation & Splitting Strategy

The data preparation phase was conducted in two fundamental steps to ensure the validity of the training.

First, the dataset was partitioned into two subsets: a **Training Set** (80% of the data) for Training the model and a **Test Set** (20% of the data) for the final evaluation. This division was performed using stratified sampling. (`stratify=y`) to ensure that the distribution of the target classes (diabetic vs. healthy patients) remained uniform between the two sets.

Subsequently, I applied a standardization to the numerical features to bring them on the same scale (mean 0 and variance 1), to help the convergence of the algorithms. For this operation, I used the scaler from the Scikit-Learn library.

```
scaler = StandardScaler()
```

It is crucial to note that the scaler was adapted (*fit*) exclusively on the Training Set and subsequently used to transform the Test Set, thus avoiding any form of *data leakage*.

3 Methodology

This section describes the implementation of the two classification models used in the study: a Random Forest classifier and a Multi-Layer Perceptron (MLP) neural network. Both models are trained on the same dataset using a consistent evaluation strategy to ensure a fair comparison. The following subsections outline the main characteristics of each model, as well as the training techniques and validation strategies applied during the experiments.

3.1 Random Forest Classifier

The main classification model adopted in this study is the **Random Forest**. This is a supervised learning method of the type *Ensemble*, which builds a multitude of decision trees during the training phase. To fully understand how this complex algorithm works, we must first analyze its fundamental component, the Decision Tree, and then the aggregation techniques that make the ”forest” superior to a single tree.

The Base Learner: The Decision Tree

The fundamental unit or *base learner* of the Random Forest is the **Decision Tree** (Figure 2). This model operates with flowchart-like logic, where each path from the root to a leaf represents a classification rule of the form **IF condition THEN class**.

The tree structure is composed of:

- **Root Node**: It represents the entire dataset and the starting point.
- **Internal Nodes**: They correspond to a **test** on an attribute (es. ”glucose level > 140?”). The algorithm operates in a *top-down* and recursive manner, searching at each step for the feature and threshold that best separate the data into homogeneous (”pure”) subsets.
- **Leaf Nodes**: These are the terminal nodes that do not perform further divisions and assign the final class label.

Despite their interpretability, individual decision trees are known to be **”Unstable Learner”** (High Variance Learners). They tend to suffer from *overfitting*, creating

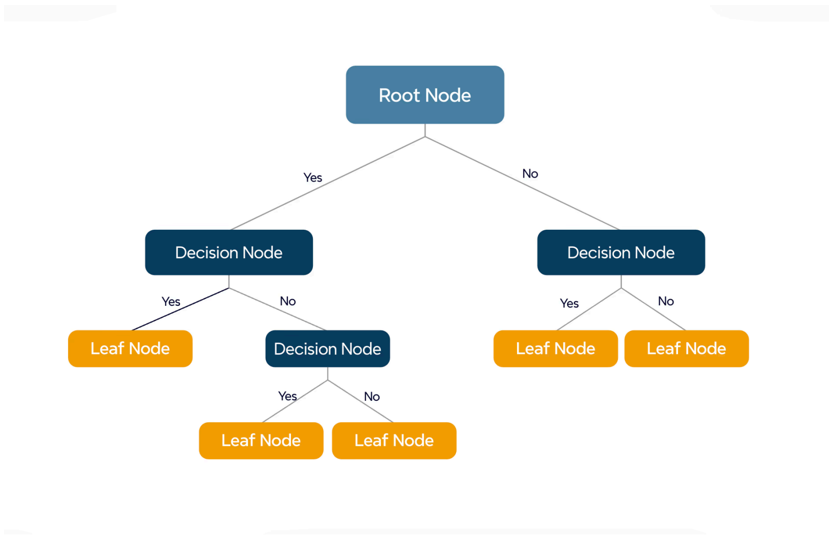


Figure 2: Generic diagram of a Decision Tree showing root node, internal nodes and leaves.

complex structures that adapt to the noise in the training data and that can change radically with small changes in the input.

