SUPERVISED MACHINE LEARNING

JAIME CASTRO CERNADAS

Machine Learning | MSc. In Computational Biology

Index

1. Preprocessing

In this dataset, the only preprocessing step applied was the removing of the “Unnamed: 0 “ feature. This feature was a row index, and it was eliminated to further clean the data and eliminate any possible interference with this unnecessary data.

No other preprocessing methods were applied, as the data had no missing values, no duplicates and no need for scaling or normalizing.

2. Algorithm evaluation pipeline

A pipeline was created for the model selection, training-test prediction and confusion matrix generation.

2.1 Hyperparameter tunning and training evaluation

GridSearchCV was used to identify the best parameter combination for the models. GridSearchCV is a systematic approach to hyperparameter tuning that evaluates all possible combinations of specified hyperparameter values to determine the combination that optimizes a performance metric. It works by splitting the training data into k-folds, training the model on k-1 folds, and validating it on the remaining fold for each combination of hyperparameters. The process is repeated for all folds, and the average validation score is computed for each hyperparameter combination. The combination with the highest score is then selected as the best set of hyperparameters (1)

Due to the small-sized dataset, the Leave-One-Out Cross-Validation (LOOCV) method was chosen. LOOCV is a cross-validation technique where each data point in the training set is used once as a validation set while the remaining points form the training set. Since LOOCV uses each data point as a validation set exactly once, it avoids overfitting to any specific subset of the training data. This is especially important for small datasets, where traditional train-validation splits may result in an insufficient number of validation samples. Additionally, LOOCV provides a more comprehensive understanding of how the model performs across all data points, ensuring that the evaluation reflects the entire dataset's characteristics.(2)

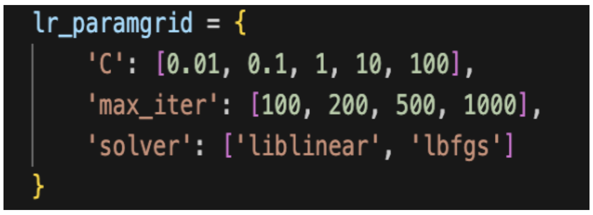
The parameter combination with the highest accuracy score was selected. Selecting the combination with the highest accuracy is a reasonable choice because accuracy is a straightforward and widely used metric that reflects the proportion of correctly classified instances. For classification tasks where the classes are balanced, accuracy is an effective metric to assess model performance. However, for imbalanced datasets, additional metrics like precision, recall, or F1-score may also need to be considered to provide a more nuanced evaluation.(3)

After selection, the best models were evaluated with the training data by LOOVC crossvalidation.The accuracy score and a clasification report was obtained.

2.1.1 Logistic Regression parameters

Logistic regression is a statistical model commonly used for binary and multiclass classification tasks. It predicts the probability of an outcome belonging to a particular class by applying a logistic function (sigmoid) to a linear combination of input features. This approach maps input features to a probability value between 0 and 1, making it well-suited for classification problems where the output is categorical (4)

The model optimizes weights for input features by minimizing a cost function, typically log loss or cross-entropy, using iterative methods. This optimization process is guided by several parameters that influence performance and behavior.



```
lr_paramgrid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'max_iter': [100, 200, 500, 1000],
    'solver': ['liblinear', 'lbfgs']
}
```

Figura 1 Grid for Logistic Regression hyperparameter tuning.

The parameters used to fine-tune logistic regression performance include:

- **Regularization strength (C):** This parameter controls the trade-off between model complexity and overfitting. Smaller values of C apply stronger regularization, simplifying the model and enhancing generalization. Larger values reduce regularization, allowing the model to fit the training data more closely.

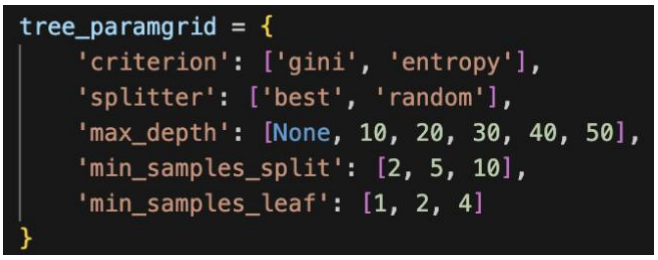
- **Maximum iterations (max_iter):** This parameter sets the maximum number of iterations for the optimization algorithm to converge. Higher values ensure the algorithm has sufficient steps to reach convergence, particularly useful for complex datasets.

- **Solver:** The solver determines the optimization algorithm used to adjust the model's weights. Different solvers are suited to different scenarios:
    - **liblinear:** Ideal for smaller datasets and binary classification, as it uses a coordinate descent algorithm.
    - **lbfgs:** Suitable for larger datasets and multiclass classification tasks, leveraging quasi-Newton optimization methods.

After applying the hyperparametet tuning pipeline, the best parameter combination for this dataset for a logistic regression approach are C=1, max_iter = 100 and Solver = lbfgs.

