https://doi.org/10.26219/heal.aueb.6189

Statistical and Machine Learning Regularization Techniques in Clinical Biostatistics: A comprehensive evaluation

By
Panagiotis S. Stamatis

A THESIS
Submitted to the Department of Statistics
of the Athens University of Economics and Business
in partial fulfilment of the requirements for
the degree of Master of Science in Statistics

Athens, Greece
May 2024

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

Μεθόδοι Κανονικοποίησης Στατιστικής και Μηχανικής Εκμάθησης στην Κλινική Βιοστατιστική: Μια ολοκληρωμένη αξιολόγηση

Παναγιώτης Σ. Σταμάτης

ΔΙΑΤΡΙΒΗ
Που υποβλήθηκε στο Τμήμα Στατιστικής
του Οικονομικού Πανεπιστημίου Αθηνών
ως μέρος των απαιτήσεων για την απόκτηση
Διπλώματος Μεταπτυχιακών Σπουδών στην Στατιστική

Αθήνα, Ελλάδα
Μάιος 2024

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

Dedication

In memory of my mother, whose love, strength, and guidance have shaped me into the person I am today.

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

Acknowledgements

I extend my sincerest gratitude to professor Vasilis Vasdekis, my esteemed supervisor, whose invaluable guidance, insightful feedback, and wealth of expertise played a pivotal role in the completion of this thesis. Additionally, I am deeply thankful to all the professors within the Masters program whose unwavering commitment to excellence has enriched my academic journey.

Last but certainly not least, I wish to express my heartfelt appreciation to my family and friends for their unwavering support, encouragement, and understanding throughout my pursuit of this Master's degree.

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

Abstract

The primary objective of this thesis is to investigate the efficacy of regularization techniques within the domain of clinical biostatistics. A comprehensive exploration of statistical and machine learning methodologies, including Penalization, Early Stopping, and Ensembling, is undertaken. Regularization, as defined, serves to control model complexity by incorporating additional information to address ill-posed problems or mitigate overfitting. Despite its conceptual clarity, the full extent of its applicability and diverse variants remains not entirely elucidated. Leveraging the R software, these techniques are applied to two distinct clinical datasets, both pertinent to prostate cancer research.

The first dataset aims to classify patients into benign or malignant tumor categories, wherein Penalization, specifically Ridge Regression, demonstrates superior performance compared to alternative methods, achieving the lowest Misclassification Error (MCE) and highest Area Under the Curve (AUC). Furthermore, the second dataset endeavors to predict the logarithm of prostate-specific antigen (PSA), a significant biomarker, in conjunction with other clinical predictors. Once more, the penalization approach, notably Elastic Net, exhibits notable performance by yielding the lowest Mean Squared Error (MSE) and Mean Absolute Error (MAE). However, the outcomes for machine learning techniques are less promising, potentially attributable to the inherently simple data relationships or issues related to dimensionality. Overall, the study underscores the utility of regularization methods in enhancing predictive accuracy within clinical biostatistics, advocating for their broader adoption and further exploration within this domain.

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

Περίληψη

Ο βασικός στόχος αυτής της διατριβής είναι η εξέταση της αποτελεσματικότητας των τεχνικών κανονικοποίησης στον τομέα της κλινικής βιοστατιστικής. Πραγματοποιείται μια περιεκτική εξερεύνηση των στατιστικών και μηχανικής μάθησης μεθοδολογιών, συμπεριλαμβανομένων της τιμώρησης, της πρόωρης διακοπής και της συνδυασμένης μάθησης. Η κανονικοποίηση, όπως ορίζεται, εξυπηρετεί τον έλεγχο της πολυπλοκότητας του μοντέλου με την ενσωμάτωση επιπλέον πληροφοριών για την αντιμετώπιση ασαφών προβλημάτων ή τη μείωση του υπερ-εκπαιδευτικού φαινομένου. Παρά την έννοια της κανονικοποίησης, η πλήρης κατανόηση της εφαρμοσιμότητάς της και των διαφορετικών εκδοχών της παραμένει ακόμα μη ξεκάθαρη. Με χρήση του λογισμικού R, αυτές οι τεχνικές εφαρμόζονται σε δύο διαφορετικά κλινικά σύνολα δεδομένων, τα οποία είναι σχετικά με την έρευνα του προστάτη καρκίνου.

Το πρώτο σύνολο δεδομένων στοχεύει στην ταξινόμηση των ασθενών σε καλοήθη ή κακοήθη όγκους, όπου η τεχνική τιμώρησης, και ειδικότερα η Ridge Regression, επιδεικνύει υψηλή απόδοση σε σύγκριση με τις εναλλακτικές μεθόδους, επιτυγχάνοντας το χαμηλότερο Missclassifcation Error (MCE) και την υψηλότερη Area Under the ROC Curve (AUC). Επιπλέον, το δεύτερο σύνολο δεδομένων στοχεύει στην πρόβλεψη του λογαρίθμου του ειδικού για τον προστάτη αντιγόνο (PSA), ενός σημαντικού βιοδείκτη, και την σχέση του με άλλες κλινικές μεταβλητές. Και πάλι, η μέθοδος της τιμώρησης, και ειδικότερα το Elastic Net, επιδεικνύει σημαντική απόδοση παρέχοντας το χαμηλότερα Mean Square Error (MSE) και Mean Absolute Error (MAE). Απροσδόκητα, τα αποτελέσματα για τις μεθόδους

https://doi.org/10.26219/heal.aueb.6189

μηχανικής μάθησης δεν είναι τόσο ελπιδοφόρα, πιθανώς λόγω των απλών σχέσεων δεδομένων ή προβλημάτων που σχετίζονται με τη διάσταση.

Συνολικά, η μελέτη υπογραμμίζει την χρησιμότητα των μεθόδων κανονικοποίησης στη βελτίωση της προβλεπτικής ακρίβειας στην κλινική βιοστατιστική, προωθώντας την ευρύτερη υιοθέτηση και περαιτέρω εξερεύνηση αυτών των μεθόδων σε αυτόν τον τομέα.

https://doi.org/10.26219/heal.aueb.6189

Contents

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

List of Figures

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

List of Tables

https://doi.org/10.26219/heal.aueb.6189

https://doi.org/10.26219/heal.aueb.6189

List of Abbreviations

1. AUC: **A**rea **U**nder the ROC **C**urve
2. IG: **I**nformation **G**ain
3. MAE: **M**ean **A**bsolute **E**rror
4. MCE: **M**iss **C**lassification **E**rror
5. MSE: **M**ean **S**quare **E**rror
6. OOB: **O**ut **O**f **B**ag
7. TNR: **T**rue **N**egative **R**ate
8. TPR: **T**rue **P**ositive **R**ate

https://doi.org/10.26219/heal.aueb.6189

Chapter 1

Introduction

In today's digital age, we find ourselves inundated with vast volumes of data, giving rise to concepts like «Big Data» that have become integral to our daily lives. This data revolution necessitates effective management and analysis to avoid its relegation to mere digital oblivion. Machine learning techniques and statistical methods have emerged as crucial tools in addressing this need. Among these techniques, a category known as regularization plays a pivotal role. While the concept of regularization is not new, it has gained renewed significance in the context of multidimensional data.

Regularization techniques serve the purpose of controlling model complexity by incorporating additional information to address ill-posed problems or mitigate the risk of overfitting. This encompasses diverse approaches such as penalization, early stopping, and ensemble methods, each serving distinct purposes. Despite the longevity of these methods, a clear understanding of their applicability and nuances remains elusive.

Penalization techniques introduce a penalty term to be minimized alongside the subjective loss function, effectively balancing model complexity with model fit. The essence of early stopping lies in the strategic cessation of an iterative process, aligning with the goal of harmonizing model simplicity and model fit. Lastly, en-

https://doi.org/10.26219/heal.aueb.6189

sembling methodologies harness the power of multiple models, orchestrating their combined efforts to mitigate prediction variability. Our study will encompass a spectrum of statistical methods, including Ridge Regression, LASSO, Elastic Net, and machine learning algorithms such as Decision Trees, Random Forest, and Gradient Boosted Trees. While these techniques are widely employed by scientists and researchers, their inherent role as regularization methods is often underappreciated.

The primary aim of this thesis is to bridge this knowledge gap by exploring the realm of regularization techniques within the context of clinical biostatistics and healthcare data. This field is of particular interest due to the infrequent utilization of such methods in practical applications. The practical implementation and assessment of these techniques will be carried out using the R statistical software.

The structure of this dissertation is as follows: Chapter 2 provides a comprehensive literature review, commencing with an overview of Generalized Linear Models and delving into theoretical aspects of regularization, encompassing Penalization, Early stopping, and Ensemble learning. Chapters 3 and 4 focus on the practical implementation of these approaches, with a specific focus on analyzing data related to prostate cancer patients. Chapter 5 offers concluding insights, discusses the limitations of this thesis, and outlines potential avenues for future research.

The code implemented to apply and evaluate the regularization techniques for the two datasets is largely similar, given that the same packages and framework were utilized. Consequently, specific adjustments in the algorithms were necessary to accommodate the differing tasks of regression and classification. For the sake of brevity, representative R code pertaining to the prostate cancer dataset II can be in the Appendix.

Chapter 2

Regularization Techniques

2.1 Generalized Linear Models

According to McCullagh and Nelder (1989), generalized linear models (GLMs) is a statistical framework that extends the linear regression model to accommodate a broader range of response variable types, including continuous, binary, count, and categorical data. They consist of three components:

- Random Component: This specifies the response variable y and its probability distribution. The observations $y = (y_1, y_2, ..., y_n)^T$ on that distribution are treated as independent.
- Linear Predictor or Systematic Component: For a parameter vector $\beta = (\beta_1, \beta_2, ..., \beta_p)^T$ and a $n \times p$ model matrix X that contains values of p explanatory variables for the n observations, the linear predictor is $X\beta$.
- Link Function: This is a function g applied to each component $E(y)$ that relates it to the linear predictor, $g[E(y)] = X\beta$.

In practise, we see a wide range of response variables with a wide range of associated distributions:
GLMs form a class of statistical models for response variable whose distribution

https://doi.org/10.26219/heal.aueb.6189

Response Type	Range	Distribution
Continuous	(-∞, ∞)	Normal(μ, σ^2)
Binary	{0,1}	Bernoulli(π)
Polytomous	{1,2,...,K}	Multinomial(π_κ)
Count	{0,1,...,n}	Binomial(n,π)
Count	{0,1,...}	Poisson(λ)
Continuous	(0,∞)	Gamma(α, β)
Continuous	(0,1)	Beta(α, β)

Table 2.1: Basic distributions

belongs to the exponential dispersion family, with a probability density or mass function of the form:

$$f_Y(y;\theta,\phi) = exp\{\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\}$$

where θ is the canonical or natural or location parameter, ϕ is the dispersion parameter and $b(\theta)$ is the cumulant function. The canonical parameter has key relationships with both $E(Y)$ and $V(Y)$ while the dispersion parameter has a key relationship with $V(Y)$. When the dispersion parameter is known, we say that the distribution is a member of the exponential family.