Aggregation through Bagging (Bootstrap Aggregating)

To mitigate the high variance of individual trees, Random Forest uses a technique called **Bagging**. The central idea is that combining many unstable, independent models leads to a more robust overall model.

The process takes place in two phases:

1. **Bootstrap**: Given a training dataset D of size N, the algorithm generates B new datasets $(D_1, D_2, \ldots, D_B)$ by randomly drawing samples from D and *with reintroduction*.

2. **Aggregation**: An independent decision tree is trained on each dataset D_b. The final prediction is given by the **majority voting** of all the trees in the forest.

The Random Forest Algorithm

Bagging alone is not sufficient if the trees are too similar (correlated) to each other. As studies demonstrate, averaging correlated models does not effectively reduce the error.

Random Forest then introduces **Random Feature Selection**: during the construction of each tree, at each node, the model does not evaluate all the available features to decide the best split, but only considers a random subset m. This technique forces the trees to be structurally different ("decorrelated"), dramatically improving the ensemble's generalization ability.

Scikit-Learn Implementation

In this project I used the implementation of the **Scikit-Learn** library (`RandomForestClassifier`). It is essential to note a technical difference compared to the original theoretical algorithm: while the classic version uses the "Majority Vote" (*Hard Voting*), Scikit-Learn's implementation applies **Probability Averaging** (*Soft Voting*).

Instead of simply counting class votes (0 or 1), the model calculates the average of the class probabilities estimated by each individual tree. This approach ("averaging probabilistic prediction") tends to provide more accurate results and more precise ROC curves, since it takes into account each tree's level of confidence in its prediction.

3.2 Multi-Layer Perceptron Classifier

The **Multi-Layer Perceptron (MLP)** is a type of Artificial Neural Network (ANN), that belongs to the class of supervised learning models. While Decision Tree creates a set of explicit rules, the MLP is inspired (in a very abstract way) by the functioning of the human brain, building a model composed of interconnected basic computational units, called artificial neurons. In this project, this model has been implemented using the class `MLPClassifier` from the `sklearn.neural_network` library.

The fundamental building block of a neural network is the **Perceptron**. A single perceptron receives a set of input, computes a weighted sum of these inputs and applies an activation function to produce an output. However, a single perceptron is a linear classifier and can only solve linearly separable problems. It notoriously fails in simple but nonlinear problems, like the logical XOR problem.

To overcome this limitation, MLP models connect multiple perceptrons (or "units") in a layered structure, allowing the network to learn and model complex and nonlinear relationships.

Feed-Forward Architecture(FF-ML)

The architecture used in this project is a *feed-forward* Multi-Layer Perceptron (FF-ML). This is the most common neural network architecture for classification and regression problems. Its main features are:

- **Layers**: Neurons are organized into sequential layers:
    1. **Input Layer**: Receives the feature values (in our case, the 8 clinical measurements).
    2. **Hidden Layers**: One or more computational layers between the input and the output. This is where the network learns complex data representations.
    3. **Output Layer**: Produces the final result (in our case, the probability of diabetes).
- **Feed-Forward**: Information moves in only one direction, from the input, through the hidden layers, to the output. There are no backward or recurrent connections.
- **Fully-Connected**: In a standard MLP, each neuron in one layer is connected to *every* neuron in the next layer.

The expressive "capacity" of the network (i.e., its ability to model complex functions) depends on the number of hidden layers and the number of neurons in each of them.

In Figure 3 it is possible to see a graphical representation of a generic MLP.

Nonlinear Activation Functions

To allow the network to learn nonlinear relationships, the output of each neuron in the hidden layers is transformed by a nonlinear **activation function**. Unlike the step function (a simple 0/1 switch) of the original perceptron, modern MLPs use continuous and differentiable functions, which are essential to the training process.

The standard choice in modern networks, as well as the one used in this project (`activation='relu'`), is the **Rectified Linear Unit (ReLU)**.

Its function is $f(x) = \max(0, x)$. ReLU is computationally very efficient and helps maintain a robust gradient flow during training, making the training process more stable and faster, especially in networks with multiple layers.