2.1.2 Decission Tree parameters

A decision tree is a versatile machine learning algorithm used for classification and regression tasks. It builds a tree-like model of decisions based on input features, splitting data into subsets at each node based on feature values. The algorithm proceeds until a stopping criterion is met, such as reaching a maximum depth or achieving pure leaf nodes. The decision tree's interpretability and ability to handle both numerical and categorical data make it a popular choice for various applications (5)



```
tree_paramgrid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

Figura 2 Grid for hyperparameter tuning of Decision Tree

The parameters used to fine-tune decision tree performance include:

- **Criterion:** This parameter determines the function used to measure the quality of a split. The options are:
  - **Gini:** Uses Gini impurity to assess splits, favoring balanced splits.
  - **Entropy:** Uses information gain to maximize the purity of child nodes.

- **Splitter:** Specifies the strategy for selecting splits at each node:
  - **Best:** Considers all features and selects the optimal split.
  - **Random:** Considers a random subset of features for splitting.

- **Maximum depth (max_depth):** Limits the depth of the tree to prevent overfitting. A deeper tree captures more detail but may overfit the training data, while a shallower tree enhances generalization.

- **Minimum samples to split (min_samples_split):** Sets the minimum number of samples required to split an internal node. Higher values reduce overfitting by ensuring splits are based on sufficiently large subsets.

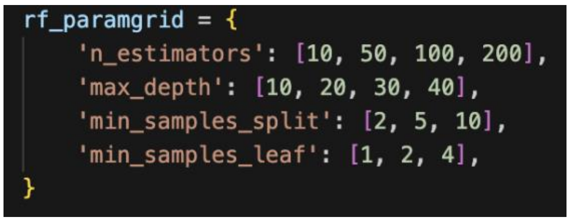
- **Minimum samples per leaf (min_samples_leaf):** Defines the minimum number of samples required in a leaf node. Larger values prevent splits that produce overly small leaf nodes, enhancing generalization.

The hyperparameter tuning pipeline determined the following optimal parameters for this dataset:

- **Criterion:** Gini
- **Max_depth:** None
- **Min_samples_split:** 2
- **Min_samples_leaf:** 2
- **Splitter:** Best

2.1.3 Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve classification and regression performance. Each tree in the forest is built on a random subset of the training data and features, introducing diversity among the trees. Predictions are made by aggregating the outputs of individual trees, either through majority voting (classification) or averaging (regression). This ensemble approach reduces the risk of overfitting and enhances generalization compared to a single decision tree (6)



```
rf_paramgrid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [10, 20, 30, 40],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}
```

Figura 3 Grid for hyperparameter tuning of Random Forest

The parameters used to fine-tune Random Forest performance include:

- **Number of estimators (n_estimators):** Specifies the number of trees in the forest. More trees generally improve performance but increase computational cost.

- **Maximum depth (max_depth):** Limits the depth of individual trees to control model complexity. Deeper trees capture more patterns but may overfit the data.

- **Minimum samples to split (min_samples_split):** Defines the minimum number of samples required to split a node. Higher values help prevent overfitting by ensuring splits occur only for sufficiently large subsets.

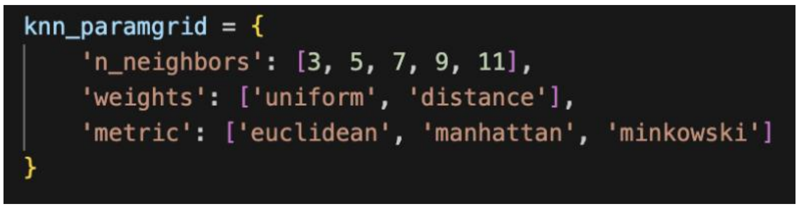
- **Minimum samples per leaf (min_samples_leaf):** Specifies the minimum number of samples required in a leaf node. Larger values encourage more generalizable models by preventing overly small leaves.

The hyperparameter tuning pipeline determined the following optimal parameters for this dataset:

- **n_estimators:** 100
- **Max_depth:** 10
- **Min_samples_split:** 2
- **Min_samples_leaf:** 1

2.1.4 K-Nearest Neighbour

K-Nearest Neighbors (KNN) is a simple and effective algorithm used for classification and regression tasks. The algorithm works by identifying the k-nearest data points to a query point based on a chosen distance metric and predicting the output based on these neighbors. In classification, the output is typically determined by majority voting among the neighbors, while in regression, the output is the average of their values. KNN's simplicity and non-parametric nature make it a popular choice for many applications, especially with well-separated data (7)