Properties

Consider now the likelihood for a single observation from the exponential family

$$\mathcal{L}(\theta_i,\phi,y_i) = exp\{\frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi)\} \tag{2.1}$$

and the corresponding log-likehood

$$log\mathcal{L}(\theta_i,\phi,y_i) = \frac{y_i\theta_i - b(\theta_i)}{a(\phi)} + c(y_i,\phi). \tag{2.2}$$

The first partial derivative with respect to θ_i is the score or informant function for θ_i and is given by:

$$\frac{\partial}{\partial \theta_i}\lambda(\theta_i, \phi; y_i) = U(\theta) = \frac{y_i - b'(\theta_i)}{a\phi}. \tag{2.3}$$

Using general results from likelihood theory, we know that under appropriate regularity conditions satisfied by the exponential dispersion family:

$$E[U(\theta_i)] = 0 \quad \text{and} \quad V[U(\theta_i)] = E[U(\theta_i^2)] = -E[\frac{\partial U(\theta_i)}{\partial \theta_i}].$$

Since the score has mean zero, we find that $E[\frac{Y_i - b'(\theta_i)}{a_i(\phi)}] = 0$ and consequently that $E[Y_i] = \beta'(\theta_i)$. The second partial derivative respectively is

$$\frac{\partial^2}{\partial \theta_i^2}\lambda(\theta_i, \phi; y_i) = -\frac{b''(\theta_i)}{\alpha_i(\phi)} \tag{2.4}$$

which is the observed information for the canonical parameter from the i^{th} observation. This is also the expected information and using the above properties it follows that

$$V[U(\theta_i)] = V\left[\frac{Y_i - b'(\theta_i)}{\alpha_i(\phi)}\right] = \frac{b''(\theta_i)}{\alpha(\phi)}$$

so that

$$\frac{V(Y_i)}{\alpha_i^2(\phi)} = \frac{b''(\theta_i)}{\alpha(\phi)} \quad \text{hence} \quad V[Y_i] = b''(\theta_i)\alpha(\phi).$$

The variance of Y_i is therefore a function of both θ_i and ϕ. As proven before the canonical parameter is a function of μ_i, that is $\mu_i = b'(\theta_i) \Rightarrow \theta_i = \theta(\mu_i) = b'^{-1}(\mu_i)$ so we can write

$$V[Y_i] = \beta''(\theta(\mu_i))\alpha(\phi) \tag{2.5}$$

which is called the *variance function.* In summary, the function $b(\cdot)$ determines moments of y_i. This function is called the *cumulant function*, because when $a(\phi) =$

1 its derivatives yield the cumulants[1] of the distribution. For the exponential dispersion family the probability density function has the form of (2.1). The systematic component constitutes a means of specifying how the distribution depends on a vector of covariates X_i. In GLMs we model the conditional mean, $\mu_i = E(Y_i|X_i)$, which provides a connection between X_i and distribution of Y_i via the canonical parameter θ_i and the cumulant function $b(\theta_i)$. Specifically the relationship between μ_i and X_i is given by $g(\mu_i) = X_i^T\beta$, where we link the linear predictor to the distribution of Y_i via a transformation of μ_i. The link function $g(\cdot)$ transforms the conditional mean μ_i rather than the response variable Y_i. The inverse of the link function provides the specification of the model on scale of μ_i, $\mu_i = g^{-1}(X_i^T\beta)$. Link functions are therefore usually monotone and have a well-defined inverse. Recall that the mean and the canonical parameter are linked via the derivative of the cumulant function $E[Y_i] = \mu_i = b'(\theta_i)$. An important link function is the *canonical* link, $g(\mu_i) = \theta(\mu_i)$, which results by viewing the canonical parameter θ_i as a function of μ_i. It holds that $g(\cdot) = b'(\cdot)^{-1}$. In particularly, the most indicative examples are:

- In linear regression we specify $\mu_i = X_i^T\beta$ where $g(\cdot)$ is the identity link
- In logistic regression we specify $log(\frac{\mu_i}{1-\mu_i}) = X_i^T\beta$ where $g(\cdot)$ is the logit link
- In poisson regression we specify $log(\mu_i) = X_i^T\beta$ where $g(\cdot)$ is the log link

[1]The cumulants $\{\kappa_n\}$ are coefficients in a power series expansion of the log mgf, $log[E(e^{ty}] = \sum_{n=1}^{\infty} \kappa_n t^n/n!$. The moments determine cumulants and vice versa.

 https://doi.org/10.26219/heal.aueb.6189

2.2 Penalization

Penalization approaches aim to control the complexity of a model by adding a penalty term to the objective function that the model seeks to optimize. The basic principle underlying penalization is to achieve a good fit to the training data while preventing the model from becoming overly complex or overfitting, thus achieving the trade-off between model complexity and model fit. Penalization discourages essentially extreme parameter values in the model, favoring simplicity and generalizability in new unseen data.

Assuming we build regression-type parametric models, denoting the vector of the observed data as y and as θ the vector which comprises the regression coefficients, the penalization approaches can take in a frequentist and loss-based formulation the form:

$$L(y;\theta) + pen(\theta), \tag{2.6}$$

where $L(\cdot;\cdot)$ representing a subjective loss function and $pen(\cdot)$ a non-negative penalty term. A loss function quantifies the disparity between predicted and actual outcomes in a model. It guides optimization by representing the objective to minimize in predictive modeling, forming the foundation for techniques like penalization to balance complexity and fit. Some well-known loss function in statistics and machine learning are:

- Regression Losses
  - Mean Square Error/ Quadratic Loss/ L2 Loss : $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
  - Mean Absolute Error/ L1 Loss : $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
  - Mean Bias Error : $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$
- Classification Losses
  - Cross Entropy Loss/ Negative Log Likelihood: $-\sum_{i=1}^{n}(y_i \cdot log(\hat{y}_i) + (1-y_i)\cdot log(1-\hat{y}_i))$
  - Hinge Loss : $\sum_{i=1}^{n} max(0, 1 - y_i \cdot \hat{y}_i)$

https://doi.org/10.26219/heal.aueb.6189

Linear Regression

We consider a multivariate linear regression problem with p explanatory covariates and let

- x_i represents the vector of predictor variables associated with the $i-$th observation
- y_i represents the observed or true outcome for the $i-$th observation and
- $\hat{y}_i$ denotes the predicted or estimated outcome for the $i-$th observation based on the linear regression model,

we assume that the relationship between the dependent and independent variables is expressed as:

$$y_i = \theta^T X_i + \epsilon_i,$$

where ϵ_i is a mean 0 noise random variable and assuming the intercept being equal to zero (standardization). The method of least squares provides estimates of the parameters by minimization of the least squares objective function

$$\underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{N} \left(y_i - \theta^T X_i\right)^2 . \tag{2.7}$$

However there are cases, where this approach does not provide satisfactory results. For instance, this estimation procedure is a good one if $X^T X$, when in the form of a correlation matrix, is nearly a unit matrix. If this does not hold, the least squares estimates does not provide sensible results. Therefore one is forced to treat the estimated prediction function as black box or to drop predictors in order to destroy the correlation among the X_i which used to form the $X^T X$.

Moreover, in case of "big data" where $p > N$, the least squares estimates are not unique and there is a infinite set of solutions that make the objective function equal to zero yielding to overfitting. Consequently, the lack of the important aspects,

 https://doi.org/10.26219/heal.aueb.6189

i.e. prediction accuracy and interpretation of the model, demands a constraint or regularization, in the model process.

2.2.1 Ridge Regression

In accordance with Arthur E. et al. (1970), the Ridge estimator can give solution to some of the above difficulties and obtain a point estimate with a smaller mean square error. Specifically, Ridge Regression finds the solution of $\hat{\theta}$ to the optimization problem:

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \theta^T X_i\right)^2 \quad \text{subject} \quad \text{to} \quad \sum_{j=1}^{P} \theta_j^2 \leq t. \tag{2.8}$$

After applying Lagrange multipliers give rise to the ridge estimator:

$$\hat{\theta}_{Ridge} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T X_i)^2 + \lambda ||\theta||_2^2, \tag{2.9}$$

where $||\theta||_2^2 = \sum_{j=1}^{p} \theta_j^2$ is the Euclidean norm of the coefficient vector θ and the λ is the tuning parameter which controls the shrinkage. There is one-to-one correspondence between the parameters λ and t. This penalty is called the L_2 penalty. It can be proven that the Ridge Regression has a closed-form solution which ensembles the usual OLS estimator as:

$$\hat{\theta}_{Ridge} = \left(\mathbb{X}^T \mathbb{X} + \lambda \mathbb{I}_p\right)^{-1} \mathbb{X}^T \mathbb{Y}, \tag{2.10}$$

where $\mathbb{X}$ is the $n \times p$ data matrix and $\mathbb{I}_p$ is the $p \times p$ identity matrix. Typically in penalization approaches, we first standardize the predictors X so that each column is centered $\left(\frac{1}{N} \sum_{i=1}^{N} x_i = 0\right)$ and has unit variance $\left(\frac{1}{N} \sum_{i=1}^{N} x_i^2 = 1\right)$. In case the predictors are measured on the same scale, no such a process is needed. For convenience we also assume that the dependent variable has been centered too.

 https://doi.org/10.26219/heal.aueb.6189

Letting $\hat{\theta_{LS}} = (\mathbb{X}^T\mathbb{X}^{-1}\mathbb{X}^T\mathbb{Y})$ the estimator derived from the least square regression, the ridge estimations are very similar with the least squares coefficients, but just the coefficients are moved towards 0, since in the matrix inverse, there is an extra $\mathbb{I}_d$ term. That is, the ridge estimator shrinks the estimator towards 0.

2.2.2 LASSO

Tibshirani R. (1996) proposed a new technique called the LASSO, for "Least Absolute Shrinkage and Selection Operator" in order to deal with the limitations of the usual OLS approach. Nowadays, LASSO is one of the most popular penalized parametric regression model. Because of its appealing properties, it has radically influenced modern statistical research. LASSO finds the solution of $\hat{\theta}$ to the optimization problem:

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y_i - \theta^T X_i\right)^2 \quad \text{subject} \quad \text{to} \quad \sum_{j=1}^{P} |\theta_j| \leq t. \tag{2.11}$$

After applying Lagrange multipliers the LASSO estimator is:

$$\hat{\theta}_{LASSO} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta^T X_i)^2 + \lambda ||\theta||_1, \tag{2.12}$$

where $||\theta||_1 = \sum_{j=1}^{p}$ is the 1-norm of the coefficient vector θ. This penalty is called the L_1 penalty. As in Ridge Regression, the λ is called a tuning parameter and controls the shrinkage of the model. More about the λ is included in the remainder of this section.

The main difference between Ridge Regression and LASSO is LASSO yields sparse solutions since some coefficients estimates set to exactly zero, while Ridge tends to retain all predictors. Figure 2.1 contrasts the two constraints in the ridge regression and lasso. The residual sum of squares has elliptical contours, centered at the full least-squares estimates. The constraint region for ridge regression is

https://doi.org/10.26219/heal.aueb.6189

the disk $\beta_1^2 + \beta_2^2 \leq t^2$, while that for lasso is the diamond $|\beta_1| + |\beta_2| \leq t$. Both methods find the first point where the elliptical contours hit the constraint region. Unlike the disk, the diamond has corners; if the solution occurs at a corner, then it has one parameter $\beta_j = \theta_j$ equal to zero. When $p > 2$, the diamond becomes a rhomboid, and has many corners, flat edges and faces enhancing the sparsity of model leading potential coefficients exactly to zero.