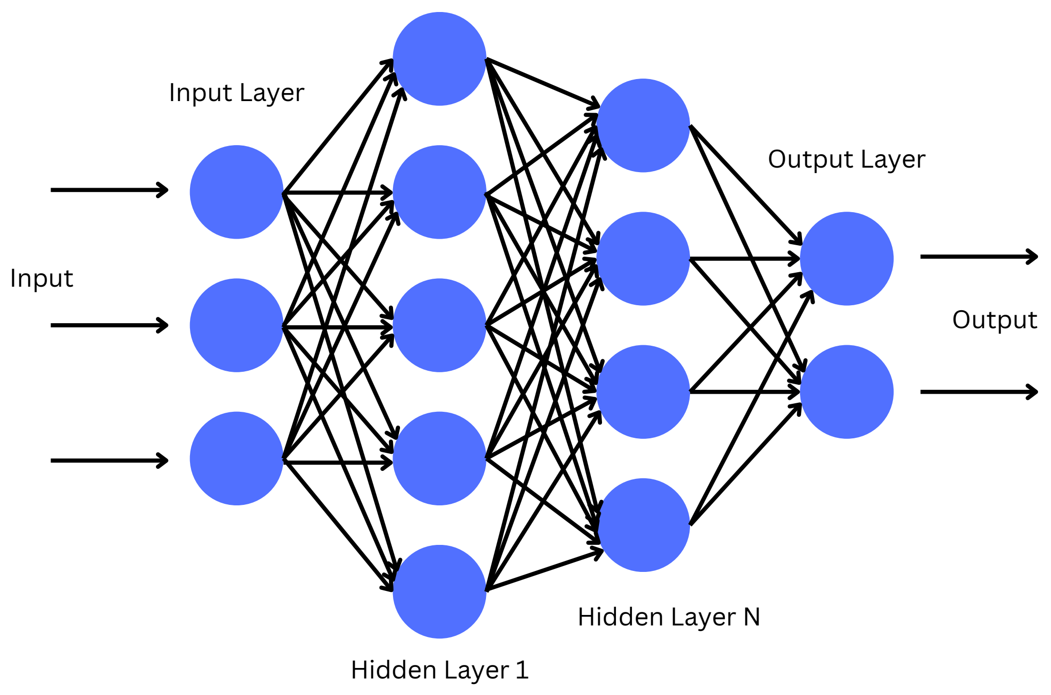


Figure 3: Generic architecture of a Multi-Layer Perceptron (MLP)

Training using Back-propagation

The goal of training is to find the set of **weights** (the ”strengths” of the connections between neurons) that minimizes the model’s error on the training data. This process is managed by an algorithm called **Back-propagation**.

The algorithm runs in two phases for each training sample:

1. **Forward Propagation**: The input (e.g., the 8 features of a patient) is passed through the network, layer by layer. In each neuron, the weighted sum of the inputs is calculated and the activation function (ReLU) is applied. This process continues until the output layer, where a prediction is obtained.

2. **Error Calculation**: The prediction is compared to the actual value (the label 0 or 1) using a **Loss Function**, such as *Cross-Entropy*. The error indicates how much the network ”got wrong.”

3. **Backward Propagation**: This is the heart of the learning process. The algorithm calculates the error **gradient**. The gradient is a mathematical concept (a

vector) that indicates the direction of maximum slope of the error function; essentially, it tells us the steepest ”path” to take to reduce the error. The algorithm then calculates how much the total error is affected by each individual weight in the network, using the *chain rule* of differential calculus and propagating the error ”backwards” from the output to the input.

4. **Weight Update**: The weights are updated slightly in the opposite direction of the gradient (a process called **gradient descent**) to reduce error.

3.3 Hyperparameter Optimization

Hyperparameters are configuration parameters external to the model, whose values are not learned directly from the data during training but must be set a priori. Their choice is crucial because they directly influence the model’s complexity, learning speed, and, above all, its generalization ability. A careful configuration is essential to find the right balance (trade-off) between:

- **Underfitting**: The model is too simple and fails to capture underlying patterns in the data.
- **Overfitting**: The model is too complex and ends up ”memorizing” the training data, including the noise, and then fails to correctly predict new data.