```
knn_paramgrid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski']
}
```

Figura 4 Grid for hyperparameter tuning of KNN

The parameters used to fine-tune KNN performance include:

- **Number of neighbors (n_neighbors):** Determines the number of nearest data points considered when making a prediction. Smaller values can make the model sensitive to noise, while larger values improve stability but may blur class boundaries.

- **Weights:** Specifies how the neighbors contribute to the prediction:
    - **Uniform:** All neighbors contribute equally.

    - **Distance:** Closer neighbors have more influence on the prediction.

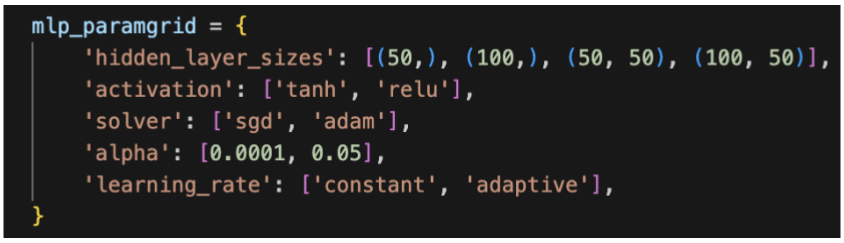
- **Metric:** Defines the distance metric used to identify the nearest neighbors. Common options include:
    - **Euclidean:** Measures straight-line distance.
    - **Manhattan:** Measures distance along axes.
    - **Minkowski:** Generalized distance metric that includes both Euclidean and Manhattan as special cases.

The hyperparameter tuning pipeline determined the following optimal parameters for this dataset:

- **Metric:** Euclidean
- **n_neighbors:** 3
- **Weights:** Uniform

2.1.5 Multilayer Perceptron Classifier

A Multi-Layer Perceptron (MLP) is a type of artificial neural network designed for supervised learning tasks such as classification and regression. It consists of an input layer, one or more hidden layers, and an output layer. Each layer contains neurons (nodes) connected by weighted edges, which are adjusted during training to minimize the prediction error. MLPs use backpropagation to iteratively update weights and biases, optimizing a loss function to achieve accurate predictions (8)



