Oregon State University

College of Science
Department of Statistics

Predictive Modeling in Low- and High-Dimensional Settings: A Machine Learning Approach to Idiopathic Pulmonary Fibrosis (IPF) Progression

Submitted by:

Benson Cyril Nana Boakye

A writing project submitted in partial fulfillment

of the requirements for the degree

Master of Science in Statistics

Under the supervision of

Professor Lisa Ganio

June 2, 2025

APPROVAL

of a writing project submitted by

Benson Cyril Nana Boakye

This writing project has been reviewed by the writing project advisor and has been found to be satisfactory regarding content, English usage, formatting, citation style, bibliographic style, and consistency, and is ready for submission to the Statistics Faculty.

Date

Prof. Lisa Ganio
Writing Project Advisor

Date

Prof. Tate Jacobson
Graduate Committee Member

Date

Prof. Yuan jiang
Graduate Committee Member

Contents

List of Tables

List of Figures

1 Introduction

A practical application of statistical modeling is using existing data to predict future outcomes and understand the underlying structure that drives those outcomes. In biomedical data science, this dual capability is crucial for identifying clinically relevant covariates and building models that inform prognosis, stratify risk, and guide decisions. This analysis focuses on **`Idiopathic Pulmonary Fibrosis (IPF)`**, a chronic, progressive, and irreversible lung disease. `IPF` is especially challenging due to its unpredictability: some patients decline slowly, others deteriorate rapidly with respiratory failure. This variability makes it hard for doctors to identify which patients are at high risk and when to intervene.

In statistical learning, modeling objectives can be framed around two paradigms: **`inference and prediction`**. Inference seeks to quantify and interpret the relationships between variables and outcomes. `Prediction` focuses on building models that generalize to new data and produce accurate forecasts even if the internal structure is less interpretable. I used logistic regression, decision trees, random forests, and LASSO logistic regression, to explore how well different models could predict whether patients with IPF would experience disease progression or not. I used multi-phased analytical framework: In `Phase 1`, models are trained on a curated set of 14 covariates. This phase allows us to assess performance across different modeling approaches in a controlled, **`low-dimensional setting`** (fewer predictors than observations). In `Phase 2`, I extended the analysis to the full feature set of 1,129 covariates to explore how well different models perform in a **`high-dimensional setting`** (more predictors than observations).

1.1 Objectives of the Study

This study has three main objectives. First, to build and compare classification models for IPF progression across **`low - and high-dimensional settings`**. Second, to evaluate model performance using a range of classification metrics, including accuracy, sensitivity, specificity, F1 score, AUC, Positive and Negative Predicted Value and others. Finally, to develop practical proficiency in clinical data analysis by executing a complete machine learning workflow encompassing data preprocessing, model development, variable selection, and performance evaluation.

2 Methodology

This section provides a conceptual overview of the statistical techniques used in this study, offering theoretical background to support the analysis that follows.

2.1 The Generalized Linear Model (GLM)

`GLM` is a flexible extension of ordinary linear regression that allows the response variable to follow a distribution from the exponential family rather than assuming normality [2]. Given parameters $\boldsymbol{\theta}$ and data $\boldsymbol{x}$, a distribution belongs to the `exponential family` if its probability density function can be expressed as:

$$f(\boldsymbol{x} \mid \boldsymbol{\theta}) = \boldsymbol{h}(\boldsymbol{x})\, \boldsymbol{c}(\boldsymbol{\theta}) \, \exp\left(\sum_{i=1}^{k} \boldsymbol{w}_i(\boldsymbol{\theta})\, \boldsymbol{t}_i(\boldsymbol{x})\right)$$

where $\boldsymbol{h}(x)$ is a function of the data, $\boldsymbol{c}(\theta)$ is a function of the parameters, $\boldsymbol{w}_i(\theta)$ are natural parameter functions, and $\boldsymbol{t}_i(x)$ are sufficient statistics. The general framework for the model is given as:

$$\boldsymbol{g}(\boldsymbol{\mu}_i) = \boldsymbol{g}\left(E(\boldsymbol{y}_i \mid \tilde{\mathbf{x}}_i, \boldsymbol{\theta})\right) = \boldsymbol{x}_t'\boldsymbol{\beta} = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 x_{i1} + \cdots + \boldsymbol{\beta}_p x_{ip}$$

This expression highlights the three core components of a GLM: `Random component:`$y_i \sim$ exponential family with mean $\mu_i = E(y_i \mid \tilde{\mathbf{x}}_i, \boldsymbol{\theta})$; `Systematic component:` the linear predictor $\boldsymbol{x}_t'\boldsymbol{\beta}$; and a `Link function` $g(\cdot)$, which connects the mean of the response to the linear predictor.

2.2 Logistic Regression

Logistic regression is one of the most widely used algorithms in supervised machine learning and a commonly form of GLM, particularly for binary classification. It models the probability of a binary outcome as a function of one or more explanatory variables, assuming the response follows a binomial distribution, often framed as counts of successes in independent Bernoulli trials. Logistic regression models the log-odds of the outcome as a linear function of the predictors. This is achieved using the logit function:

$$\boldsymbol{g}[\pi(\tilde{\mathbf{x}})] = \log\left[\frac{\pi(\tilde{\mathbf{x}})}{1 - \pi(\tilde{\mathbf{x}})}\right] = \eta = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 x_1 + \boldsymbol{\beta}_2 x_2 + \cdots + \boldsymbol{\beta}_p x_p$$