To find the optimal values, instead of relying on manual trials or common practices, a systematic search was performed using the `GridSearchCV` class of the **Scikit-Learn** library.

```
GridSearchCV(model, params, cv=kf, scoring='roc_auc', n_jobs=-1)
```

This method performs an exhaustive (”grid”) search by testing all possible combinations of parameters specified by the user.

3.3.1 Validation using Stratified K-Fold

Within the Grid Search process, the **K-Fold Cross-Validation** technique was used to evaluate the quality of each single combination of hyperparameters without risking overfitting on the training set.

```
kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

The procedure divides the training set into k equal-sized subsets called ”folds.” The model is trained k times: in each iteration, one fold is used as the validation set, while the remaining $k-1$ folds are used for training. The final performance of the combination is the average of the scores obtained in the k rounds.

Specifically, in this project I used a **Stratified K-Fold Cross-Validation** with $k = 5$ (Figure 4). The ”stratified” aspect is crucial given that our dataset has a class imbalance (approximately 65% negative and 35% positive). Stratification ensures that each fold maintains exactly the same proportion of classes as the full dataset. This prevents scenarios where a fold might, by pure chance, contain very few examples of diabetic patients, leading to a biased evaluation.

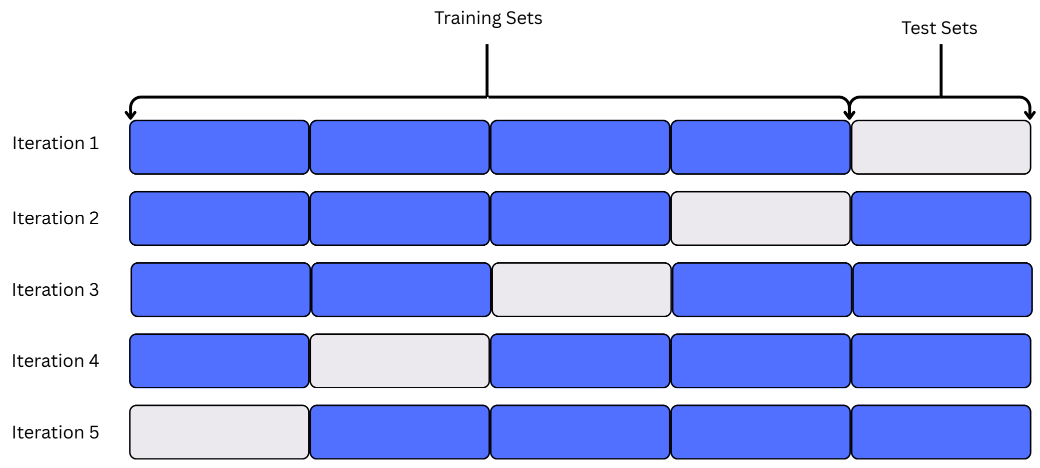


Figure 4: Schematic of the K-Fold Cross-Validation with $k = 5$, used within the Grid Search.

It is important to note that as a scoring metric to select the best parameters, I chose the **AUC (Area Under the Curve) ROC**, instead of simple accuracy. Given the unbalanced nature of the medical problem, the AUC provides a more reliable measure of the model’s ability to distinguish between classes.

After completing the optimization process on the training set, the models configured with the best combination of hyperparameters were saved for final evaluation. The following tables show the best hyperparameters identified for each model.

Random Forest Classifier

Hyperparameter	Description and Justification
`n_estimators=200`	Number of trees in the forest. The research selected 200 estimators, indicating that a larger ensemble is needed on this dataset to stabilize prediction variance and ensure robust performance across the validation folds.
`criterion='gini'`	A function to measure the quality of the split, is generally computationally faster than entropy.
`max_depth=20`	Maximum tree depth. The optimal value of 20 allows the model to capture deep and complex non-linear relationships between the features.