```
mlp_paramgrid = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
}
```

Figura 5 Grid for hyperparameter tuning of MLP

The parameters used to fine-tune MLP performance include:

- **Hidden layer sizes (hidden_layer_sizes):** Defines the number of neurons in each hidden layer. A deeper and wider network can capture more complex patterns but may risk overfitting.

- **Activation function (activation):** Specifies the non-linear transformation applied to the input of each neuron. Common options include:
    - **Relu (Rectified Linear Unit):** Efficient and widely used for deep networks.

    - **Tanh:** Maps inputs to values between -1 and 1, suitable for certain tasks.
    - **Logistic (sigmoid):** Maps inputs to values between 0 and 1, often used for binary classification.

- **Solver:** Determines the optimization algorithm for weight updates:
    - **Adam:** Combines the advantages of RMSProp and momentum, suitable for large datasets.
    - **SGD:** Stochastic Gradient Descent, effective for smaller datasets.

- **Alpha:** The L2 regularization term that prevents overfitting by penalizing large weights. Smaller values reduce regularization, allowing more flexibility, while larger values increase regularization.

- **Learning rate (learning_rate):** Controls the step size for weight updates:
    - **Constant:** Uses a fixed learning rate throughout training.
    - **Adaptive:** Adjusts the learning rate dynamically based on performance.

The hyperparameter tuning pipeline determined the following optimal parameters for this dataset:

- **Activation:** Relu
- **Alpha:** 0.0001
- **Hidden_layer_sizes:** 50,50
- **Learning_rate:** Constant
- **Solver:** Adam

2.2 Test evaluation

The models were evaluated with a set of unseen data. This is done to assess how well the model performs on data it has never encountered before. This ensures that the model has not overfitted to the training data and provides a more realistic measure of its effectiveness. Unseen data evaluation is a critical step in validating the robustness of the model and ensuring its reliability for practical applications.

A confusion matrix was plotted to support the results from the test data prediction. A confusion matrix is a visualization tool that summarizes the performance of a classification model by displaying the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. It helps to understand not just the overall accuracy but also the distribution of errors among the classes.

3 Results

The following tables provide performance metrics (precision, recall, F1-score, and overall accuracy) for the best models evaluated on the training and test datasets. The metrics are presented for two classes: NR (Negative Response) and R (Response).

3.1 Logistic Regression

Training

	Precision	Recall	F1-score	Support
NR	0.92	0.92	0.92	13
R	0.94	0.94	0.94	18
Accuracy			0.94	31

Table 1 Training performance metrics for the best Logistic Regression model

The model shows high precision (0.92 for NR and 0.94 for R) and recall (0.92 for NR and 0.94 for R) for both classes, indicating strong performance in correctly identifying positive and negative cases during training.The F1-scores (0.92 for NR and 0.94 for R) reflect a good balance between precision and recall, signifying robust performance across both classes. The overall accuracy is 0.94, indicating that the model correctly classifies 94% of the training data instances.

Test

	Precision	Recall	F1-score	Support
NR	0.91	1.00	0.95	10