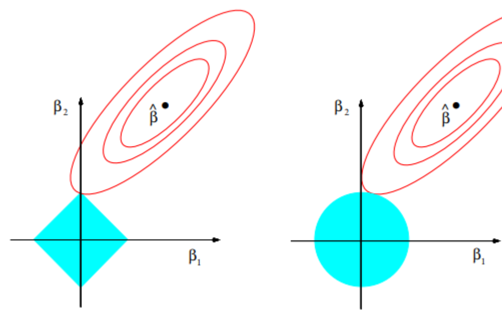

Figure 2.1: Estimation picture for the lasso (left) and ridge regression (right). The solid blue areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the residual-sum-of-squares function. The point $\hat{\beta}$ depicts the usual (unconstrained) least-squares estimate

In case we have normalized the predictors, so that $\mathbb{X}^T\mathbb{X} = \mathbb{I}_p$, the LASSO estimates can be written as

$$\hat{\theta}_{LASSO,j} = \hat{\theta}_{LS,j} \times max\left\{0, 1 - \frac{n\lambda}{|\hat{\theta}_{LS,j}|}\right\} \tag{2.13}$$

for j=1,2,...,p. Thus the LASSO coefficients are the ones derived from the least square method but shrinking to 0 and specifically for those predictors whose estimated coefficients' values are less than $n\lambda$ are set 0.

https://doi.org/10.26219/heal.aueb.6189

2.2.3 Elastic Net

Although LASSO has good properties and has shown success in many situations, it has some limitations. Namely:

- In case of $p > n$, before saturating LASSO selects at most n predictors due to the nature of the optimization problem. Once it has selected n variables, it typically keeps those coefficients non-zero and sets the others to zero, even if some of the excluded variables might also be relevant.

- If the correlation bonds between a group of predictors are very high, LASSO tends to select only one of them not caring at all which one is selected

- In usual cases where $n > p$, and if the pairwise correlations between the covariates are high, it has been empirically observed that Ridge Regression outperforms LASSO

Trevor Hastie and Hui Zou (2005) have proposed the Elastic Net penalization approach to overcome obstacles occurred with the two previous penalization methods, such the ones mentioned. It essentially comprises the Ridge and the LASSO penalty, and find the solution of $\hat{\theta}$ to the optimization problem:

$$\hat{\theta}_{ElNet} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^T X_i)^2 + \lambda\left[\frac{1}{2}(1-\alpha)||\theta||_2^2 + \alpha||\theta||_1\right], \quad (2.14)$$

where $\alpha \in [0, 1]$ is a tuning parameter. By construction, the penalty applied to an individual predictor is given by

$$\frac{1}{2}(1-\alpha)\theta_j^2 + \alpha|\theta_j|. \quad (2.15)$$

When $\alpha = 1$ it yields to the L_1 penalty, when $\alpha = 0$ it yields to the L_2 penalty while for any value between this range (and $\lambda > 0$) it yields to a strictly convex optimization problem and therefore a unique solution does exist. The predictors are once again assumed to be standardized and the response variable centralized.

https://doi.org/10.26219/heal.aueb.6189

Lq Penalties & Orthogonal Design

Frank and Friedman (1993), introduced the Bridge Regression which minimizes the RSS subject to the constraint of $\sum |\theta_j|^q \leq t$. That is we can assume the criterion

$$\underset{\theta \in R^p}{\text{minimize}} \left\{ \frac{1}{2N} \sum_{i=1}^{N} (y_i - \sum_{j=1}^{p} x_{ij}\theta_j)^2 + \lambda \sum_{j=1}^{p} |\theta_j|^q \right\}. \tag{2.16}$$

Essentially Ridge and LASSO are special cases of the Bridge Regression with $q = 2$ and $q = 1$ respectively. For $q = 0$, the term counts the number of non-zero elements in the vector θ and solving 2.16 leads to best-subset[2] selection. Best-subset selection leads to a nonconvex and combinatorial optimization problem (NP hard), and is usually not feasible to be solved (2^p models). Figure 2.1 displays the constraint regions corresponding to those penalties for the case of two predictors.

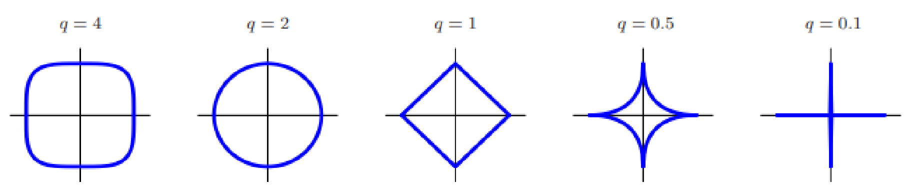

Figure 2.2: Constraint regions $\sum_{j=1}^{p} |\theta_j|^q \leq 1$ for different values of q. For $q < 1$, the constraint region is nonconvex

In the case of the orthogonal design case, i.e. assuming $\mathbb{X}^T\mathbb{X} = \mathbb{I}$, the penalized estimates are just a transformation of the least-squares estimates $\widetilde{\theta}$ (Hastie et al, 2015) according to Table 2.2 . As it can be observed, the Ridge regression does a proportional shrinkage, the LASSO translates each coefficient by a constant factor λ and truncates at zero, while the best-subset selection leaves the coefficient alone if it is bigger than $\sqrt{2\lambda}$, and otherwise sets it to zero.

[2] Best-subset selection is derived from the L_0 penalization where the penalty takes the form $\text{pen}(\theta) = \lambda \sum_j 1(\theta_j \neq 0)$

q	Estimator	Formula
0	Best subset	$\widetilde{\theta}_j \cdot \mathbb{I}\left[\lvert\widetilde{\theta}_j\rvert > \sqrt{2\lambda}\right]$
1	LASSO	$\text{sign}(\widetilde{\theta}_j)(\lvert\widetilde{\theta}_j\rvert - \lambda)_+$
2	Ridge	$\widetilde{\theta}_j/(1+\lambda)$

Table 2.2: Estimators of θ_j in the case of orthogonal model matrix $\mathbb{X}$

Generalized Linear Models

Penalization can also be applied to many methods other than linear regression. That is, a general form of the penalized generalized linear model will be introduced. Similarly with the minimization of the least squares in the linear regression, it is to fit the generalized linear model by minimizing a penalized maximum likelihood. The penalty term can be either L_1,L_2 or a combination of the two, yielding to elastic net. Thus, the optimization problem need to be solved can be expressed as

$$\min_{\theta_0,\theta} \frac{1}{N}\sum_{i=1}^{N} w_i l(y_i, \theta_0 + \theta^T X_i) + \lambda\left[(1-\alpha)||\theta||_2^2/2 + \alpha||\theta||_1\right], \tag{2.17}$$

where

$$l(y_i, \theta_0 + \theta^T X_i) = -log\left[\mathcal{L}(y_i, \theta_0 + \theta^T X_i)\right]$$

is the negative logarithm of the likelihood and hence maximizing likelihood is the same as minimizing the $l(y_i, \theta_0 + \theta^T X_i)$. The 2.17 can be adjusted according the generalized linear model need to be penalized. The w_i are weights and are typically used to handle unequal variances or heteroscedasticity in the data, while the tuning parameters λ and α control the overall strength of the penalty.

Hyperparameters Tuning

The hyperparameter λ in the penalization approaches (Friede et al. 2022)controls the complexity of the model; smaller values of λ free up more parameters and allow the model to adapt more closely to the training data. Conversely, larger values of λ restrict the parameters more, leading to sparser, more interpretable models that fit the data less closely. If $\lambda = 0$, the penalty loses its importance such that the resulting estimate $\hat{\theta}$ minimizes the underlying loss irrespective of the chosen penalty, while $\lambda > 0$ the estimate minimizes the loss subject to a constraint imposed by a penalty.

Forgetting about interpretability, we can ask for the value of λ that gives the most accurate model for predicting independent test data from the same population. Such accuracy is called the generalization ability of the model. A value of λ that is too large can prevent the model from capturing the main signal in the data, while too small a value can lead to overfitting. In this latter case, the model adapts to the noise as well as the signal that is present in the training data. In both cases, the prediction error on a test set will be inflated.

There is usually an intermediate value of λ that strikes a good balance between these two extremes, and in the process, produces a model with some coefficients equal to zero. A corresponding tuning procedure is achievable for the hyperparameter α in the elastic net. Usually, in order to estimate the most appropriate values, we can create artificial training and test sets by splitting up the given dataset at random, and estimating performance on the test data, using procedures such as cross-validation (CV).

There is also a significant relationship between the tuned values of the hyperparameters and the coefficients paths. The coefficients' paths in penalization techniques, particularly when tuned through cross-validation, resemble a finely threaded journey through model complexity. When using cross-validation to optimize hyperparameters, such as lambda or alpha, the regularization strength applied to the

https://doi.org/10.26219/heal.aueb.6189

coefficients varies. This journey can be visualized as a sequence of models, each associated with a different hyperparameter setting, encompassing the entire spectrum of model complexity.

2.2.4 Bayesian Penalization

The likelihood for the linear regression model is given by

$$y_i|\theta_0, x_i, \theta, \sigma^2 \sim Normal(\theta_0 + \sum_{j=1}^{p} x_{ij}\theta_j, \sigma^2) \tag{2.18}$$

where θ_0 represents the intercept, θ_j the regression coefficient for the predictor j, and σ^2 is the residual variance. In a Bayesian analysis (Van Erp et al. 2019), a prior distribution is specified for each parameter in the model, e.g., $p(\theta_0, \theta, \sigma^2, \lambda) = p(\theta_0)p(\theta|\sigma^2, \lambda)p(\sigma^2)p(\lambda)$. The prior for θ is conditioned on the residual variance σ^2, as well as on λ. The conditioning on σ^2 is necessary in certain cases to obtain a unimodal posterior.

In Bayesian penalized regression, λ is a parameter in the prior but has similar role as the penalty parameter in classical penalized regression. Since this penalty parameter λ is used to penalize the regression coefficient, it only appears in the prior for θ. We generally assume that the priors for the regression coefficients are independent. The prior distribution is then multiplied by the likelihood of the data to obtain the posterior distribution.

Shrinkage Priors Overview

In the foregoing discussion of frequentist-based penalization methods, we have explored the principles of controlling model complexity through the imposition of penalties on model parameters. This has led to an understanding of how regularization techniques, such as Ridge Regression, LASSO, and Elastic Net, operate within a frequentist framework. It is noteworthy that a parallel perspective ex-

https://doi.org/10.26219/heal.aueb.6189

ists within the Bayesian paradigm, where the control of model complexity can be achieved by assigning appropriate prior distributions to the hyperparameters governing the model. In this context, we recognize the fundamental correspondence between the frequentist-based penalization and the Bayesian approach, both serving as mechanisms to achieve parsimonious models while balancing the trade-off between model fit and complexity. An overview of shrinkage priors is following.

According to Hsiang (1975), the Ridge prior corresponds to Normal priors centered around zero on the regression coefficients, i.e.

$$\theta_j|\lambda, \sigma^2 \sim \text{Normal}\left(0, \frac{\sigma^2}{\lambda}\right), \tag{2.19}$$

$$\lambda \sim \text{half-Cauchy}(0, 1)$$

The posterior mean estimates will correspond to estimates obtained using the L_2 penalty.