`min_samples_split=2`	Minimum number of samples needed to split a node. A value of 2 allows the trees to grow fully and drill down into fine-grained details of the data distribution.
`class_weight='balanced'`	(Fixed parameter) Assigns weights to classes inversely proportional to their frequency. Essential for managing the dataset imbalance between diabetic and non-diabetic cases.

Table 1: Optimal hyperparameters for the Random Forest model identified using GridSearchCV.

Multi-Layer Perceptron (MLP)

For both models, the `random_state=42` hyperparameter was set to ensure the reproducibility of the experiments, allowing anyone running the code to obtain the exact same results.

Hyperparameter	**Description and Justification**
`hidden_layer_sizes=(100, 50)`	Network architecture. The research identified a larger two-layer hidden structure (100 and 50 neurons) as optimal, providing sufficient capacity to model the interactions between the 8 clinical features without excessive complexity.
`learning_rate_init=0.01`	A value of 0.01 indicates that the algorithm benefits from larger update steps to quickly converge towards the global minimum of the loss function.
`alpha=0.01`	A value of 0.01 was selected to penalize the weights, preventing the model from becoming too dependent on specific patterns and reducing the risk of overfitting on the training data.
`activation='relu'`	(Fixed parameter) ReLU activation function, a modern standard for deep networks that avoids the vanishing gradient problem.
`solver='adam'`	(Fixed parameter) Efficient optimizer, chosen for its robustness and speed of convergence on relatively large datasets.
`early_stopping=True`	(Fixed parameter) Stops training if validation performance does not improve for a set number of epochs, saving computational resources.

Table 2: Optimal hyperparameters for the MLP model identified using GridSearchCV.

4 Evaluation and Results

The models are evaluated and their performance compared using multiple metrics. Assessment is performed through accuracy, precision, recall, F1-score, and ROC-AUC, providing a comprehensive view of the predictive capabilities. The following subsections present a detailed analysis of the results, including performance metrics, confusion matrices, and ROC curves with AUC, highlighting the strengths and limitations of each model.

4.1 Performance Metrics

To evaluate and compare the performance of Random Forest(RF) and Multi-Layer Perceptron (MLP) models, several standard metrics for binary classification were used. Each metric offers a different perspective on the model's effectiveness. The performance of the two models can be seen in the image 5.