R	1.00	0.92	0.96	12
Accuracy			0.95	22

Table 2 Test performance metrics for the best Logistic Regression model

On the test set, the model maintains strong performance, with precision values of 0.91 (NR) and 1.00 (R). Recall is perfect for NR (1.00) but slightly lower for R (0.92), suggesting that the model occasionally misses positive cases for the R class.

The F1-scores are 0.95 (NR) and 0.96 (R), demonstrating consistent performance in balancing precision and recall, even on unseen data.

The overall accuracy is 0.95, showing that 95% of the test instances are classified correctly, indicating strong generalization of the model.

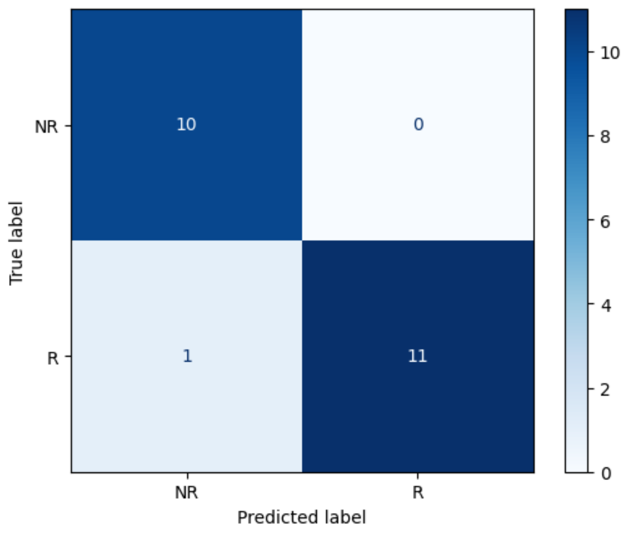


Figura 6 Confusion matrix of the best Logistic Regression model for the test dataset

The confusion matrix for the test data reveals that the model performs exceptionally well, correctly classifying 10 instances of the NR class and 11 instances of the R class, achieving an overall accuracy of 95%. There are no false positives for the NR class, while the R class has one false negative, indicating that one positive case was misclassified as negative. This aligns with the previously observed metrics, showing perfect recall for NR and slightly lower recall for R. The model demonstrates a strong balance between classes, with minimal errors and no significant bias.

Feature importance analysis reveals that positive coefficients (e.g., n_rs9960669 with 0.172, odds ratio 1.188) increase the likelihood of the positive class, while negative coefficients (e.g., n_rs3188513 with -0.854, odds ratio 0.426) reduce it. The most influential positive predictors include n_rs9960669 (18.8% increase in odds) and n_rs933069 (15.9% increase in odds).

Conversely, n_rs3188513, n_rs5274755, and n_rs344903 strongly decrease the odds of the positive class (reductions of 57.4%, 55.1%, and 54.2%, respectively). Strong predictors like these can guide feature engineering, while features with minimal impact (e.g., n_rs6236416) could be considered for removal to simplify the model.

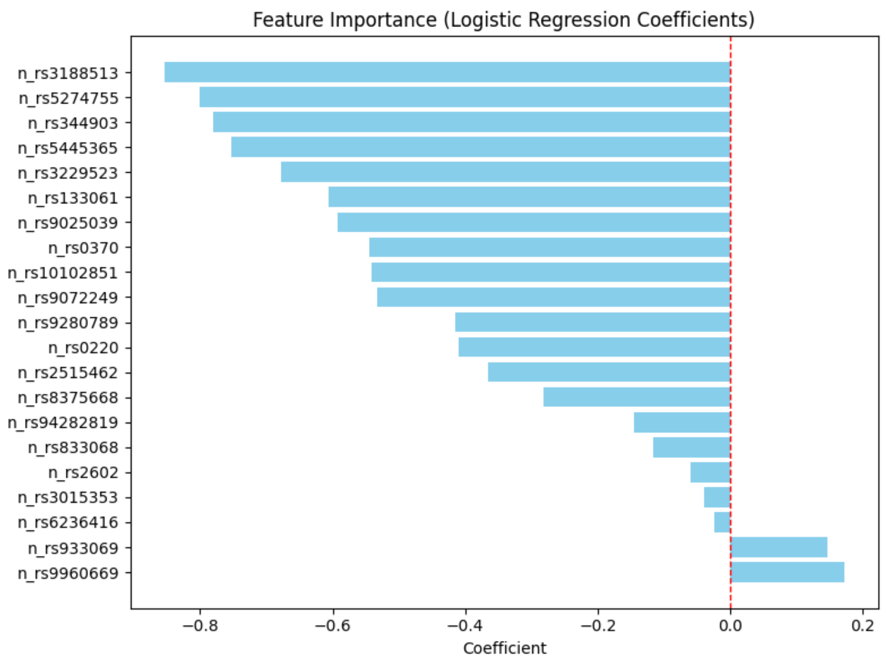


Figura 7 Logistic Regression feature importance

3.2 Decision Tree

<u>Training</u>

	Precision	Recall	F1-score	Support
NR	0.75	0.92	0.83	13
R	0.93	0.78	0.85	18
Accuracy			0.84	22

Table 3 Training performance metrics for the best Decision Tree model

The performance metrics for the Decision Tree model demonstrate varied results between training and test datasets. During training, the model achieves an overall accuracy of 84%, with an F1-score of 0.83 for the NR class and 0.85 for the R class. The recall for NR is notably high at 0.92, indicating strong identification of negative cases, while the recall for R is lower at 0.78, suggesting some missed positive cases.

Test

	Precision	Recall	F1-score	Support
NR	0.67	0.40	0.50	10
R	0.62	0.83	0.71	12
Accuracy			0.64	22

Table 4 Test performance metrics for the best Decision Tree model

On the test data, the performance drops significantly, with an overall accuracy of 64%. The precision and recall for NR (0.67 and 0.40, respectively) and for R (0.62 and 0.83, respectively) reflect reduced effectiveness in generalizing to unseen data. The F1-scores also highlight this decline, at 0.50 for NR and 0.71 for R. These results indicate that the Decision Tree model struggles to maintain its training performance on the test set, potentially due to overfitting.

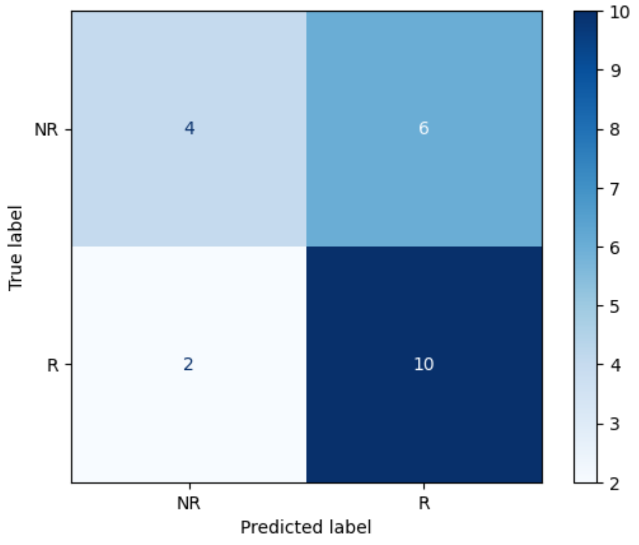


Figura 8 Confusion matrix of the best Decision Tree model for the test dataset