We can extend the ridge prior by making the prior variances predictor-specific allowing for more variation. That is the Bayesian LASSO introduced by Park and Casella (2008), can be obtained as a scale mixture of normals with an exponential mixing density:

$$\theta_j|\tau_j^2, \sigma^2 \sim \text{Normal}(0, \sigma^2\tau_j^2)$$

$$\tau_j^2|\lambda^2 \sim \text{Exponential}\left(\frac{\lambda^2}{2}\right)$$

$$\lambda \sim \text{half-Cauchy}(0, 1)$$

Integrating τ_j^2 out results in double-exponential or Laplace priors on the regression coefficients, that is

$$\theta_j|\lambda, \sigma \sim \text{double-Exponential}\left(0, \frac{\sigma}{\lambda}\right) \tag{2.20}$$

https://doi.org/10.26219/heal.aueb.6189

Finally, the Elastic Net prior can be obtained as the following mixture of Normals according with Li and Lin (2010).

$$\theta_j|\lambda_2, \tau_j, \sigma^2 \sim \text{Normal}\left(0, (\frac{\lambda_2}{\sigma^2}\frac{\tau_j}{\tau_j - 1})^{-1}\right)$$

$$\tau_j|\lambda_2, \lambda_1, \sigma^2 \sim \text{Truncated-Gamma}\left(\frac{1}{2}, \frac{8\lambda_2\sigma^2}{\lambda_1^2}\right),$$

$$\lambda_1 \sim \text{half-Cauchy}(0, 1)$$

$$\lambda_2 \sim \text{half-Cauchy}(0, 1)$$

where the truncated Gamma density has support $(1, \infty)$. This implies the following conditional prior distributions for the regression coefficients:

$$p(\theta_j|\sigma^2, \lambda_1\lambda_2) = C(\lambda_1, \lambda_2, \sigma^2)exp\left\{-\frac{1}{2\sigma^2}(\lambda_1|\theta_j| + \lambda_2\theta_j^2)\right\}, \tag{2.21}$$

where $C(\lambda_1, \lambda_2, \sigma^2)$ denotes the normalizing constant. The corresponding posterior modes for θ_j are equivalent to the estimates from the classical elastic net penalty. Expression 2.21 illustrates how the elastic net priors offers a combination of the double-exponential prior, the lasso penalty $\lambda|\theta_j|$, and the normal prior, the ridge penalty $\lambda\theta_j^2$.

https://doi.org/10.26219/heal.aueb.6189

2.3 Early Stopping

Many statistical and machine learning algorithms construct a (possibly complicated) model by iteratively improving a simple model toward the maximum complex scenario permitted by the model specification. In such cases, one method of inducing regularization is to stop the fitting process before the most complex model is achieved, i.e. to identify the best trade-off between model simplicity (models close to the initial model) and fit to the data (models close to the final, most complex model) by stopping the fitting process early. In particular, the penalization approaches discussed in the previous subsection can also be included in this framework when considering the complete path of coefficients produced by varying the penalty parameter from infinity (simplest model determined by minimizing the penalty) to zero (full model without the penalty). Early stopping can be applied in the sense of ”stopping” at an optimal value for the penalty parameter via tuning.

2.3.1 Classification and Regression Trees

Overview

Classification and Regression Trees is a powerful non-parametric supervised learning algorithm employed in both classification and regression tasks, which construct the so-called decision trees. Its structure resembles that of a hierarchical tree, comprising key elements such as a root node, branches, internal nodes, and leaf nodes. This tree begins with a root node, devoid of incoming branches, from which branches emanate and lead to internal nodes, also termed decision nodes. These nodes, guided by available features, perform evaluations to create homogeneous subsets of data, ultimately represented by leaf nodes or terminal nodes. Each leaf node encapsulates a set of potential outcomes within the dataset. The resulting decision trees are then used to classify or predict new data.

https://doi.org/10.26219/heal.aueb.6189

CART

CART (Classification and Regression Trees) algorithm, was introduced by Breiman et al. (1984), in their fundamental paper "Classification and Regression Trees". CART constructs decision trees using a learning sample, which is a set of historical data with pre-assigned classes for all observations. In the context of clinical biostatistics, a learning sample would provide information about the patients' status. CARTs repeatedly partition the predictor space using binary splits of the covariate domain, simply asking YES/NO questions. Let m_{Parent} be a parent node and m_{Left} and m_{Right} the child nodes resulting by splitting the m_{Parent}. A mere illustration of the algorithm is illustrated in the next Figure.

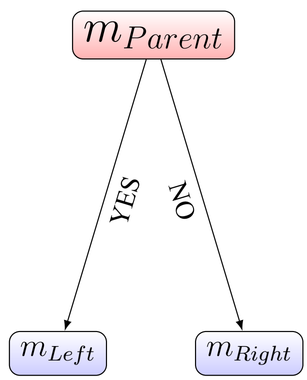

Figure 2.3: CART splitting algorithm

Those questions for instance could be: "Is patient's age greater than 70?" or "Has the patient CHD?" etc. CART is eligible to handle easily both numerical and categorical variables. In practical terms, one significant advantage of the CART algorithm is that the shape and structure of its decision trees remain consistent even when you apply monotone transformations to the input variables. This means that we can replace a variable with its logarithm or square root value, and the fundamental structure of the tree will remain the same.

Suppose now, that we have p explanatory variables $X_1, X_2, ..., X_p$ and n observations. Then each of the X_i can be :

https://doi.org/10.26219/heal.aueb.6189

- a numeric variable: $n-1$ possible splits
- an ordered factor with k levels: $k-1$ possible splits
- an unordered factor with k levels: $2^{k-1}-1$ possible splits

The best split is the one which results in the greatest decrease in impurity. Depending the task, various measures of impurity will be introduced. An overview methodology of CART can be expressed as:

1. Construction of maximum tree by binary splitting based on an impurity measure
2. Choice of the right tree size, determined by tree pruning
3. Classify or predict new observations

Details about each of those steps will be discussed in the remainder of this subsection.

Regression Trees

As Breiman et al. (1984) introduced, assume our data consists of P explanatory variables denoted as X, and a response variable Y with N pairs of observations, $(x_i, y_i), \quad for \quad i = 1, 2, ..., N$ with $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$. The algorithm will determine the splitting features and the split points. Let J partitioned regions, $R_1, R_2, ..., R_J$, and we model the response as a constant c_j in each region

$$f(x) = \sum_{j=1}^{J} c_j I(x \in R_j). \tag{2.22}$$

Adopting as loss function the sum of squares $\sum(y_i - f(x_i))^2$, it is easy to observe that the best candidate $\hat{c}_j$ is the average of the y_i in region R_j:

$$\hat{c}_j = ave(y_i | x_i \in R_j) \tag{2.23}$$

 https://doi.org/10.26219/heal.aueb.6189

The optimization problem we need to solve in terms of minimizing the sum of squares is computationally infeasible, thus we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable k and split point s and define the sub-regions

$$R_1(k,s) = \{X|X_k \leq s\} \quad \text{and} \quad R_2(k,s) = \{X|X_k > s\} \tag{2.24}$$

Then we seek the splitting variable k and split point s that solve

$$\min_{k,s}\left[\min_{c_1}\sum_{x_i\in R_1(k,s)}(y_i - c_1)^2 + \min_{c_2}\sum_{x_i\in R_2(k,s)}(y_i - c_2)^2\right]. \tag{2.25}$$

For any pair (k,s) the inner minimization is solved by

$$\hat{c_1} = ave(y_i|x_i \in R_1(k,s)) \quad \text{and} \quad \hat{c_2} = ave(y_i|x_i \in R_2(k,s)) \tag{2.26}$$

The determination of the split point s for each splitting variable may be done extremely fast, and hence the optimal pair (k,s) can be determined by scanning through all of the inputs. After determining the optimal split, we partition the data into the two resulting regions and repeat the splitting process on each region. The method is then repeated for all of the generated regions.

Classification Trees

In case we have a K-class classification task (Breiman et al. 1984), the algorithm need to undergo some adjustments concerning splitting criterion. Considering a node m, representing a region R_m with N_m observations let

$$\hat{p}_{mk} = \frac{1}{N_m}\sum_{x_i\in R_m} I(y_i = k), \tag{2.27}$$

the proportion of observations which belong in class k, in a node m. We classify the observations in node m to class $k(m) = \underset{k}{\text{argmax}}\hat{p}_{mk}$, the majority class in node m. Denoting as $Q_m(T)$ a type of measure of node impurity, can be discriminated as:

- Missclassification Error:

$$\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)} \tag{2.28}$$

- Gini Index:

$$\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{2.29}$$

- Cross Entropy or Deviance:

$$-\sum_{k=1}^{K} \hat{p}_{mk} log_2(\hat{p}_{mk}). \tag{2.30}$$

In case we have a binary classification task and denoting as p the proportion in the second class, the above measures are simplified as

- Missclassification Error: $1 - \text{max}(p, 1 - p)$
- Gini Index: $2p(1 - p)$
- Cross Entropy or Deviance: $-plog(p) - (1 - p)log(1 - p)$

All of those measures are similar, but cross entropy and Gini index are differentiable, and hence more liable to numerical optimization. It is imperative to thoroughly understand the capabilities of each of these measures and assign them specific roles, whether it be to guide the expansion of the tree or to facilitate the pruning of the maximal tree structure. To do so, let us assume a scenario where we have a binary classification task where each class consists of 400 patients.

https://doi.org/10.26219/heal.aueb.6189

Denoting as ● the patients who have done the COVID-19 vaccination and as ⋆ those who have not, assume the following splits.

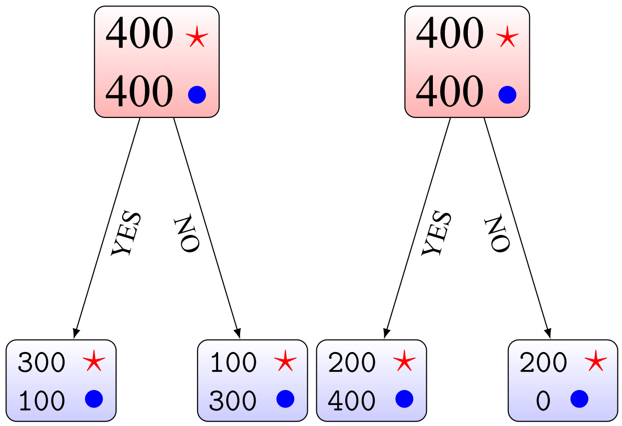

Figure 2.4: Impurity measures evaluation

Then the impurity measures for each of the two splits are:

- First split: MCE=0.24, Gini Index=0.37, Entropy=0.81
- Second split: MCE=0.24, Gini Index=0.33, Entropy=0.68

The missclassification error for both splits are 0.24, while the Gini index and the Entropy are lower in the second split indicating that the second split has produced purer child nodes, thus it is preferable. Since Gini Index and cross-entropy are more vulnerable to changes in the node probabilities are usually selected in order to grow a tree. On the other hand, the missclassification rate is used for the tree pruning.

Assuming the use of cross-entropy as the criterion for growing the tree, it becomes necessary to find a measure that can effectively quantify the changes in each node during the splitting process. This is where the concept of Information Gain (IG) plays a crucial role. Information Gain serves as a measure to quantify the change in node impurity that occurs with each split.

The computation of Information Gain is defined as follows:

$$IG(X,Y) = E(Y) - E(Y|X) \tag{2.31}$$

where $E(Y)$ is the entropy of the node with respect to the target variable Y before the split and $E(Y|X)$ is the new entropy in the node resulting from accounting for the predictor X after the split. In essence, Information Gain quantifies the reduction in uncertainty (or impurity) achieved in a node when considering a particular predictor for splitting. The higher the IG value, the greater the reduction in impurity, indicating that the chosen predictor is effective at separating the data into more homogeneous subsets.

During the tree-building process, the algorithm examines all possible binary splits, calculating the IG for each potential split. It then selects the predictor that results in the highest Information Gain as the splitting variable for that node. This iterative process continues until a stopping criterion, such as a maximum tree depth or minimum node size, is met. To illustrate this process, let us consider an example that clearly demonstrates how the Entropy and Information Gain are computed and how IG helps determine the best splitting variable for a node.