Where $\pi(\tilde{\mathbf{x}})$ is the **probability of success** and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are the explanatory variables. Each β_j denotes the estimated coefficient corresponding to the predictor named. The **logit function** transforms the probability $\pi(\tilde{\mathbf{x}})$ from the range $(0, 1)$ to the entire real number line $(-\infty, \infty)$, making it possible to model probabilities through a linear equation. Solving for $\pi(\tilde{\mathbf{x}})$, yields the logistic or sigmoid function. [3]:

$$\pi(\tilde{\mathbf{x}}) = \frac{e^k}{1+e^k} = \frac{1}{1+e^{-k}} \quad \text{where} \quad k = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 x_1 + \boldsymbol{\beta}_2 x_2 + \cdots + \boldsymbol{\beta}_p x_p$$

The parameters $\boldsymbol{\beta}$ in the model are estimated using `maximum likelihood estimation (MLE)`, which identifies the estimates of $\boldsymbol{\beta}$ that maximize the likelihood of observing the outcomes in the dataset. The likelihood for logistic regression is:

$$\mathcal{L}(\pi(\tilde{\mathbf{x}}_i) \mid \mathbf{y}) = \prod_{i=1}^{n} \pi(\tilde{\mathbf{x}}_i)^{y_i} \left(1 - \pi(\tilde{\mathbf{x}}_i)\right)^{1-y_i}$$

In practice, the `log-likelihood` form is used for numerical optimization, as it is easier to handle:

$$\boldsymbol{\ell}(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left[\mathbf{y}_i \log\left(\frac{1}{1+e^{-k_i}}\right) + (1-\mathbf{y}_i)\log\left(1 - \frac{1}{1+e^{-k_i}}\right)\right] \quad \text{where } k_i = \boldsymbol{\beta}_0 + \boldsymbol{\beta}_1 x_{i1} + \cdots + \boldsymbol{\beta}_p x_{ip}$$

This function is optimized iteratively until convergence criteria are met with `Newton-Raphson which translates to Iteratively Reweighted Least Squares (IRLS)`.

2.2.1 Assumptions

In using logistic regression, I had to keep a few key assumptions in mind. First, the method assumes a `linear relationship` between the log-odds of the outcome and the betas, known as linearity in the logit [3]. It also assumes that the predictors are not `highly correlated`, because multicollinearity can make it difficult to tell which variables matter. Another assumption is that each observation is independent, meaning that the outcome for each patient is not influenced by the outcome of another. Since I did not have information on relationships or shared environments in the data, I assumed independence. I also had to be mindful of outliers. In smaller datasets, they can be detected with simple visualizations or basic statistical thresholds, but in high-dimensional data, it is much harder to identify when an observation is extreme.

2.2.2 Logistic Lasso Regression

To address challenges like multicollinearity and overfitting, I applied Lasso, a regularized form of logistic regression. Regularization helps simplify models and improves their ability to generalize to new data, which is especially important in high-dimensional settings [4]. Lasso also performs variable selection. Unlike Ridge regression, which shrinks all coefficients toward zero but keeps them in the model using L2 penalty, Lasso uses an L1 penalty defined as $\lambda \sum_{j=1}^{p} |\beta_j|$, that can shrink some coefficients exactly to zero, effectively removing irrelevant predictors. The loss function of logistic lasso regression is given by:

$$\hat{\boldsymbol{\beta}}_{\text{Lasso}} = \arg\min_{\boldsymbol{\beta}} \left\{ -\sum_{i=1}^{n} \left[y_i \log\left(\pi(\tilde{\mathbf{x}}_i)\right) + (1-y_i)\log\left(1-\pi(\tilde{\mathbf{x}}_i)\right)\right] + \lambda \sum_{j=1}^{p} |\beta_j| \right\}$$

where the tuning parameter λ controls the severity of the penalty. Lasso regression can handle mild multicollinearity without compromising interpretability, but when predictors are highly correlated, it may arbitrarily retain one and discard the others.

2.3 Random Forest

Random forest is a machine learning algorithm that creates an ensemble of decision trees to make more accurate prediction than a single tree. Since decision trees constitute the fundamental building blocks of the Random Forest algorithm, it is necessary to first provide a brief overview of the decision tree model before proceeding to the ensemble mechanism and its implementation in the Random Forest framework.

2.3.1 Decision Trees

A decision tree is a non-parametric machine learning method used for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf node. At each step, the data are split into two parts based on the value of a predictor, aiming to create groups that are as pure as possible, that is, where most or all of the observations in that node belong to the same class. This process is typically guided by the **`Classification and Regression Tree (CART) algorithm`**. When the outcome is continuous, the model is called a regression tree; when it is categorical, it is a classification tree. Since this study involves predicting a binary outcome, I focus specifically on classification trees.

2.3.2 Measuring Node Impurity in Classification Trees

Instead of minimizing residual sum of squares like in regression trees, classification trees create purer groups by minimizing **`node impurity`**. Common impurity measures include the **`classification error rate`**, defined as $E_m = 1 - \max_k \left(\frac{n_{mk}}{n_m}\right)$, where n_m is the number of observations in node m, and n_{mk} is the number of observations in the kth class. It measures the proportion of misclassified observations in the node. The **`entropy`** is given by $D_m = -\sum_{k=1}^{K} p_{mk} \log_2(p_{mk})$, and quantifies the level of disorder or uncertainty in a node. The **`Gini index`** is $G_m = 1 - \sum_{k=1}^{K} p_{mk}^2$, where $p_{mk} = \frac{n_{mk}}{n_m}$ is the proportion of observations in class k within node m. It measures the total variance across the K classes in a node. In general, the classification error is not sensitive enough to differences in class purity, for tree construction, so the **`Gini index`** or **`entropy`** is typically preferred [5].