The confusion matrix for the Decision Tree model on the test data highlights the model's challenges in accurately classifying instances. For the NR class, only 4 instances were correctly classified as NR (true negatives), while 6 were misclassified as R (false positives), indicating a significant issue in identifying negative cases. For the R class, the model correctly classified 10 instances as R (true positives), but 2 were misclassified as NR (false negatives), showing slightly better performance for this class.

This distribution aligns with the metrics observed earlier, where the recall for R is higher than that for NR, indicating the model's tendency to favor positive classifications. The overall accuracy of 64% reflects the imbalance in performance between the two classes and the general difficulty the model faces in handling unseen data effectively. This confusion matrix underlines the need for further tuning or adjustments, such as addressing class imbalance or using ensemble methods to improve generalization.

The feature importance was calculated and plotted in Figure 8. The decision tree model heavily relies on three key features: n_rs344903, n_rs3229523, and n_rs9072249, with n_rs344903 being the most influential as it drives the root split, reducing impurity significantly

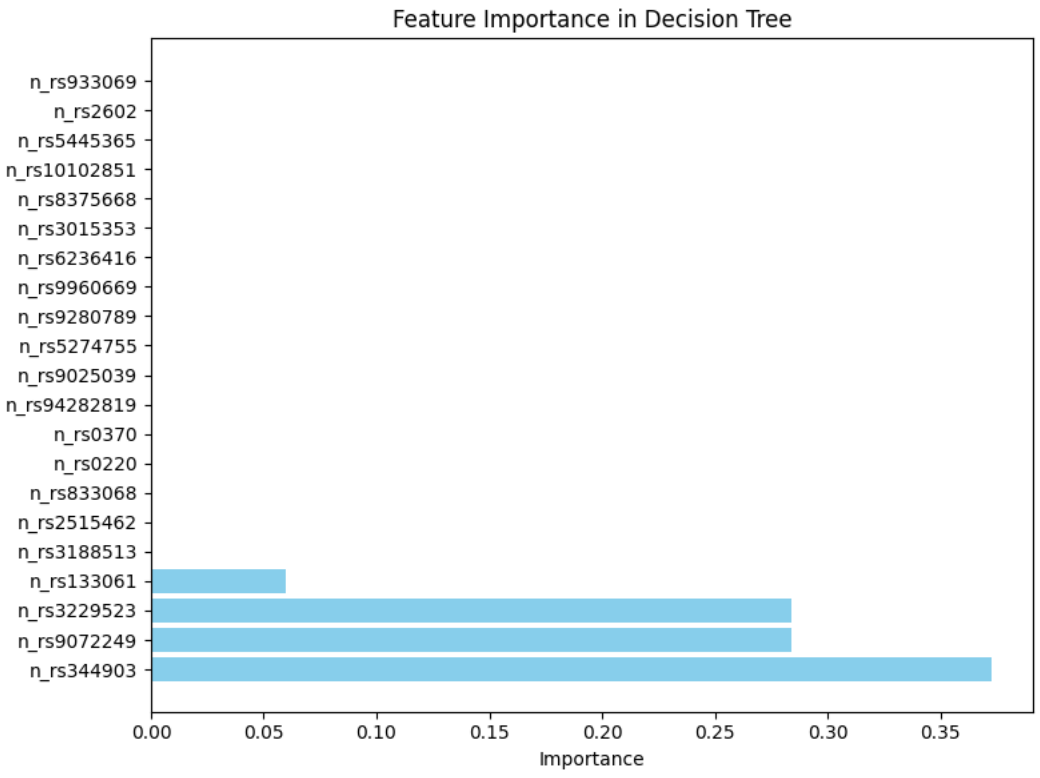


Figura 9 Feature importance for Decision Tree classifier

To expand the interpretation of the Decision Tree model, the tree structure was plotted. The decision tree model begins with a root node split on n_rs344903 (the most important feature), dividing the data into two branches with a Gini impurity of 0.487, where the left branch (n_rs344903 ≤ 1.5) primarily contains R samples and the right branch (n_rs344903 > 1.5) mostly contains NR samples. The left branch further splits on n_rs133061, creating a pure leaf node with 11 R samples (Gini = 0) and a mixed node with 1 R and 1 NR sample (Gini = 0.5). The right branch splits on n_rs3229523, followed by n_rs9072249, producing pure leaf nodes with either all R or all NR samples. Most splits lead to pure or nearly pure leaf nodes, demonstrating the tree's ability to separate the classes effectively, with small areas of impurity (e.g., 1 R and 1 NR) highlighting potential feature limitations. Key features like n_rs344903, n_rs3229523, and n_rs9072249 dominate the splits, driving the model's predictive power, while other features contribute minimally or not at all

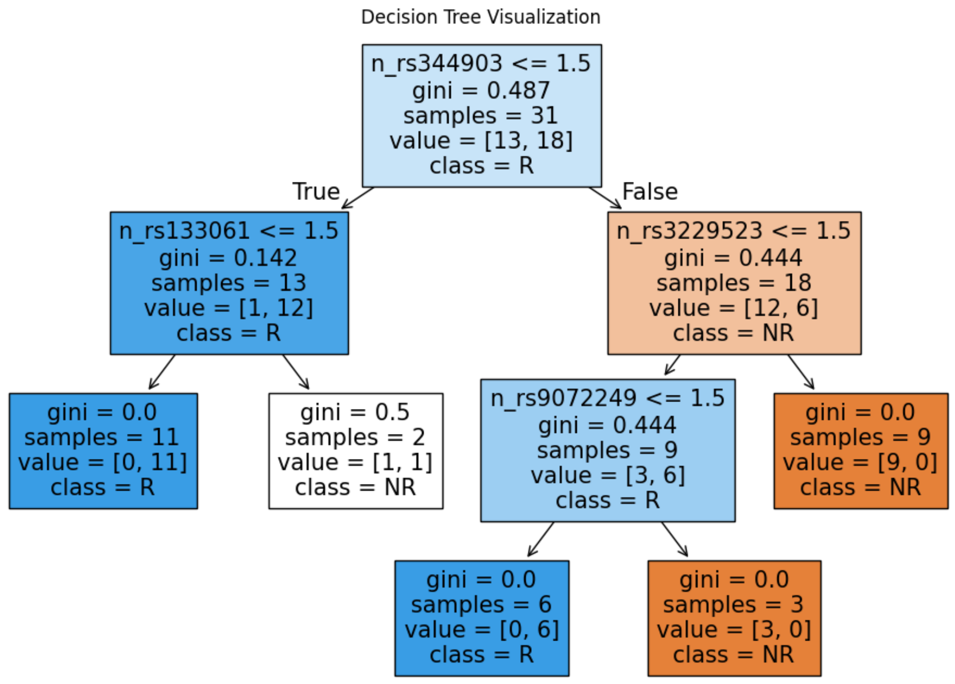


Figura 10 Decision Tree structure

3.3 Random Forest

<u>Training</u>

	Precision	Recall	F1-score	Support
NR	1.00	0.85	0.92	13
R	0.90	1.00	0.95	18
Accuracy			0.94	31

Table 5 Training performance metrics for the best Random Forest model

The Random Forest model shows strong performance on both the training and test datasets. During training, the model achieves a high overall accuracy of 94%, with F1-scores of 0.92 for the NR class and 0.95 for the R class. The recall for NR (0.85) and precision for R (0.90) indicate slight variability, but overall, the model captures patterns effectively in the training data.

Test

	Precision	Recall	F1-score	Support
NR	1.00	0.80	0.89	10
R	0.86	1.00	0.92	12
Accuracy			0.91	22

Table 6 Test performance metrics for the best Random Forest model