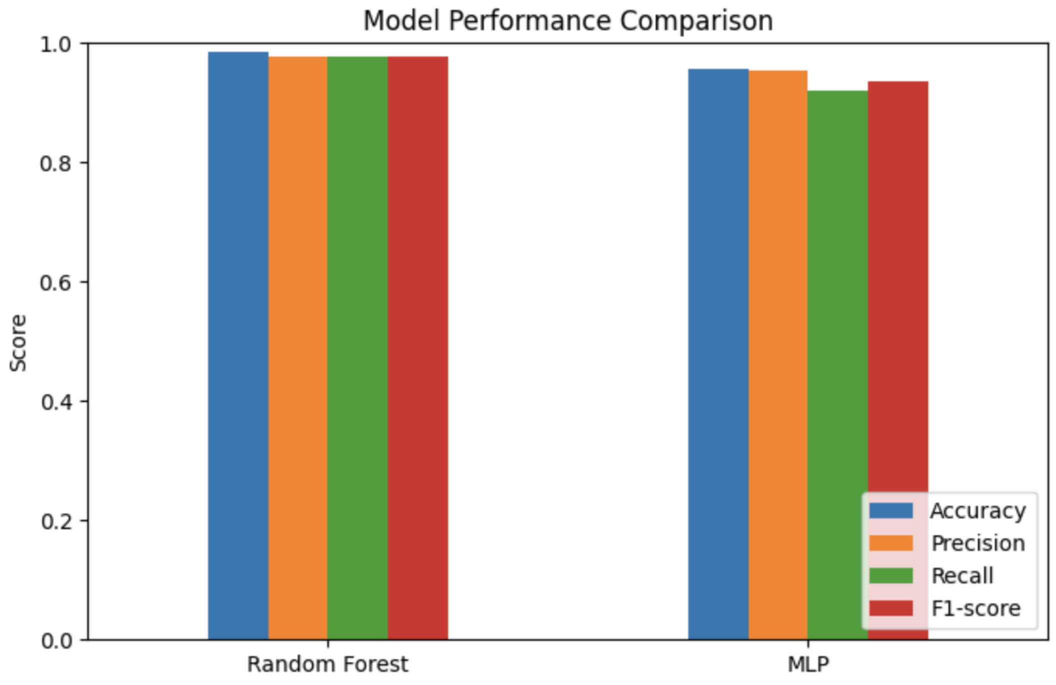


Figure 5: Performance of the models.

Accuracy

Accuracy is the most intuitive metric and represents the proportion of correct predictions (both positive and negative) compared to the total number of predictions made. It is calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where:

- TP (True Positives): Positive cases predicted correctly.
- TN (True Negatives): Negative cases predicted correctly.
- FP (False Positives): Negative cases incorrectly predicted as positive (False Alarms).
- FN (False Negatives): Positive cases incorrectly predicted as negative (Missed Diagnoses).

Accuracy provides a general view of performance, but can be misleading with unbalanced datasets.

Precision

Precision measures the proportion of positive predictions that were actually correct. It answers the question: "Of all the patients the model identified as diabetic, how many actually had diabetes?" It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

High precision indicates that the model generates few false positives (false alarms). This is especially important when the cost of a false positive is high.

Recall (Sensitivity or True Positive Rate)

Recall measures the proportion of actual positive cases that the model was able to correctly identify. It answers the question: "Of all the patients who actually had diabetes, how many were correctly identified by the model?" It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

A high recall indicates that the model generates few false negatives (missed diagnoses). This is crucial in contexts like the medical field, where missing a positive case (e.g., a diabetic patient) can have serious consequences.

F1-Score

The F1 score is the harmonic mean of precision and recall. It provides a single score that balances both metrics. It is particularly useful when seeking a compromise between minimizing false positives and minimizing false negatives, especially in the presence of unbalanced classes. It is calculated as:

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

A high F1-Score indicates that the model performs well in terms of both precision and recall.

Analysis of Results

The results obtained for the two models are summarized in the following table:

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
Random Forest	0.9850	0.9781	0.9780	0.9782
MLP	0.9575	0.9545	0.9197	0.9368

A quantitative analysis of performance shows that both models achieved excellent levels of accuracy, with an overall accuracy above 96%. Specifically, the **Random Forest** emerged as the unequivocally superior model across all evaluated metrics.