2.3.3 Algorithm for Building a Classification Tree

To build a classification tree, I used a method called **`recursive binary splitting`**, which builds the tree step-by-step by selecting the best split at each point [5]. The process starts with the full training dataset. For each predictor and possible split, I checked how well the split separated the classes using impurity measures like the Gini index or entropy. The best split divides the data into two parts, $\{X_j \leq s\}$ and $\{X_j > s\}$, where X_j is the chosen variable and s is the split point. This process continues on each subset until a stopping rule is met, such as reaching a minimum number of observations or maximum depth. Once the tree is fully grown, I used **`cost-complexity pruning`** [5] to simplify it and avoid overfitting. This involves minimizing the function $C_\alpha(T) = \sum_{m=1}^{|T|} \text{Error}(R_m) + \alpha|T|$, where $\text{Error}(R_m)$ is the impurity of region R_m, $|T|$ is the number of terminal nodes, and α controls the penalty for complexity. Although pruning uses training data, I chose the best α using cross-validation to improve the performance of the tree on new data.

2.3.4 Bagging

Bagging, short for **`bootstrap aggregation`**, is a method used to make predictions more stable and accurate, especially for models like decision trees. It works by taking many random samples from the training data (with replacement), building a model on each sample, and then combining their results. For classification, the final prediction is based on majority vote using all predictions from all random samples; for regression, it's the average of the predictions. The bagged prediction is given by $\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b^*(x)$, where B is the number of models and $\hat{f}_b^*(x)$ is the prediction from the bth model.

2.3.5 Out-of-Bag Error Estimation

Bagging offers a built-in way to estimate test error without a separate validation set or cross-validation, through what is known as **`Out-of-Bag (OOB)`** error. Since each tree is trained on a bootstrap sample, about one-third of the training data is left out for that tree. These unused observations are called *out-of-bag*. For each observation, we collect predictions from all trees where it was not included in training and combine them to form an OOB prediction. For classification, we take a majority vote among these predictions. The OOB error is calculated as $\text{OOB Error} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{1}{B_i} \sum_{b=1}^{B_i} I(\hat{y}_{i,\text{OOB}}^{(b)} \neq y_i) \right)$, where $\hat{y}_{i,\text{OOB}}^{(b)}$ is the out-of-bag prediction for the i-th observation made by the b-th tree, and y_i is the true class label. The indicator function

$I(\cdot)$ equals 1 if the prediction matches the true label and 0 otherwise.This provides an unbiased estimate of test error and performs similarly to `leave-one-out cross-validation (LOOCV)`.

2.3.6 How Random Forest Works

Random Forest build on bagging by adding randomness during tree construction to reduce correlation between trees and improve predictive perfomance. Like bagging, each tree is trained on a bootstrap sample of the data. However, instead of considering all p predictors at each split, Random Forests randomly select a subset of m predictors. A common heuristic is to set $m \approx \sqrt{p}$, which ensures that a small portion of the features is evaluated at each node. If $m = p$, the method becomes standard bagging. Each tree's structure is influenced by a random vector Θ_k, which guides how the tree is built such as, variable selection and split criteria. [6]. Each tree acts as a classifier $h(X, \Theta_k)$, where X is the input. The final prediction $\hat{Y}$ is made by majority vote: $\hat{Y} = \arg\max_c \sum_{k=1}^{K} I(h(X, \Theta_k) = c)$, where $I(\cdot)$ is the indicator function and c indexes class labels. Out-of-bag (OOB) observations, which are left out of each bootstrap sample, are used to estimate test error without needing cross-validation.

2.4 Model Evaluation Metrics

A **`Receiver Operating Characteristic (ROC)`** curve is a graphical, nonparametric method used to assess the accuracy of a binary classification model. To understand the ROC curve, it helps to start with the **`confusion matrix`**, which summarizes prediction outcomes based on a fixed cutoff. In this study, I focused on the 2x2 version of the confusion matrix for binary outcomes.

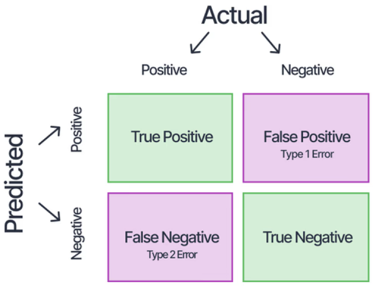

Figure 1: Confusion matrix for binary classification and a specific decision threshold

A prediction is a **`true positive (TP)`** when a positive outcome is correctly identified and a **`false positive (FP)`** if a positive outcome is incorrectly predicted. Similarly, correctly predicting a negative outcome results in a **true negative (TN)**, while incorrectly predicting a negative outcome is a **false negative (FN)**. Since models like logistic regression output probabilities, we need to choose a threshold to convert those into binary predictions. The **ROC** curve captures the trade-off between correctly identifying positive cases and incorrectly classifying negatives as positives as this cutoff shifts. **Sensitivity** (also called the **true positive rate** or **recall**) measures the proportion of actual positive cases that are correctly identified by the model. It is defined as **sensitivity** $= \frac{TP}{TP+FN}$. **Specificity** measures the proportion of actual negative cases correctly identified and is given by **specificity** $= \frac{TN}{TN+FP}$. A ROC curve is a connected line graph that plots sensitivity against $1 -$ **specificity** for a range of cutoff values. The diagonal reference line in the ROC plot represents random guessing, and any model performing above this line is better than chance.

Precision, also called Positive Predictive Value (PPV), measures the proportion of true positives (TP) among all predicted positives. It is defined as Precision $= \frac{TP}{TP+FP}$. A high precision means most patients predicted to progress actually did progress. In contrast, Negative Predictive Value (NPV), is the proportion of true negatives out of all predicted negatives: NPV $= \frac{TN}{TN+FN}$.