Assuming a clinical study focused on cardiovascular disease risk, a team of biostatisticians aims to develop a model to predict the presence or not of a cardiovascular disease (CVD) in a group of patients. The total number of patients are 30, which 16 have a CVD and 14 do not have CVD. For this classification task, the scientists have only 2 available predictors:

- Age (Continuous Variable): The age of patients in years
- Smoking Status (Categorical with 3 levels): 1. Non-smoker 2.Former smoker 3.Current smoker

The algorithm needs to decide between those two possible features to split and to do so, the information gain will be utilized.

https://doi.org/10.26219/heal.aueb.6189

First Instance

Figure 2.5 displays the tree growth according to the predictor `Age`. That is, splitting the parent node leads to two child nodes, hence

$$E(CVD) = -\frac{16}{30}log_2(\frac{16}{30}) - \frac{14}{30}log_2(\frac{14}{30}) = 0.99$$

$$E(Age \leq 50) = -\frac{12}{13}log_2(\frac{12}{13}) - \frac{1}{13}log_2(\frac{1}{13}) = 0.39$$

$$E(Age > 50K) = -\frac{4}{17}log_2(\frac{4}{17}) - \frac{13}{17}log_2(\frac{13}{17}) = 0.79$$

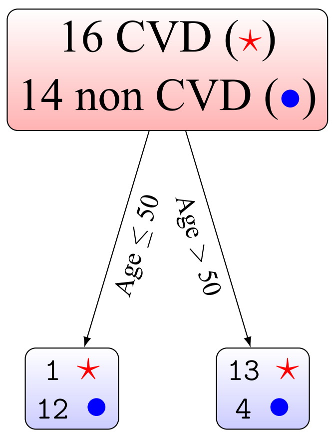

Figure 2.5: CART splitting based on Age

The weighted average of entropy for each child node is:

$$E(Age) = \frac{13}{30} \cdot 0.39 + \frac{17}{30} \cdot 0.79 = 0.62$$

Therefore the Information Gain can be computed as:

$$IG(CVD, Age) = E(CVD) - E(CVD|Age) = 0.99 - 0.62 = \mathbf{0.37}$$

https://doi.org/10.26219/heal.aueb.6189

Second Instance

Figure 2.6 displays the tree growth according to the categories of the predictor `Smoking Status`. That is, splitting the parent node leads to three child nodes, hence

$$E(\textit{Non-smoker}) = -\frac{7}{8} log_2\left(\frac{7}{8}\right) - \frac{1}{8} log_2\left(\frac{1}{8}\right) = 0.54$$

$$E(\textit{Former smoker}) = -\frac{4}{10} log_2\left(\frac{4}{10}\right) - \frac{6}{10} log_2\left(\frac{6}{10}\right) = 0.97$$

$$E(\textit{Current smoker}) = -\frac{5}{12} log_2\left(\frac{5}{12}\right) - \frac{7}{12} log_2\left(\frac{7}{12}\right) = 0.98$$

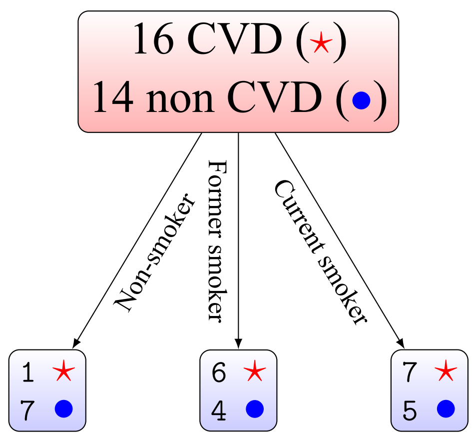

Figure 2.6: CART splitting based on Smoking Status

The weighted average of entropy for each child node is:

$$E(\textit{Smoking Status}) = \frac{8}{30} \cdot 0.54 + \frac{10}{30} \cdot 0.97 + \frac{12}{30} \cdot 0.98 = 0.86$$

Therefore the information gain can be computed as:

$$IG(\textit{CVD,Smoking Status}) = E(\textit{CVD}) - E(\textit{CVD}|\textit{Smoking Status}) = 0.99 - 0.86 = \mathbf{0.13}$$

https://doi.org/10.26219/heal.aueb.6189

We can observe that the information gain derived from feature `Age` is almost three times higher than those derived from the feature `Smoking Status`. Consequently it can be assumed that the predictor Age provides more information about the target variable and therefore the Age will be utilized as the first splitting criterion in the decision stump. The algorithm iteratively applies this approach in order to reduce the node impurity.

Tree Pruning

The early stopping regularization approach in CARTs lies on finding the optimal number of splits of the root node. The procedure in which we try to find the optimal number of splits it is called tree pruning. According to Breiman et al (1984) and Hastie et al. (2001), pruning a tree aims to result in a smaller tree with respect to the number of splits which potentially has lower variance, better interpretability and slightly more bias. The process of pruning a tree is relatively simple.

Considering a regression tree, the strategy is to grow a large tree (T_0) and stopping the binary splitting when some minimum node size is achieved. The pruning of the initial tree (T_0) is achieved via *cost-complexity* function. In particularly, denoting as T a subtree $(T \in T_0)$ which is the result obtained by eliminating any number of its internal nodes, we define terminal nodes by m leading to a region R_m. Let also $|T|$ reflecting the number of terminal nodes in T and:

$$
\begin{aligned}
N_m &= \#\{x_i \in R_m\} \\
\hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i \quad \text{and} \\
\mathcal{Q}_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \qquad (2.32)
\end{aligned}
$$

where $Q_m(T)$ representing a measure of node impurity as we have already discussed in classification trees, which in our case is the squared-error node of impurity we define the cost complexity criterion as:

$$C_a(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{2.33}$$

The motivation is to find, for each value of α, the subtree $T_\alpha \in T_0$ to minimize $C_a(T)$. The tuning parameter $\alpha > 0$, which is also called cost complexity parameter governs the trade-off between tree size and its goodness of fit to the data. Large values of α yield in smaller trees T_α, whereas for smaller values of α yield in larger trees T_α. Moreover the tuning parameter can also take the value zero which results in the initial unpruned tree T_0.

To identify and construct this optimal subtree T_α, a method known as *weakest link pruning* is employed. This approach involves a step-by-step process of collapsing internal nodes within the tree which produces the smallest per-node increase in $\sum_m N_m Q_m(T)$. The goal of weakest link pruning is to iteratively identify and eliminate those internal nodes that contribute the least to the overall cost while maintaining the tree's predictive accuracy. Estimation of α can be determined through cross-validation. Finally as far as classification trees is concerned, the measure of node impurity $\mathcal{Q}_m(T)$ is usually the missclassification error.

Missing Values & Surrogate Variables

According to Breiman et al. (1984) and Hastie et al. (2001), surrogates are alternate splitting variables that can be used in case the primary predictor for a split contains missing data. Surrogate variables are ascertained through an evaluation based on their association or correlation with the primary predictor. The objective is to identify substitutes that can reasonably approximate the role of the primary predictor when data is missing. These surrogate variables are subsequently em-

https://doi.org/10.26219/heal.aueb.6189

ployed to make split decisions for observations lacking information about the primary predictor.

Following the selection of the primary predictor, a hierarchy of surrogate predictors and their corresponding split points is established. The most effective surrogate, which provides outcomes most akin to those derived from the primary predictor, is selected as the initial substitute. The same process is then applied to designate the second surrogate, the predictor that follows in performance, and so forth.

In essence, the goal is to determine the optimal surrogate and, potentially, the associated split point that most closely replicates the results of the primary predictor when missing. On one hand, we take advantage of the surrogate variables by reducing data loss since by leveraging surrogate variables the tree utilize more features for training and prediction leading to potentially better performance and by improving generalizability since using surrogate variables can help the model avoid overfitting to specific patterns in the training data with complete information. On the other hand some limitations must be considered, since the effectiveness of the surrogates depends on how strong they correlate with the missing feature. Thus, in case of weak correlation might not provide reliable information.

https://doi.org/10.26219/heal.aueb.6189

2.3.2 Boosting

Background & Overview

The boosting methodology emerged in the field of machine learning, introduced by Robert Shapire and Yoram Singer (1996). Initially, its focus was on solving classification tasks, with the most notable algorithm being AdaBoostM1. Subsequently, numerous other boosting approaches were developed. Among these, the Gradient Boosting algorithm, introduced by Jerome Friedman (1999), played a pivotal role by expanding the methodology to regression tasks. Furthermore, in 1999, Leo Breiman proposed a hybrid technique called Adaptive Bagging, which combined aspects of bagging and boosting. The evolution and growing prominence of boosting approaches in modern machine learning techniques are undoubtedly intriguing and have significantly impacted the field.

The boosting algorithm is an ensemble method, inspired by the concept that an ensemble of weak learners can produce predictive results that are just as accurate as a single strong learner. Here, a weak learner typically refers to a model, especially in classification tasks, that performs slightly better than random guessing (e.g., with accuracy slightly above 50%). In contrast, a strong learner is capable of achieving high accuracy, nearly 95% or more. The fundamental idea behind boosting is to iteratively enhance the performance of the weak learner by learning from its previous applications. It does so by modifying the training data, where observations are re-weighted in each round based on the weak learner's performance. The algorithm's primary goal is to focus on the challenging elements of the data by repeatedly applying the weak learner to observations that carry higher weights, as those previously misclassified observations receive increased importance. This process ultimately leads to improved predictive performance.

https://doi.org/10.26219/heal.aueb.6189

AdaBoost

As previously mentioned, one of the most renowned boosting algorithms is AdaBoost. According to Shapire and Singer (1996) and Hastie et al. (2001), AdaBoost is fundamentally an adaptive algorithm, characterized by its ability to automatically adjust its parameters according to the data based on its current performance in each iteration. This adaptability allows AdaBoost to effectively focus on observations that are challenging to classify, iteratively improving its overall performance. To delve in the algorithm, consider a two-class classification task given a vector or predicted variable X and the response variable Y coded as $Y \in \{-1, 1\}$. A classifier $G(X)$ produces a prediction taking one of the two possible values. The error rate on the training data is

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} I\left(y_i \neq G(x_i)\right), \tag{2.34}$$

and the expected error rate in test data is $E_{XY} I(Y \neq G(X))$. The application of the weak classification algorithm sequentially will produce a sequence of weak classifiers $G_m(x), \quad m = 1, 2, ..., M$. The predictions from all of them, are combined through a weighted majority vote to produce the final prediction

$$G(x) = sign(\sum_{m=1}^{M}) \alpha_m G_m(x), \tag{2.35}$$

where $\alpha_1, \alpha_2, ..., \alpha_M$ are computed by the boosting algorithm and reflect the contribution of each respective $G_m(x)$. The purpose is to give higher influence to the more accurate classifiers. The data modifications at each iteration of the AdaBoost.M1 algorithm consist of applying weights $w_1, w_2, ..., w_N$ to each of the training observations (x_i, y_i). Initially all of the observations are set to $w_i = 1/N$ and for each successive iteration $m = 2, 3, ..., M$ the observation weights are individually modified and the classification algorithm is reapplied to the weighted observations. At step m, the weights of observations that were misclassified by the classifier $G_{m-1}(x)$ produced in the previous step are increased, while the weights of observations that were successfully classified are decreased. As iterations progress, observations that are difficult to appropriately categorise gain greater influence. A detailed overview of the algorithm is presented below.