On the test set, the model maintains robust performance, with an accuracy of 91%. The F1-scores are 0.89 for NR and 0.92 for R, indicating consistent balance between precision and recall. For the NR class, the recall drops slightly to 0.80, while precision remains perfect (1.00), suggesting that the model occasionally misses some negative cases. For the R class, the recall is perfect (1.00), and precision is high (0.86), demonstrating the model's ability to identify all positive cases with minor over-predictions.

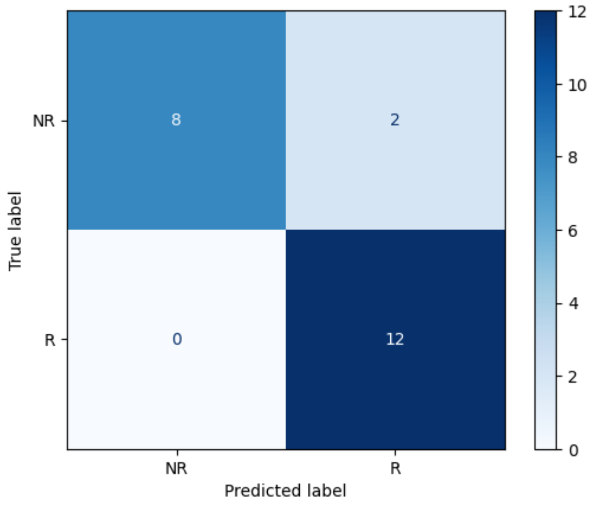


Figura 11 Confusion matrix of the best Random Forest model for the test dataset

The confusion matrix for the test data further supports these metrics. It shows that 8 instances of the NR class are correctly classified, with 2 misclassified as R. For the R class, all 12 instances are correctly classified. This performance highlights the Random Forest model's strong ability to generalize to unseen data, with only minor issues in accurately classifying some negative cases. Overall, the Random Forest demonstrates reliability and high predictive capability for this dataset.

3.4 K-Nearest Neighbour

Training

	Precision	Recall	F1-score	Support
NR	0.92	0.85	0.88	13
R	0.89	0.94	0.92	18
Accuracy			0.90	31

Table 7 Training performance metrics for the best KNN model

The KNN model exhibits consistent but slightly lower performance compared to other algorithms. During training, the model achieves an accuracy of 90%, with an F1-score of 0.88 for the NR class and 0.92 for the R class. The recall for NR (0.85) indicates good identification of negative cases, while the recall for R is higher at 0.94, reflecting a strong ability to capture positive cases. Precision is balanced across both classes, at 0.92 for NR and 0.89 for R, suggesting the model performs reliably during training.

Test

	Precision	Recall	F1-score	Support
NR	0.82	0.90	0.86	10
R	0.91	0.83	0.87	12
Accuracy			0.86	22

Table 8 Test performance metrics for the best KNN model

On the test set, the model's accuracy drops to 86%. For the NR class, precision is 0.82, and recall improves to 0.90, resulting in an F1-score of 0.86. For the R class, precision rises to 0.91, while recall decreases to 0.83, producing an F1-score of 0.87. These results indicate the model generalizes moderately well, though with slightly uneven performance across classes.

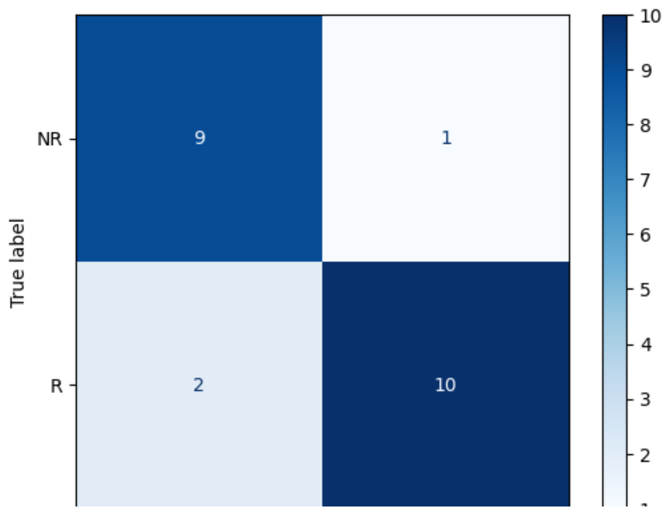


Figura 12 Confusion matrix of the best KNN model for the test dataset

The confusion matrix highlights the test set predictions: 9 instances of NR are correctly classified, with 1 misclassified as R. For the R class, 10 instances are correctly classified, while 2 are misclassified as NR. This suggests the model slightly favors identifying negative cases, with a minor tendency to misclassify positive cases as negative. Overall, KNN provides reliable performance but may benefit from fine-tuning to address its variability in precision and recall across classes.

3.5 Multilayer Perceptron Classifier

Training

	Precision	Recall	F1-score	Support
NR	0.75	0.69	0.72	13
R	0.79	0.83	0.81	18
Accuracy			0.77	31

Table 9 Training performance metrics for the best MLP model

The MLP model demonstrates moderate performance on both training and test datasets. During training, the model achieves an overall accuracy of 77%, with F1-scores of 0.72 for the NR class and 0.81 for the R class. Precision and recall for the NR class are 0.75 and 0.69, respectively, indicating that the model struggles slightly with identifying all negative cases. For the R class, precision is 0.79 and recall is 0.83, showing better identification of positive cases but still with some room for improvement.