The **F1 Score** is the harmonic mean of **Precision** and **Recall**, and serves as a single metric that balances the trade-off between these two. It ranges between 0 and 1. A model with high precision but low recall may miss many true positive cases, while a model with high recall but low precision may include too many false alarms. The F1 score combines both into one robust measure: It is defined as F1 $= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$

Accuracy is one of the most commonly used metrics for evaluating classification models. It measures the proportion of total correct predictions (both positive and negative) out of all predictions made. Formally, it is defined as: Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$. A high Accuracy score is thought to be effective at distinguishing between patients who will progress and those who will not. However, accuracy can be misleading when the number of progressors greatly exceeds that of non-progressors. In such cases, other metrics like Precision, Recall, Balanced Accuracy and the F1 Score may provide a more reliable evaluation of model performance.

The **Area Under the Curve (AUC)** summarizes the performance of a classification model by measuring the area under its ROC curve. In this study, AUC was used to evaluate how well each model distinguishes between patients who progressed and those who did not. A higher AUC indicates better discriminatory ability. An

AUC greater than 0.5 suggests that the model performs better than random guessing, with values between 0.7 and 0.8 considered acceptable, and those above 0.9 indicating high discrimination [3].

`k-Fold Cross-Validation:` To perform cross-validation, the data are first split into K equally sized folds. For each fold $k = 1, \dots, K$, the k-th fold is used as the validation set, and the remaining $K - 1$ folds as the training set. The model is fit on the training data to obtain $\hat{f}_{(-k)}$. Predictions are then made on the validation set: $\hat{f}_{(-k)}(x_i)$ for $i \in \text{fold}_k$. The validation error for fold k is computed as $\text{CVErr}_k = \frac{1}{n_k} \sum_{i \in \text{fold}_k} L(y_i, \hat{f}_{(-k)}(x_i))$, where $L(\cdot, \cdot)$ is a loss function and n_k is the number of observations in fold k. After looping through all folds, the average cross-validation error across all folds is computed as $\text{CVErr} = \frac{1}{K} \sum_{k=1}^{K} \text{CVErr}_k$. [5]

Guiding principle: In general, with a multistep modeling procedure, cross-validation must be applied to the entire sequence of modeling steps. In particular, samples must be *left out* before any selection or filtering steps are applied [5].

3 Data Collection and Description

The dataset included 60 `IPF` patients monitored over 80 weeks, with proteomic profiling across 1,129 covariates [1]. There were no missing values in the dataset. Notable demographic variables include `Male` (binary: 1 = male, 0 = female) and `Smoke_Status` (categorical: `Never`, `Current`, `Past`). I performed exploratory data analysis (EDA) and found that out of the 60 participants, 35 (58%) experienced disease progression.

4 Phase 1: Low-Dimensional Predictive Modeling

In this phase of the analysis, I focused on evaluating predictive performance using a curated set of 14 covariates. These included six biomarkers identified by Ashley et al. (2016) [1] as being relevant to disease progression: `FCN2`, `Cathepsin_S`, `LGMN`, `VEGF_sR2`, `ICOS`, and `TRY3`, along with eight additional variables that I randomly selected. By narrowing the input space to a low-dimensional set, this phase allowed me to fairly compare different modeling approaches in a simplified setting. I applied three classification algorithms: Decision trees, Random forests, and standard logistic regression. I split the data into 70% for training and 30% for testing, using a fixed random seed to keep results consistent. The two demographic variables, `Male` and `Smoke_Status`, were also included. I also noticed that the final row of the `Cytochrome_c` variable contained an extreme outlier, about ten times higher than typical values, which could skew model performance. To handle this, I removed the outlier value and imputed a new value using the mean of the `Cytochrome_c` column, not because of missingness, but to reduce the influence of the outlier and improve robustness in this phase. I generated a correlation heatmap, which showed mostly low to moderate correlations among variables. The strongest was between `Cadherin_E` and `a2_Macroglobulin` (r = 0.52), suggesting minimal redundancy in the information each variable contributes.

4.1 Decision Tree

I began by training a decision tree classifier on the same training and test datasets described earlier. As a simpler model, the decision tree provided a baseline for evaluating predictive performance. Using the `rpart` package, I fitted a classification tree on the training data and applied cost-complexity pruning to control model complexity and reduce overfitting. I used 10-fold cross-validation through the `caret` package to tune the complexity parameter (`cp`), which determines the minimum improvement in model fit required for a split to be retained. The best performance was achieved at `cp` = 0.421, as shown in Figure 2 which was then used to fit the final pruned tree.. The pruned model achieved a training accuracy of 95.2%, misclassifying only 2 out of 42 observations, corresponding to a 4.8% error rate. This suggests a good fit while avoiding the complexity of the full tree.

4.1.1 Prediction and Evaluation

The confusion matrix for the pruned decision tree is shown in table 2. The model correctly classified 10 out of 11 progression cases and 2 out of 7 non-progression cases. This yielded a sensitivity of 90.9% and a specificity of 28.6%, indicating that while the model was highly effective at identifying true positive cases, it struggled to correctly classify non-progressors, as seen in table 1. The overall test accuracy was 66.7%, with a test error rate of 33.3%. The `Positive Predictive Value (PPV)` and `Negative Predictive Value (NPV)` were both 66.7%, suggesting moderate reliability in both progression and non-progression predictions. The `Balanced Accuracy` was 59.7%, reflecting imbalanced class performance. The `F1 score` for the positive class was 0.769, indicating a good balance between precision and recall in identifying progressors. The ROC curve in Figure 3 stayed above the diagonal, indicating that the model performed better than random guessing, but only marginally. With an `AUC` of 0.61, the pruned decision tree correctly ranks a progressor above a non-progressor about 61% of the time. This reflects limited discriminative ability. While the model maintained strong sensitivity, it continued to struggle with specificity, reinforcing earlier observations from the confusion matrix. The curve's sharp shape and early plateau further illustrate the model's bias toward correctly identifying progression cases while misclassifying many non-progressors.