Algorithm 1 AdaBoost.M1 Algorithm

1: Initialize the observation weights $w_i = 1/N, \quad i = 1, 2, ..., N$
2: For $m = 2, 3, ..., M$
    (a) Fit a classifier $G_m(x)$ to the training data using weights w_i
    (b) Compute
$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$
    (c) Compute $\alpha_m = log((1 - err_m)/err_m)$
    (d) Set $w_i \longleftarrow w_i \cdot exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], \quad i = 1, 2, ..., N$
3: Output $G(x) = sign\left[(\sum_{m=1}^{M})\alpha_m G_m(x)\right]$

The current classifier $G_m(x)$ is induced on the weighted observations at line 2a. The resulting weighted error rate is computed at line 2b. Line 2c calculates the weight α_m given to $G_m(x)$ in producing the final classifier $G(x)$. The individual weights of each of the observations are updated for the next iteration at line 2d. Observations misclassified by $G_m(x)$ have their weights scaled by a factor $exp(\alpha_m)$, increasing their relative influence for inducing the next classifier $G_{m+1}(x)$ in the sequence.

Forward Additive Stagewise Modeling

The ensemble generated by the boosting algorithm can be fundamentally perceived as a method for constructing an additive expansion within a framework of basis functions. In the context of AdaBoost, these basis functions correspond to the individual classifiers, denoted as G(x). In general, basis function expansions (Hastie et al., 2001) take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m), \tag{2.36}$$

where β_m are the expansion coefficients and $b(x;\gamma) \in \mathbb{R}$ are usually simple functions of the multivariate vector x, characterized by a set of parameters γ. These models are usually fitted by minimizing a loss function that is averaged over the training data, such as the squared-error or a likelihood-based loss function,

$$\min_{\{\beta_m . \gamma_m\}_1^M} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x; \gamma_m))\right). \tag{2.37}$$

Since 2.37 demands computationally intensive numerical optimization techniques, forward stagewise modelling approximates it by adding additional basis functions to the expansion in a sequential manner without altering the parameters and coefficients of those that have previously been added. The overview of the algorithm is described below.

As described in the algorithm, the optimal basis function $b(x;\gamma_m)$ and corresponding coefficient β_m is added to the current expansion $f_{m-1}(x)$. Assuming as loss function the squared-error loss, the optimization problem is expressed as

$$L(y_i, f_{m-1}(x_i) + \beta b(x_i \gamma)) = (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2, \tag{2.38}$$

Algorithm 2 Forward Stagewise Additive Modelling

1: Initialize $f_0(x) = 0$
2: For $m = 1 \quad to \quad M$:
(a) Compute

$$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\text{argmin}} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)\right)$$

(b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

where $y_i - f_{m-1}(x_i)$ is simply the residual of the current model on the ith observation, thus the term $\beta_m b(x; \gamma_m)$ that best fits the current residuals is added to the expansion at each step. Similarly concerning a classification task, the AdaBoost algorithm can be expressed as a forward stagewise additive modelling using the exponential loss function $L(y, f(x)) = exp(-yf(x))$.

Gradient Boosted Decision Trees

Friedman (1999) introduced the Gradient Boosting algorithm, often referred to as ”Statistical Boosting”, which bridged the gap for regression tasks. This variant of the algorithm creates additive regression models by iteratively applying a weak learner to the current pseudo-residuals using least squares in each step. These pseudo-residuals represent the gradient of the loss function with respect to the model values for each data point in the training set, evaluated at the current iteration. The gradient boosting approach can be applied in case we consider as weak learners, regression and classification trees (CART).

In CART algorithm, a constant c_j is assigned to each partitioned region R_j, thus the predictive rule is

$$x \in R_j \longrightarrow f(x) = c_j$$

https://doi.org/10.26219/heal.aueb.6189

Denoting a single tree as

$$T(x;\Theta) = \sum_{j=1}^{J} c_j I(x \in R_j), \tag{2.39}$$

with parameters $\Theta = \{R_j, c_j\}_1^J$, which can be found by minimizing the empirical risk

$$\hat{\Theta} = \underset{\Theta}{\text{argmin}} \sum_{j=1}^{J} \sum_{x_i \in R_J} L(y_i, c_j). \tag{2.40}$$

The above optimization problem is a combinatorial one, and hence we usually seek for suboptimal solutions. One of these is by implementing a top-down recursive partitioning algorithm to find the R_m, such as the one described in CART algorithm. That is, a boosted tree model can be expressed as a sum of such trees, i.e.

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m), \tag{2.41}$$

induced in a forward stagewise manner as the one described in Algorithm 2. At each iteration, the optimization problem need to be solved in the forward stepwise procedure is

$$\hat{\Theta_m} = \underset{\Theta_m}{\text{argmin}} \sum_{i=1}^{N} L\left(y_i, f_{m-1}(x_i) + T(x;\Theta_m)\right), \tag{2.42}$$

for the region set and constants $\Theta_m = \{R_{jm}, c_{jm}\}_1^{J_m}$ of the next tree, given the current model $f_{m-1}(x)$. Given the regions R_{jm}, we can easily find the optimal values for the constants c_{jm} as

$$\hat{c}_{jm} = \underset{c_{jm}}{\text{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + c_{jm}). \tag{2.43}$$

https://doi.org/10.26219/heal.aueb.6189

To optimize Equation 2.42, we employ the gradient boosting approach. Specifically, this optimization problem can be tackled using any appropriate differentiable loss function of our choice. That is, at mth iteration we induce a tree $T(x;\Theta_m)$ whose predictions are as close as possible to the negative gradient of the loss function evaluated at the previous *(m-1)*th iteration. Hence the tree induced can be expressed as

$$\tilde{\Theta}_m = \underset{\Theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(r_{im} - T(x_i;\Theta)\right)^2 , \tag{2.44}$$

where

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}(x_i)} . \tag{2.45}$$

An overview of this variant of gradient boosting is given below.

Algorithm 3 Gradient Boosted Decision Trees Algorithm

Initialize $f_0(x) = \underset{c}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, c)$

For *m = 1 to M*:

(a) For $i = 1, 2, ..., N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{m-1}}$$

(b) Fit a tree to the targets r_{im} giving terminal regions $R_{jm}, \quad j = 1, 2..., J_m$

(c) For $j = 1, 2, ..., J_m$ compute

$$c_{jm} = \underset{c}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + c)$$

(d) Update $f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J_m} c_{jm} I(x \in R_{jm})$

Output $\hat{f}(x) = f_m(x)$

https://doi.org/10.26219/heal.aueb.6189

The algorithm for both regression and classification tasks is identical. The vital difference of course lies on the choice of the loss function used to minimize the optimization problem and consequently the pseudo-residuals r_{im} derived. In a regression task, the most common loss function is

$$L(y_i, f(x)) = \frac{1}{2}\left(y_i - f(x)\right)^2, \tag{2.46}$$

whose pseudo-residuals are the same with the ones derived from the linear regression. On the other hand concerning a classification task, it is reasonable to consider the log-likehood as a candidate loss function. In particularly, the log-likehood concerning a binary outcome is $\sum y_i \cdot log(p) + (1 - y_i) \cdot log(1 - p)$. Since the better the prediction the higher the value of the log-likehood, in order to utilize it as a loss function we will multiply it by (-1). Applying some modifications in the formula in order to transform it in a (differentiable) function of the predicted *log(odds)* instead of the predicted probability p, it leads to the loss function

$$L(y_i, f(x)) = -y_i \cdot log(odds) + log\left(1 + e^{log(odds))}\right). \tag{2.47}$$

In an effort to enhance clarity and facilitate a comprehensive understanding of the underlying distinctions between regression and classification tasks, we aim to succinctly delineate the key variances in the algorithm pertaining to these two objectives.

Function	Regression	Classification
$L(y, f(x))$	$\frac{1}{2}\left(y_i - f(x)\right)^2$	$-y_i log(odds) + log\left(1 + e^{log(odds)}\right)$
r_{im}	$y_i - f(x)$	$y_i - \frac{e^{log(odds)}}{1+e^{log(odds)}}$
c_{jm}	$\overline{r_m}$	$\frac{\sum Residual_i}{\sum\left(p_{(i,m-1)} * (1-p_{(i,m-1)})\right)}$

Table 2.3: Key Differences Between Regression and Classification Tasks in the Gradient Boosted Trees Algorithm

https://doi.org/10.26219/heal.aueb.6189

The optimal output value c_{jm} for each leaf in regression is straightforward, since minimizing the corresponding loss function yields to the mean of the residuals in each region J_m. In classification although it is more complex and for this reason, we can approximate the loss function with a second order Taylor polynomial which yield to the result indicated in the Table 2.3.

The parameter ν in Algorithm 3 (line 2d) serves as a crucial factor influencing the learning rate of the boosting procedure. A lower ν value, signifying more shrinkage, results in a larger training risk for a given number of iterations M. These parameters are not independent; smaller ν values lead to larger M values, indicating a trade-off between them. Moreover, a small value for the learning rate reduces the effect each individual tree has on the final prediction, thus improving accuracy in the long term. Overall, these parameters play a pivotal role in the early stopping regularization process.

2.3.3 Interpretation

Machine learning algorithms are often characterized as black box models due to their inherent complexity, which can make it challenging to gain insight into how they arrive at their decisions. Notably, ensemble methods such as random forests and boosting algorithms contribute to this complexity, as the ensemble's inner workings are not easily interpretable. In contrast, a single decision tree is considered to be more interpretable due to its inherent simplicity.

In the realm of parametric modeling, the importance of model interpretation cannot be overstated. When using the final model for making predictions or classifications on new data, understanding the role and influence of each predictor variable becomes paramount. This is particularly crucial in datasets with a large number of predictors, where identifying the most influential variables can significantly impact model understanding and decision-making. To address the need for model interpretability, variable importance scores, as introduced by Breiman, serve as a

https://doi.org/10.26219/heal.aueb.6189

valuable tool.

Implementing the CART algorithm and concerning the final decision tree derived T, Breiman et al. (1984) proposed as a measure of relevance, for the predictor X_l, the quantity

$$\mathcal{I}_l^2(T) = \sum_{m=1}^{M-1} \hat{i}_m^2 I(X(m) = l). \tag{2.48}$$

Typically, each of the predictors $X(m)$ is eligible to binary partition the region associated with the node m at every step of the algorithm. Denoting as $\hat{i}_m^2$ a measure of splitting improvement (such as the difference in Gini Index), the variable which maximizes it will be selected as the splitting variable. Thus, the variable's squared relative relevance is the sum of such squared improvements over all internal nodes for which variable X_l was picked as the splitting variable. The square root of the measure $\mathcal{I}_l^2(T)$ is the actual relevance.

While numerous methodologies exist for calculating and interpreting variable relevance based on this measure, it's important to note that different authors and R packages may vary slightly in their approaches to quantifying these relevances. In addition, these relevance measures are often referred to as importance scores, as they assign each variable a score indicating its relevance or contribution to the model. This concept leads to the creation of importance plots, which provide a visual representation of Breiman's methodology for assessing variable importance.

This importance measure is easily adapted to the additive tree expansions where it merely average it across the trees

$$\mathcal{I}_l^2 = \frac{1}{M} \sum_{m=1}^{M} I_l^2(T_m). \tag{2.49}$$

Distinct discrimination for the measure of improvement selected between regression and classification trees is sensible. In the former ones usually the measure

https://doi.org/10.26219/heal.aueb.6189

selected is the MSE, while for the latter ones can be the Gini impurity or the MCE. In case we have multinomial classification, there are certain methods employed which result into easier implementation of the algorithm.