The most significant comparison occurs when observing the individual components of Precision and Recall:

- **Precision**: The Random Forest achieved a higher precision (**97.81%**) compared to the MLP (**95.45%**). This indicates that the Random Forest is more robust in minimizing false positives, meaning that when it predicts diabetes, it is highly likely to be correct.
- **Recall**: This is the most critical difference for a medical diagnostic tool. The Random Forest reached a Recall of **97.81%**, significantly higher than the MLP's

91.97%. This gap of nearly 6 percentage points means the Random Forest is much more effective at identifying actual diabetic patients, minimizing the dangerous risk of false negatives that the MLP might miss.

4.2 Confusion Matrix Analysis

The **Confusion Matrix** is a visual tool that compares real labels with those predicted by the model, allowing you to analyze the typology of errors made. The four cells of the matrix represent True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN), as defined in the previous section.

The confusion matrices obtained for the Random Forest (RF) and Multi-Layer Perceptron (MLP) models are shown in Figure 6.

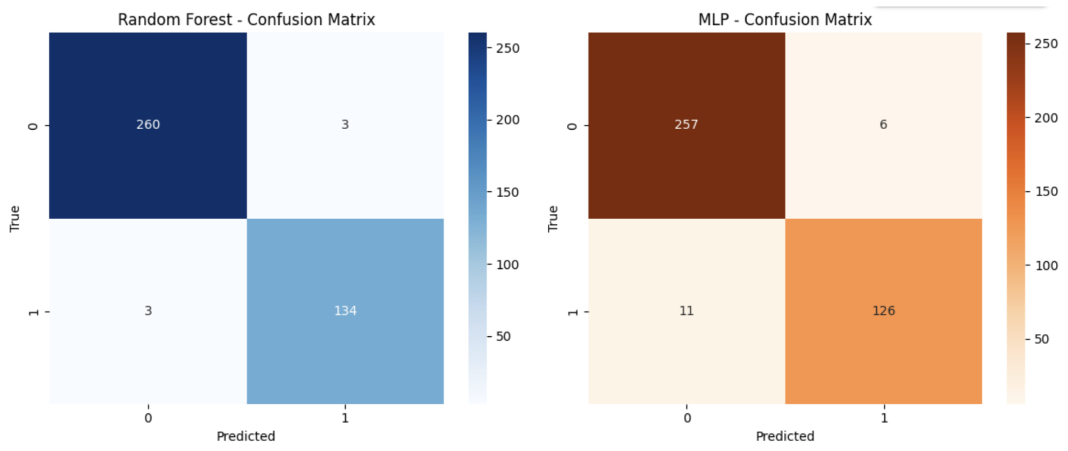


Figure 6: Confusion matrices for Random Forest (left) and MLP (right).

A visual comparison of the two matrices highlights:

- **Diabetic Patients**: The Random Forest excels in identifying sick patients. This is reflected by a higher density of True Positives and a minimal number of False Negatives compared to the MLP, meaning that critical diagnoses are rarely missed.
- **Healthy Patients**: The Random Forest also manages healthy cases better than the MLP. It generates fewer False Positives (”false alarms”), demonstrating a more stable decision boundary.

In conclusion, the visual analysis tell us that the Random Forest provides a more reliable clinical tool.

4.3 ROC Curve and AUC

The **Receiver Operating Characteristic (ROC)** curve shows the model's ability to distinguish between two classes (diabetics vs. healthy) by plotting the **True Positive Rate (TPR)** against the **False Positive Rate (FPR)**. The TPR (or Recall, $TPR = TP/(TP + FN)$) measures how many diabetics are correctly identified, while the FPR ($FPR = FP/(FP + TN)$) indicates the rate of "false alarms" (healthy patients mislabeled). A **perfect** model reaches the upper left corner (100% TPR, 0% FPR), whereas a **useless** model follows the diagonal line.

The **Area Under the Curve (AUC)** quantifies this performance into a single number, representing the probability that the classifier ranks a random positive instance higher than a negative one. AUC values range from 0 to 1: a value of 0.5 indicates no discriminant ability (random prediction), while values closer to 1.0 signify better separability, with 1.0 representing a perfect classifier.