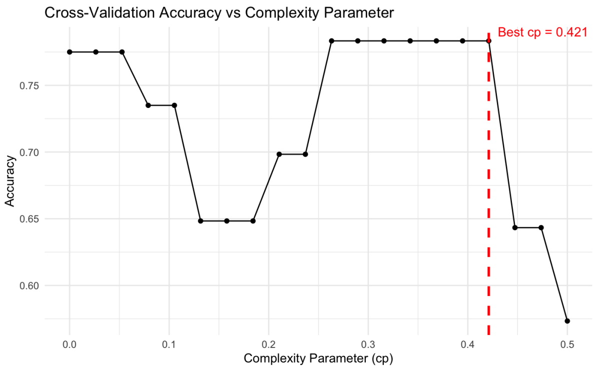

Figure 2: Cross-validation error by tree size.

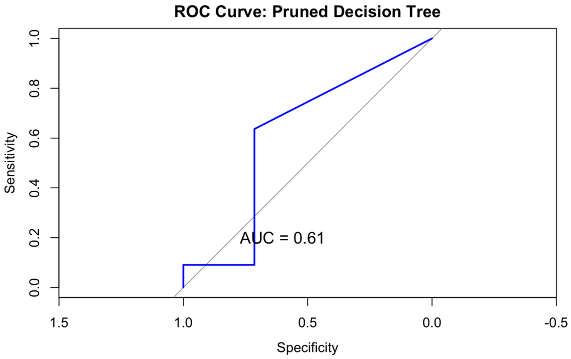

Figure 3: ROC curve for the pruned decision tree model

4.2 Random Forest

Random Forest was implemented using the `randomForest` package in R. I began by deciding how many trees the model should use. Using too few trees can cause the model's error rates to vary considerably, making it difficult to assess performance reliably. Increasing the number of trees helps stabilized these estimates. In this analysis, models were trained on 70% of the data, and the choice of `ntree` = 500 was guided by the Out-of-Bag (OOB) error. The OOB error plot in Figure 4 indicated that the error rate stabilized around 500 trees. Although adding more trees generally improves performance, the gains diminish beyond a certain point. In this case, adding trees beyond 500 did not lead to meaningful improvement, making 500 a practical choice that balances accuracy and computational efficiency. Once I settled on the number of trees, I tuned the `mtry` parameter, which controls how many predictors are randomly chosen at each split. If too many are used, the trees can end up looking too similar, which reduces the benefit of having many trees. If too few are used, the splits might not be very useful, which can hurt performance. To find an optimal balance, a range of `mtry` values was tested and their corresponding Out-of-Bag (OOB) errors were evaluated. As shown in Figure 5, the lowest out-of-bag (OOB) error (approximately 0.215) was achieved when using 4 features at each split. Since this configuration clearly outperformed the others, `mtry` = 4 was selected for the final model. Choosing the slightly smaller value introduces additional randomness into the trees, which can improve generalization and reduce the risk of overfitting.

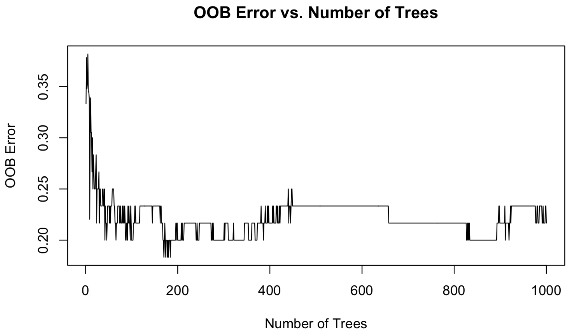

Figure 4: OOB error vs. number of trees

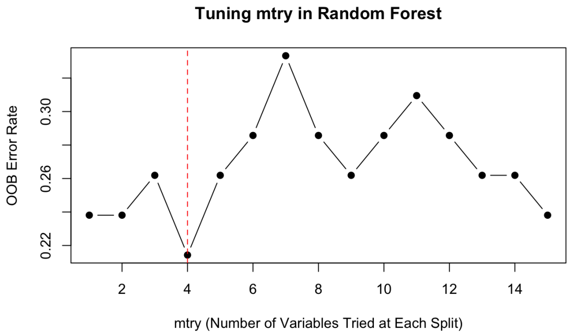

Figure 5: OOB error across different `mtry` values

After choosing the number of trees (`ntree` = 500) and tuning `mtry` to 4, I trained the final model and evaluated its performance using OOB error rates. Figure 6 illustrates how the error changed as more trees were added to the forest. The y-axis represents the proportion of OOB observations misclassified at each stage of the ensemble. Initially, with one tree, the error is highest because only a single tree has been trained, and OOB predictions are based on limited information. With two trees, each observation may receive one or two votes (depending on its inclusion in the OOB sample), making error estimates somewhat more informed but still unstable. The black curve in Figure 6 shows overall OOB error rate across all numbers of trees. It decline sharply within the first 50 to 100 trees, followed by a gradual plateau. After 200 trees, the OOB error stabilizes around 0.238, indicating that adding more trees does not help much. The red dashed line shows the OOB error for class 0 (non-progressors), which fluctuates between 0.3 and 0.4, suggesting difficulty in correctly identifying non-progressors. In contrast, the green dotted line for class 1 (progressors) drops quickly below 0.15 and remains stable, indicating the model is more confident and accurate in predicting progression events. This asymmetry is also evident in the confusion matrix: the class error for non-progressors is 0.389, while for progressors it is only 0.125. Overall, the model appears more effective at identifying patients at risk of progression.