Test

	Precision	Recall	F1-score	Support
NR	0.80	0.80	0.80	10
R	0.83	0.83	0.83	12
Accuracy			0.82	22

Table 10 Test performance metrics for the best MLP model

On the test set, the overall accuracy increases slightly to 82%. Both classes exhibit balanced performance, with precision, recall, and F1-scores of 0.80 for the NR class and 0.83 for the R class. This consistency between precision and recall highlights the model's ability to generalize well across the test set, despite the moderate performance levels.

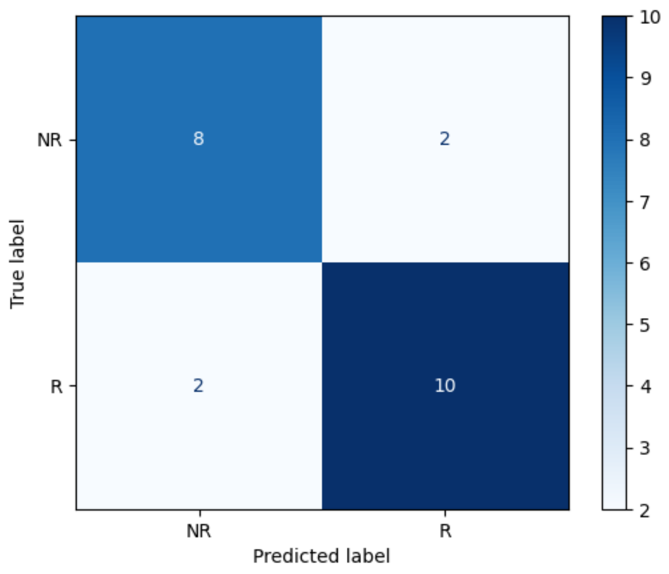


Figura 13 Confusion matrix of the best MLP model for the test dataset

The confusion matrix for the test data reveals that 8 instances of the NR class are correctly classified, with 2 misclassified as R, and 10 instances of the R class are correctly classified, with 2 misclassified as NR. These results confirm that the MLP model is relatively balanced in its predictions for both classes but could benefit from further optimization to reduce misclassifications and enhance overall performance.

4. Model selection

Based on the performance metrics, the **Logistic Regression** model stands out as the best model. It achieves the highest test accuracy (0.95), demonstrating excellent generalization to unseen data, while maintaining consistent training accuracy (0.94), indicating robustness without overfitting.

Its simplicity and interpretability further enhance its appeal, making it easier to understand and explain compared to more complex models like Random Forest or MLP. Although Random Forest is a close second (0.91) in test accuracy, logistic regression's higher performance and straightforward nature make it the more favorable choice. Models like KNN and MLP have lower test accuracies (0.86 and 0.82, respectively), and the Decision Tree is clearly the weakest (0.64) with severe overfitting. Therefore, logistic regression is the most reliable, accurate, and interpretable model, making it the best option for deployment.

	Training accuracy	Test Accuracy
Logistic Regression	0.94	0.95
Decision Tree	0.84	0.64
Random Forest	0.94	0.91
KNN	0.90	0.86
MLP	0.77	0.82

Table 11 Training and Test accuracies for the different machine learning models.

5. Bibliografía

1. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. 2012 [citado 23 de diciembre de 2024]; Disponible en: https://arxiv.org/abs/1201.0490

2. Arlot S, Celisse A. A survey of cross-validation procedures for model selection. Statist Surv [Internet]. 1 de enero de 2010 [citado 23 de diciembre de 2024];4(none). Disponible en: https://projecteuclid.org/journals/statistics-surveys/volume-4/issue-none/A-survey-of-cross-validation-procedures-for-model-selection/10.1214/09-SS054.full

3. Sokolova M, Lapalme G. A systematic analysis of performance measures for classification tasks. Information Processing & Management. julio de 2009;45(4):427-37.

4. Hosmer DW, Lemeshow S, Sturdivant RX. Applied Logistic Regression [Internet]. 1.ª ed. Wiley; 2013 [citado 23 de diciembre de 2024]. (Wiley Series in Probability and Statistics). Disponible en: https://onlinelibrary.wiley.com/doi/book/10.1002/9781118548387

5. Quinlan JR. Learning decision tree classifiers. ACM Comput Surv. marzo de 1996;28(1):71-2.

6. Breiman, Leo. Random Forest. 2001;45:5-32.

7. Altman NS. An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. The American Statistician. agosto de 1992;46(3):175-85.

8. LeCun Y, Bengio Y, Hinton G. Deep learning. Nature. 28 de mayo de 2015;521(7553):436-44.