Assuming we have K-class classification, a strategy called one-vs.-rest can be applied. Within this context, we actually transform the multinomial classification problem to multiple binary classification problems. To do so, we train a single classifier per class, with the samples of that class as positive samples and all other samples as negatives. Assuming as classifiers decision trees we have K different models, thus 2.49 generalizes to

$$\mathcal{I}_{lk}^2 = \frac{1}{M}\sum_{m=1}^{M}\mathcal{I}_l^2(T_{km}), \tag{2.50}$$

where I_{lk} is the relevance of X_l in discriminating the class k observations from the other classes. The overall relevance of X_l is obtained by averaging over all classes.

https://doi.org/10.26219/heal.aueb.6189

2.4 Ensembling

Ensembling regularization techniques represent a powerful paradigm in the realm of machine learning, offering a potent strategy to enhance model performance and bolster predictive accuracy. Unlike traditional regularization methods that directly modify a single model, ensembling regularization leverages the diversity of multiple models to mitigate issues of overfitting and instability. Imagine a scenario where a single model excels at capturing intricate details within the training data but falters when tasked with making predictions on entirely new and unseen data. In such cases, ensembling regularization steps in as a remedy by combining an array of model variants, each with its unique strengths and idiosyncrasies. This ensemble approach not only diminishes the model's inherent variability but also facilitates the creation of more robust and generalizable predictions. In the remainder of the subsection, we delve into one of the most known ensembling techniques, the random forest algorithm.

2.4.1 Random Forests

Background & Overview

In spite of the notable advantages of the Classification and Regression Trees (CART) algorithm, it is not immune to inherent limitations. One of the most substantial challenges lies in the potential for high variance in the tree structures even after the pruning process. To address this issue, the concept of ensemble learning was explored, resulting in the development of the Bagging (Bootstrap Aggregation) and Random Forest algorithms. Bagging, introduced by Breiman (1994), is a meta-algorithm designed to enhance the accuracy of machine learning models for classification and regression tasks. It accomplishes this by mitigating the variance in the final predictions. The Bagging approach involves creating multiple subsets of the training dataset through random sampling with replacement, training

https://doi.org/10.26219/heal.aueb.6189

separate models (often trees) on each subset, and subsequently aggregating their predictions.

Subsequently, in 2001, Breiman introduced an improved variant called the 'Random Forest.' This method involves an additional randomization step in which it builds an ensemble of decision trees with the goal of reducing correlation among them, leading to lower variance. Empirical evidence presented by Breiman indicated that the Random Forest technique yields results akin to those achieved by the Boosting algorithm. In general, we can assume the Random Forest algorithm, as in improved version of the CART algorithm. A simple overview of the methodology, is illustrated in Figure 2.7.

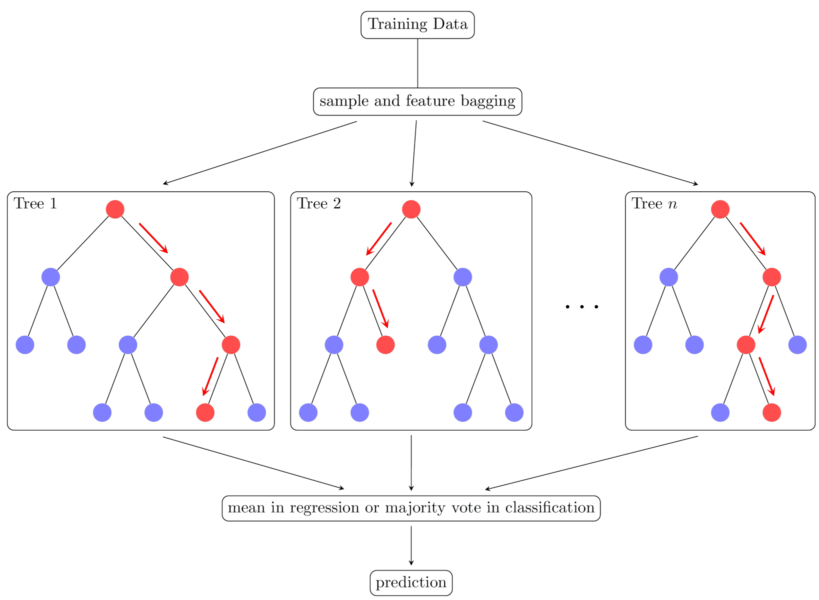

Figure 2.7: Random Forest methodology

Random Forest algorithm

The constraints and limitations of the Classification and Regression Trees (CART) algorithm paved the way for the emergence of ensemble learning and model averaging techniques, both of which exhibit some common characteristics. In the context of the Random Forest ensemble algorithm, it introduces two distinct randomization elements into the process of constructing models based on decision trees.

- Random sampling with replacement: A bootstrap sample comprises a randomly drawn subset from the training dataset, with equal size n, obtained through sampling with replacement. By performing this procedure repeatedly for a given number B, a sequence of B decision tree models is generated.

- Random feature selection: The Random Forest algorithm employs a strategy of selecting a random subset of attributes m for consideration during each split, effectively excluding certain features from the training data obtained through bootstrapping.

These two randomization steps, achieving a de-correlation (more to come) of the single trees resulting to lower variance. The size of the random subset of predictors at each node, can be assumed as a tuning parameter. Typical value of m is the square root of the total number of predictors in the data, $m = \sqrt{p}$. After B such trees $\{T(x; \Theta_b)\}_1^B$ are grown, the random forest predictor is

$$\hat{f}_{rf}^B(x) = \frac{1}{B} \sum_{b=1}^{B} T(x; \Theta_b). \tag{2.51}$$

The notation is similar as in the boosting subsection, where Θ_b characterizes the bth random forest tree in terms of split variables and cutpoints at each node. Moreover, the inventors made the following recommendations:

 https://doi.org/10.26219/heal.aueb.6189

- For classification, the default value for m is $\sqrt{p}$ and the minimum node size is 1

- For regression, the default value for m is $p/3$ and the minimum node size is 5.

An overview of the random forest algorithm is presented below.

Algorithm 4 Random Forest Algorithm for Regression or Classification

1: For $b = 1$ to B:
(a) Draw a bootstrap sample Z^* of size n from the training data
(b) Grow a random forest tree T_b to the boostrapped data, by recursively repeating the following steps for each terminal node of the tree, until minimum node size n_{min} is achieved
    i. Select m variables at random from the p variables
    ii. Pick the best variable/split point among the m
    iii. Split the node into two daughter nodes
2: Output the ensemble of trees $\{T_b\}_1^B$
To make a prediction at a new observation x:
Regression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$
Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random forest tree.
Then $\hat{C}_{rf}^B(x) = \text{majority vote} \quad \{\hat{C}_b(x)\}_1^B$

De-correlation & Variance Reduction

We mentioned earlier, that the introduction of an additional randomization step plays a pivotal role in generating de-correlated trees. This de-correlation is instrumental in reducing variance. To gain a deeper understanding of this process, it is valuable to explore how de-correlation is achieved. Ensemble algorithms, such as bagging and random forest, are motivated by the notion of harnessing the distinct idiosyncrasies of individual models, typically decision trees, and then averaging their outcomes to achieve a reduction in variance without incurring a significant increase in bias compared to a single, noise-sensitive model. In order to meet those expectations, the trees should be not correlated. However, in the case of bagging's bootstrap samples, when one or more predictors exert a pronounced influence, they tend to be consistently selected in a substantial proportion of the B constructed trees. This pronounced selection of influential predictors contributes to the correlation among the trees, potentially undermining the variance decrease goal.

In contrast, the random forest algorithm introduces a pivotal randomization step by considering only a subset of predictors during each split. This practice ensures that different subsets of predictors are utilized in the construction of individual trees. Consequently, this method of randomization leads to a natural de-correlation of the trees within the ensemble. From a mathematical perspective, equation 2.52 reflects the variance of the average of B identically distributed (i.d) random variables, each with variance σ^2 and pairwise correlation ρ.

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2. \tag{2.52}$$

As B increases, the second term disappears but the first remains. The vital discrimination between bagging and random forest is that, while in bagging the correlated bagged trees limit the benefits of averaging, the random forest methodology achieve this venture by reducing the correlation ρ between the trees.

2.4.2 Interpretation

As previously elucidated, the random forest algorithm is inherently characterized as a black box model, primarily due to its composition of multiple unpruned decision trees. This ensemble approach makes the interpretation of individual trees and the entire model a challenging task. To address this inherent opacity, variable importance scores, which were introduced in the preceding section, provide a mechanism for assessing the relevance and contribution of predictor variables. However, our specific interest lies in the importance scores derived from the Out-Of-Bag (OOB) samples, a widely recognized technique pioneered by Breiman (2001).

OOB samples in random forests are data points from the original dataset that are not included in the bootstrap sample used to train a particular decision tree in the forest. These OOB samples serve as a natural and efficient cross-validation mechanism within the random forest algorithm. Each decision tree is trained on a slightly different subset of the data due to the bootstrap sampling, and the OOB samples are used to evaluate the tree's performance without the need for an additional validation dataset.

The values for the jth variable are then randomly permuted in the OOB samples, and the accuracy is calculated again. The extent to which this permutation adversely impacts accuracy is recorded for each tree, and the average drop in accuracy across all trees is utilized to ascertain the importance of variable j within the context of the random forest algorithm.

The generalization of the OOB samples approach to estimate the importance scores of each feature leads us to the `permutation method`. The intuition behind this method is simple. If a variable is important, then we expect that, after permuting the values of the variable, the model's performance will worsen (based on some loss function $\mathcal{L}$). The larger the change in the performance, the more important is the variable.

https://doi.org/10.26219/heal.aueb.6189

Despite the simplicity of the idea, the permutation-based approach to measuring an explanatory-variable's importance is a very powerful model-agnostic tool for model exploration. Variable-importance measures obtained in this way may be compared between different models. The algorithm of the permutation method is described below.

Algorithm 5 Permutation Method Algorithm

1: Compute $\mathcal{L}^0 = \mathcal{L}(\hat{y}, X, y)$, the value of the loss function for the original data. Then for each explanatory variable X^j do:

(a) Create matrix $\underline{\mathrm{X}}^{*j}$ by permuting the jth column of $\underline{\mathrm{X}}$

(b) Compute model predictions $\underline{\hat{\mathrm{y}}}^{*j}$ based on the modified data $\underline{\mathrm{X}}^{*j}$

(c) Compute the value of the loss function for the modified data

$$L^{*j} = \mathcal{L}(\underline{\hat{\mathrm{y}}}^{*j}, \underline{\mathrm{X}}^{*j}, y)$$

(d) Quantify the importance of X^j by calculating

$$vip^j_{\text{Diff}} = L^{*j} - L^0 \quad \text{or} \quad vip^j_{\text{Ratio}} = L^{*j}/L^0$$

Note that the use of resampling or permuting data in step 1a involves randomness. Thus, the results of the procedure may depend on the obtained configuration of permuted values. Hence it is advisable to repeat the procedure several times. In this way, the uncertainty associated with the calculated variable importance values can be assessed.

https://doi.org/10.26219/heal.aueb.6189

Chapter 3

Application in Prostate Cancer Dataset I

3.1 Introduction

The first dataset in which the different regularization techniques will be implemented originates from Saifi S. and can be found in Kaggle[1]. This dataset consists of 100 patients with prostate cancer who are characterized by eight explanatory variables. All variables are continuous describing the patients' tumor. In particular the predictors are

- Radius: Measures the distance from the center of the tumor to its outer edge
- Texture: Measures the spatial arrangement of the pixel intensities (grayscale) in the image
- Perimeter: Measures the total distance around the boundary of the tumor
- Area: Measures the total surface of the tumor
- Smoothness: Measure how regular or uniform the cell appear in the tumour

[1] https://www.kaggle.com/sajidsaifi/prostate-cancer

https://doi.org/10.26219/heal.aueb.6189

- Compactness: Measures how tightly packed the cells are in the tumor
- Symmetry: Measures the regularity in the tumor's shape
- Fractal Dimension: Captures the structural complexity of the tumor

Intuitively we can claim that the greater the values of the variables radius, texture, perimeter, compactness, area and fractal dimension for a patient are, the more probable is for the patient to have a malignant tumor, while the greater the values of the variables smoothness and symmetry are, the less probable for a patient is to have a malignant tumor. The response variable is the diagnosis result (categorical), with two levels, benign and malignant respectively.