4.2.1 Prediction and Evaluation

The random forest model performed well. Its out-of-bag (OOB) error rate was 0.2381, which aligns with the OOB error plot, particularly the black curve that flattens after about 200 trees. On the held-out test set, the model achieved a test error rate of 0.167. It was much better at predicting who would progress (class 1), correctly identifying all such cases in the test set, with a sensitivity of 100%. This aligns with the green curve in the plot, which drops quickly and stays low. On the other hand, it did not do as well with non-progressors (class 0), misclassifying 3 out of 7, leading to a higher error rate for that class, as evident in Figure 6. This shows that the model was better at classifyinge progression than non-progression. From table 1, the model achieved 83.3% accuracy, correctly classifying 15 out of 18 patients in the held-out test set. The sensitivity (recall) was 100%, indicating it was excellent at identifying those who progressed, while the specificity was 57.1%, showing that it missed several non-progressors. The model also had a solid positive predictive value (PPV) of 78.6% and a perfect negative predictive value (NPV) of 100%. The F1 score was 88%, showing a strong balance between precision and recall. To see how well the final Random Forest model separates progressors from non-progressors, I looked at the ROC curve in Figure 7. The AUC was 0.984, which shows the model does a great job overall. A value close to 1 means it's very effective at distinguishing between the two groups. The curve rises quickly at first, meaning the model catches most true progressors without many false alarms. But as the threshold lowers, the curve flattens, showing more false positives come in as we try to catch even more true positives. This highlights the trade-off between sensitivity and specificity. Overall, the ROC curve, AUC, and earlier results all point to a model that is very good at finding progressors, though it's not as strong at ruling out non-progressors.

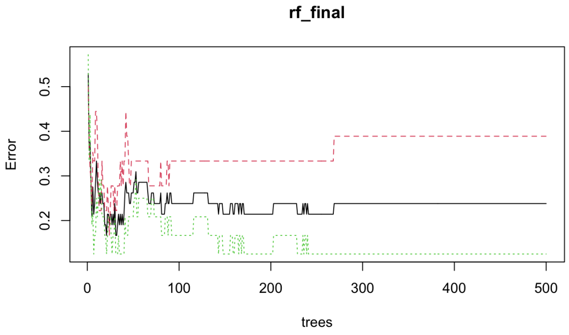

Figure 6: OOB error vs. trees: overall (black), non-progression (red), progression (green)

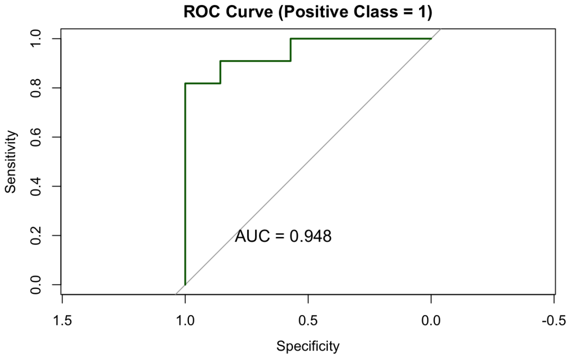

Figure 7: ROC curve for the Random Forest model

4.3 Logistic Regression

To assess predictive performance using a linear classification approach, I trained a logistic regression model on the same training and test datasets used previously. The model was fit using the `glm()` function from base R, with `family = "binomial"` to specify logistic regression. Multicollinearity was assessed using adjusted Generalized Variance Inflation Factors (GVIFs) from the `car` package, which accounts for categorical predictors such as `Smoke_Status` [7]. All predictors in the model had adjusted GVIF values below 5, indicating no serious multicollinearity issues. When evaluated on the training set, the model achieved an accuracy of 92.9% and a classification error of 7.1%, correctly classifying 22 out of 24 progression cases and 17 out of 18 non-progression cases.

4.3.1 Prediction and Evaluation

As shown in table 2, the logistic regression model correctly classified 8 out of 11 progression cases and 4 out of 7 non-progression cases. This resulted in a test accuracy of 66.7%, with a corresponding test error rate of 33.3%, as shown in table 1. The model achieved a sensitivity of 72.7% and a specificity of 57.1%, indicating that it was better at detecting true progression cases than at identifying non-progressors. The precision (positive predictive value) was also 72.7%, meaning that most predicted progression cases were correct. The F1 score was 0.727, reflecting a reasonable balance between precision and recall. The area under the ROC curve (`AUC`) is `0.727`, suggesting the model has moderate ability to distinguish between progression and non-progression cases.