Analysis of Results

The ROC curves (Figure 7) visually confirm the high discriminative power of both models, but highlight a distinct advantage for the tree-based approach.

The **Random Forest** achieves a near-perfect Area Under the Curve (AUC), outperforming the **MLP**, which reaches an AUC of **0.98**. While both scores represent excellent diagnostic capability in a clinical setting, the Random Forest curve remains closer to the upper-left vertex, demonstrating a superior sensitivity.

4.4 MLP Loss Curve Analysis

Figure 8 shows the progression of the Loss Function during the training phase. The curve shows an initial decline, a direct consequence of the optimized learning rate ($\eta = 0.01$), followed by a stabilization phase where the model fine-tunes its weights.

The activation of the **Early Stopping** mechanism is clearly visible: the training process terminated automatically around the **70th epoch** (significantly earlier than the

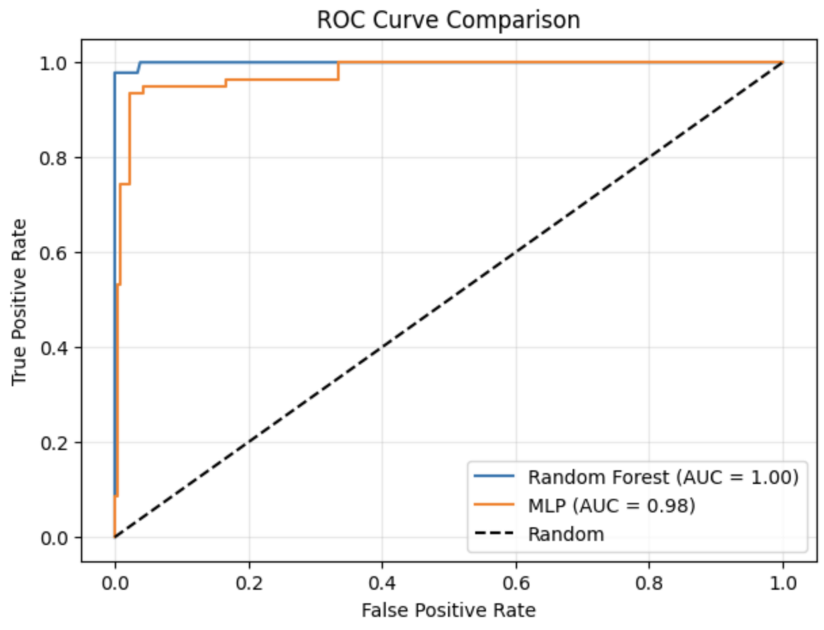


Figure 7: Comparison of ROC curves between Random Forest and MLP models.

maximum allowed limit of 1000). This demonstrates that the network converged to a stable solution relatively quickly.

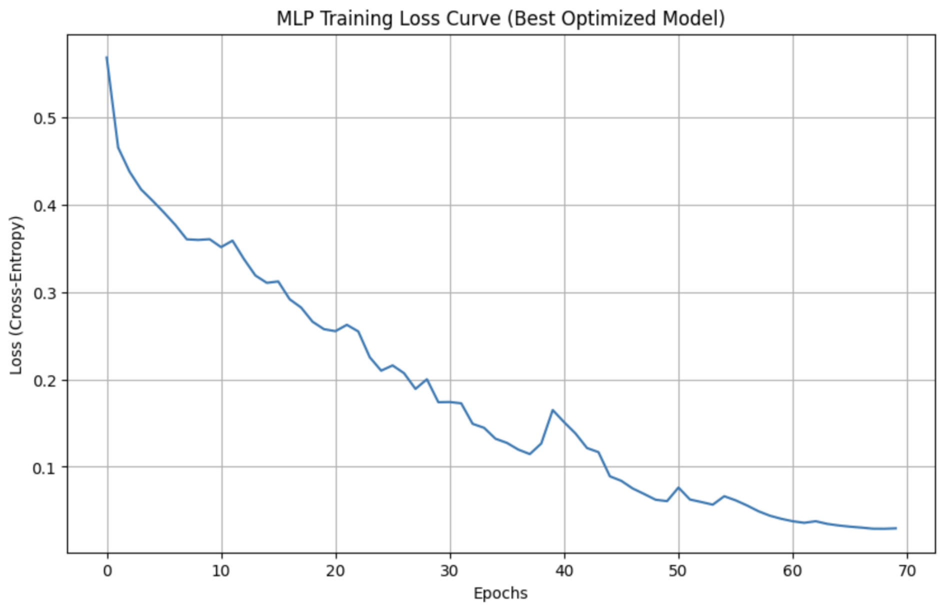


Figure 8: Training curve (Loss) of the MLP model.

5 Conclusion

This project analyzed and compared the performance of two machine learning models, **Random Forest** and **Multi-Layer Perceptron (MLP)**, for diabetes prediction.

The experimental results demonstrate the high effectiveness of both approaches, with overall accuracy exceeding **95%**. However, the comparative analysis identifies the **Random Forest** as the superior model for this diagnostic task.

- **Overall Dominance**: The Random Forest achieved the highest scores across all metrics, reaching an **Accuracy of 98.50%** and an **AUC of 0.99**.
- **Clinical Safety (Recall)**: Most importantly for medical screening, the Random Forest excelled in Sensitivity (Recall), reaching **97.81%**. This means it successfully identifies nearly all positive cases, significantly minimizing the risk of false negatives compared to the MLP.
- **Reliability**: The model also maintained a high **Precision**, ensuring that positive predictions are accurate and reducing false alarms.

In summary, while the MLP performed well, the **Random Forest** proves to be the most robust and safe choice for this specific clinical application, offering the best balance between diagnostic power and reliability.