4.4 Model Comparison Summary

Metric	**Decision Tree**	**Random Forest**	**Logistic Regression**
Accuracy	0.6667	0.8333	0.6667
Sensitivity	0.9091	1.0000	0.7273
Specificity	0.2857	0.5714	0.5714
Positive Predictive Value	0.6667	0.7857	0.7273
Negative Predictive Value	0.6667	1.0000	0.5714
F1 Score	0.7690	0.8800	0.7270
Balanced Accuracy	0.5974	0.7857	0.6494

Positive Class: 1

Table 1: Comparison of Performance Metrics Across Models

Among the three models evaluated, the random forest achieved the highest overall performance. As shown in table 1, it recorded the highest accuracy (83.3%) and perfect sensitivity (100%), correctly identifying all

Actual / Predicted	**Decision Tree**		**Random Forest**		**Logistic Regression**	
	No Prog	Prog	No Prog	Prog	No Prog	Prog
No Progression	2	5	4	3	4	3
Progression	1	10	0	11	3	8

Table 2: Confusion Matrices for Decision Tree, Random Forest, and Logistic Regression

progression cases. It also had the highest F1 score (0.88) and balanced accuracy (0.7857), indicating a strong balance between precision and recall. However, its specificity remained moderate at 57.1%, reflecting some misclassification of non-progressors. The decision tree also showed strong sensitivity (90.9%) but suffered from very low specificity (28.6%), leading to a balanced accuracy of only 59.7%. It misclassified five out of seven non-progression cases, as seen in table 2, suggesting a tendency to over-predict progression. Logistic regression demonstrated more balanced behavior, with sensitivity and specificity both at 72.7% and 57.1%, respectively. Its F1 score (0.727) and balanced accuracy (0.6494) were moderate. However, it misclassified three progression and three non-progression cases, reflecting less consistent performance than the random forest. Overall, the random forest offered the best trade-off between sensitivity and specificity, while the decision tree was highly sensitive but less discriminative for non-progression. Logistic regression maintained modest balance but lagged behind in predictive accuracy.

5 Phase 2: High-Dimensional Predictive Modeling

When the number of predictors exceeds the number of observations, building models that generalize well is challenging. In this phase, I worked with 60 observations and 1,129 predictors, a setting where overfitting is a major concern. To ensure fair evaluation, I split the data into 70% training and 30% testing, using a fixed random seed for reproducibility. All model development and tuning were done on the training set, while the test set was used only for final evaluation. This provided an honest estimate of each model's ability to generalize. I applied two strategies: `Lasso logistic regression` and `Random Forest`, both of which are well-suited for high-dimensional data. Lasso performs variable selection through regularization, while Random Forest handles complexity through ensembling and embedded feature selection.

5.1 LASSO Logistic Regression

To manage the large number of predictors and reduce overfitting, I started with Lasso logistic regression. This method is well-suited for high-dimensional settings because it performs variable selection by shrinking some coefficients to zero, simplifying the model and highlighting the most relevant predictors. To choose the best λ, I fitted the model using ten-fold cross-validation on the training data and examined the plot of misclassification error across different values of the regularization parameter λ shown in Figure 8. Each red dot represents the average error at a given λ, and the gray bars indicate variability. Two vertical lines mark key values: `lambda.min` = 0.03098, which gives the lowest average error, and `lambda.1se` = 0.05941, which yields the simplest model within one standard error of the minimum. The numbers at the top of the plot indicate how many predictors were selected at each λ. Since my goal was to build a model that generalizes well to new data, I chose `lambda.1se`, which tends to produce more stable models—especially valuable with small datasets. Using `lambda.1se`, I fit the Lasso model and made predictions on both the training and test sets. On the test set, the misclassification error was 0.5556, meaning just under half of the predictions were incorrect. The confusion matrix in table 4 showed a sensitivity of 0.4615 and specificity of 0.4000, indicating difficulty in correctly identifying both progressors and non-progressors. The accuracy was 44%, and the AUC was 0.5077—close to random guessing. These results suggest the model did not generalize well, likely due to weak signal in the predictors or the limited sample size. Given this performance, I proceeded to explore Random Forest

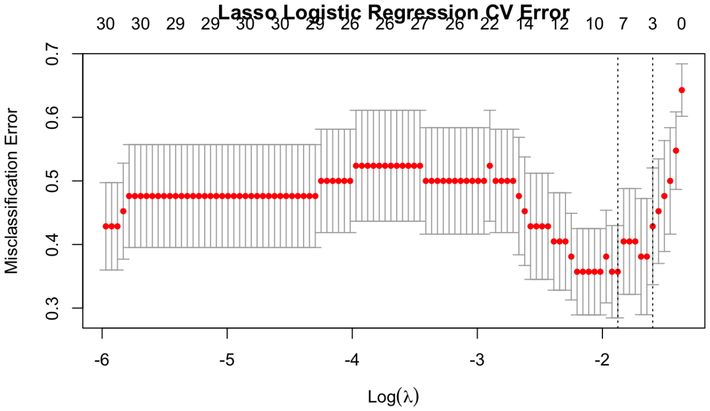

Figure 8: Cross-validation curve for Lasso logistic regression showing misclassification error

5.2 Random Forest

I used `Random Forest` as a flexible, non-linear model that works well in high-dimensional settings without assuming a specific relationship between predictors. Before fitting the final model, I tuned two important parameters: the number of trees (`ntree`) and the number of predictors randomly chosen at each split (`mtry`). I tested `ntree` values from 100 to 1500 and found that the out-of-bag (OOB) error dropped sharply before leveling off around 500 trees, so I chose `ntree = 500` as a good balance between performance and efficiency. For `mtry`, I tested values from 10 to 60. Although the typical default is around $\sqrt{p} \approx 34$, `mtry = 50` gave the best performance shown in Figure 9. This higher value likely improved the model's ability to find strong splits, especially in a noisy dataset with many weak or correlated predictors. After selecting `ntree = 500` and `mtry = 50`, I trained the final Random Forest model. Figure 10 displays the OOB error across 500 trees. The black line shows the overall error rate, which stabilizes around 0.45. The green dotted line represents the error for predicting progression cases, remaining mostly below 0.40, while the red dashed line corresponds to non-progression errors, fluctuating above 0.60. This suggests the model was better at identifying progression (class 1) than non-progression (class 0). Evaluating it on the test set, the test error was `0.3889`—lower than Lasso. The model showed balanced classification performance, with a sensitivity of `0.6154` and a specificity of `0.6000`. It achieved an `F1 score` of `0.842`, reflecting good precision and recall, and an `AUC` of `0.5692`, indicating modest but better-than-chance discrimination – table 3.

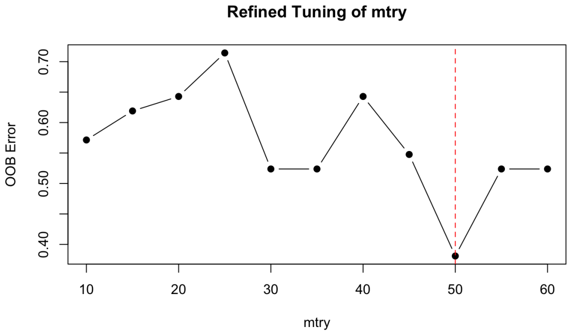

Figure 9: OOB error by number of predictors used at each split.

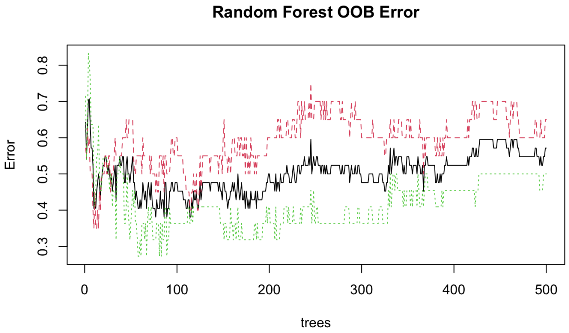

Figure 10: OOB error by class for Random Forest model.

After testing both models on the held-out test set, `Random Forest` outperformed `LASSO` across nearly every metric. It had a lower test error of 0.3889 compared to 0.5556 for LASSO, meaning it made fewer incorrect

Metric	LASSO	Random Forest
Test Error	0.5556	0.3889
Accuracy	0.4444	0.6111
Sensitivity	0.4615	0.6154
Specificity	0.4000	0.6000
Positive Predictive Value	0.6667	0.8000
Negative Predictive Value	0.2222	0.3750
F1 Score	0.5556	0.8420
Balanced Accuracy	0.4308	0.6077
AUC	0.5077	0.5692
Positive Class	1	1

Table 3: Comparison of Performance Metrics for LASSO and Random Forest

Actual / Predicted	**LASSO**		**Random Forest**	
	No Prog	Prog	No Prog	Prog
No Progression	2	3	3	2
Progression	7	6	5	8

Table 4: Confusion Matrices for LASSO and Random Forest on Test Data

predictions overall. Looking at the ability to identify progression cases, Random Forest achieved a sensitivity of 0.6154, while LASSO reached 0.4615. This shows that Random Forest was more successful in detecting true progression cases. For non-progression, Random Forest had a specificity of 0.6000, whereas LASSO had a lower value of 0.4000, indicating that LASSO was more likely to misclassify non-progression cases. The F1 score, which balances precision and recall, was 0.842 for Random Forest and 0.5556 for LASSO. This highlights the greater stability and reliability of Random Forest's predictions. In terms of positive predictive value, Random Forest predicted the progression class correctly 80% of the time compared to 66.7% for LASSO. The negative predictive value was lower for both models but still better for Random Forest at 0.3750, compared to 0.2222 for LASSO. Balanced accuracy, which averages sensitivity and specificity, was 0.6077 for Random Forest and 0.4308 for LASSO, further showing that Random Forest performed more consistently across both classes. Finally, although both models had modest AUC scores, Random Forest still showed a slight advantage with an AUC of 0.5692 compared to 0.5077 for LASSO. Overall, Random Forest captured the patterns in the data more effectively and provided more accurate and balanced predictions in this high-dimensional setting.

6 Final Reflection

This project gave me the chance to work with real clinical data and explore how predictive modeling can be applied in both low and high dimensional settings. In the low-dimensional phase, I saw how simpler models can perform well when using a small, well-chosen set of predictors. In the high-dimensional phase, where predictors far outnumbered observations, I had to be more careful to avoid overfitting. This taught me the importance of regularization and model tuning, especially when working with complex data. I used separate test set for both phases to evaluate performance fairly and understand how models behave on unseen data. Beyond the technical work, this project helped me build confidence in managing the full machine learning pipeline, from data prep and feature selection to model evaluation. Most importantly, it deepened my understanding of how to think critically about modeling choices and their impact. I'm confident I met all three objectives and grew as a practitioner through hands-on learning.

References

[1] Ashley, S. L., Xia, M., Murray, S., et al. (2016). "Six-SOMAmer Index Relating to Immune, Protease and Angiogenic Functions Predicts Progression in IPF." *PLOS ONE*, 11(8), e0159878. Available online

[2] Casella, G., and Berger, R. L. (2002). *Statistical Inference* (2nd ed.). Available online

[3] Hosmer, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). *Applied Logistic Regression.* Available online

[4] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R.* New York: Springer. Available online

[5] Jacobson, T. (2025). *Special Topics (ST_599)* [Lecture notes]. Dept. of Statistics, Oregon State University.

[6] Breiman, L. (2001). "Random Forests." *Machine Learning*, 45(1), 5–32. Available online

[7] Fox, J., & Monette, G. (1992). "Generalized Collinearity Diagnostics." *Journal of the American Statistical Association*, 87(417), 178–183. Available online