The purpose is to construct predictive models via regularization and examine their efficiency compared to "simple" models. In our case, since the response variable is binary and we want to predict whether a patient has a malignant cancer or not, the simple model will be a logistic regression model. That is, the comparisons will be based on the measurements yielded from the logistic regression model and can be considered as baseline measurements. The most representative metrics are the missclassification error (MCE) and the area under the ROC curve (AUC), while for complementary purposes the sensitivity and specificity will be calculated too.

The dataset under investigation will undergo a rigorous pre-processing phase, involving the partitioning of data into training and testing sets, primarily employing resampling techniques, with a predominant utilization of cross-validation. Given the presence of randomness in the modeling process, a total of 100 iterations will be performed, and the resultant outcomes will be aggregated to ensure statistical robustness. That is, the baseline measurements, as initially determined by logistic regression, yield the following results: MCE stands at 0.18, the AUC is recorded at 0.87 while the sensitivity (TPR) and specificity (TNR) are equal to 0.75 and 0.84 respectively.

As far as the implementation is concerned, the R software will be used. In particularly, for the penalization techniques, we have utilized the `mlr3` package in combination with the robust `glmnet` engine. The CART algorithm will be implemented using the `tidymodels` framework that provides a consistent and tidy approach to modeling and machine learning which integrates various modelling packages (e.g., glmnet,gbm, randomForest) into a unified framework, with the `rpart` engine. The random forest algorithm as introduced by Breiman will be implemented via the `randomForestSRC` package. The package uses fast OpenMP parallel processing to construct forests for regression, classification and survival analysis.

3.2 Exploratory Data Analysis

EDA is a fundamental process in data analysis that precedes formal statistical modeling. Its primary purpose is to provide insights into the dataset's characteristics and structure. In this subsection, we undertake a comprehensive EDA to reveal hidden patterns and key attributes within our data, setting the stage for subsequent analytical endeavors. The response variable is the diagnosis result and our purpose is to build predictive models in order to determine whether a patient has malignant tumor cancer or not. That is, our data set consists of 62 cases of patients with malignant tumor and the rest 38 cases reflect the patients with benign tumor.

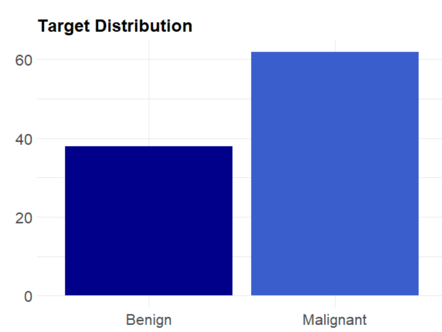

Figure 3.1: Distribution of Patients' Tumor

https://doi.org/10.26219/heal.aueb.6189

As has been already clarified, all available predictors are numeric and some representative descriptives statistics are summarized in the next Table.

Predictor	mean	median	standard deviation	minimum	maximum
radius	16.85	17.00	4.88	9.00	25.00
texture	18.23	17.50	5.19	11.00	27.00
perimeter	96.78	94.00	23.68	52.00	172.00
area	702.90	644.00	319.71	202.00	1878.00
smoothness	0.10	0.10	0.01	0.07	0.14
compactness	0.13	0.12	0.06	0.04	0.34
symmetry	0.19	0.19	0.03	0.13	0.30
fractal dimension	0.06	0.06	0.01	0.05	0.09

Table 3.1: Descriptives Statistics for Dataset I

Apparently the predictors are measured in a different scale since for instance the variable area takes values in the range $[202, 1878]$ and has standard deviation equal to 320, while fractal dimension takes values in the range $[0.05, 0.09]$ with very small standard deviation equal to 0.01. Thus we expect the measurements of the total tumors' surface between patients present high variability, while the predictors smoothness, symmetry, compactness and fractal dimension present very small variability among the patients. We can also explore how the predictors are distributed and detect any potential deviations from the Normal distribution. That is the next Figure summarizes the results. The straight lines reflect the underlying distribution while the dashed ones reflect the corresponding Normal distribution with parameters adjusted to each predictor.

As Figure 3.2 is shown, the predictors whose distribution resembles a Normal one, are perimeter, smoothness and symmetry. This can also be supported from the descriptive statistics presented earlier, where the mean is equal to the median for the variables smoothness and symmetry. Moreover the variables' distribution, texture and area, can be described as right-skewed. Before delving deeper into modelling, it is essential to explore the relationships between the predictor variables. In the remainder of this subsection, we will assess correlation scores to gain insights into

https://doi.org/10.26219/heal.aueb.6189

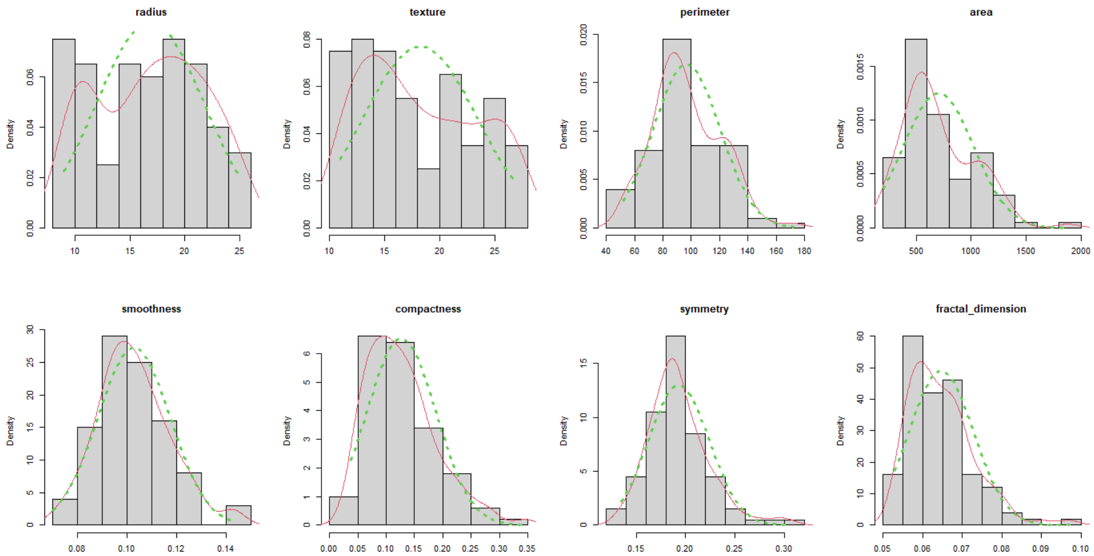

Figure 3.2: Distribution Comparison: Empirical vs. Normal for Dataset I

how these variables interact with each other. Since we anticipate monotone - not strictly linear - dependencies, the results are based on the Spearman correlation coefficient. Table 3.2 following summarizes the results.

As it can be observed, a Spearman correlation coefficient of 0.99 indicates an extremely strong positive rank-order relationship between the variables area and perimeter. This implies that, as the area of tumors increases, their perimeter tends to increase almost perfectly in sync in terms of their rankings within the dataset. Moreover moderate positive correlation (0.69) can be observed between the variables symmetry and compactness, while the variable fractal dimension shows moderate positive relationship with smoothness (0.57), compactness (0.58) and symmetry (0.56). No strong negative rank-order correlations detected between the predictors (coefficients ≥ -0.30).

	radius	*texture*	*perimeter*	*area*	*smoothness*	*compactness*	*symmetry*	*fractal_dimension*
radius								
texture	0.081							
perimeter	**-0.253**	-0.071						
area	**-0.247**	-0.070	**0.991**					
smoothness	-0.154	0.145	**0.201**	0.138				
compactness	**-0.227**	0.095	**0.526**	**0.444**	**0.576**			
symmetry	-0.072	0.059	0.189	0.112	**0.502**	**0.690**		
fractal_dimension	-0.029	0.178	**-0.233**	**-0.320**	**0.572**	**0.585**	**0.568**	

Computed correlation used spearman-method with listwise-deletion.

Table 3.2: Pairwise Spearman's Correlation Coefficients for Dataset I. The Coefficients in Bold Font Indicate Statistically Significant Correlations.

3.3 Penalization

In this section are presented the results of the penalization techniques, such as Ridge Regression, LASSO and Elastic Net. The tuning of hyperparameters, specifically lambda and alpha, will be systematically executed to ascertain their optimal values for the regularization methods under scrutiny. This fine-tuning process is imperative as it directly influences the regularization strength and the degree of feature selection, thus playing a pivotal role in model performance optimization. It is noteworthy that penalization techniques are particularly susceptible to variations in the scales of predictor variables. Therefore, prior to embarking on the regularization phase, a standardization procedure will be diligently applied to the predictor variables. This practice serves to minimize the potential for variable magnitude to unduly impact the regularization process, ensuring that the focus remains on the variable's actual predictive power rather than its scale-related effects.

Starting with the hyperparameter tuning, in ridge regression out of total 100 iterations the optimal value for lambda yielded to 0.201 in 96 iterations and 0.001 the rest 4 iterations. In LASSO, the optimal value for lambda yielded to 0.201 in 74

https://doi.org/10.26219/heal.aueb.6189

iterations and 0.001 in the remaining 26 iterations. Finally in elastic net, in 84 iterations lambda yielded to 0.461, 16 iterations yielded to 0.001 while the optimal value for alpha yielded to 0 in each of 100 iterations. Subsequently, when scrutinizing the corresponding metrics for the penalization techniques, the following outcomes are observed:

- Ridge Regression: MCE=0.13, AUC=0.91, TPR=0.93 and TNR=0.77
- LASSO : MCE=0.25, AUC=0.90, TPR=0.96 and TNR=0.70
- Elastic Net: MCE=0.19, AUC=0.90, TPR=0.95 and TNR=0.60

Upon careful examination of these results, it is evident that Ridge Regression demonstrates superior performance when compared to the other penalization techniques. Specifically, Ridge Regression exhibits a MCE of 0.13 and a higher AUC of 0.91. The results are displayed in Figure 3.3. Boxplots have been created summarizing the results of the 100 iterations.The dashed lines represent the baseline measurements yielded from the logistic regression.
In the pursuit of a comprehensive understanding of the predictive dynamics within the model, it is imperative to ascertain the individual contribution of each predictor variable to the ultimate prediction outcome. The significance of each predictor's role in the modeling process carries substantial implications for informed decision-making and refined model construction. Penalization regularization methods, by design, furnish a valuable opportunity to glean insights into the impact that each predictor variable wields in the predictive equation. By virtue of the shrinkage properties inherent to these methods, the estimated coefficients associated with each predictor can be extracted. These coefficients encapsulate the relative importance of each predictor variable in the context of the model. It is noteworthy that the magnitude of these coefficients bears profound relevance.

Specifically, the larger the absolute value of a coefficient, the more pronounced the influence of the corresponding predictor variable on the model's ability to accurately estimate the presence of malignant tumors in patients. Given that a distinct model has been fitted for each iteration, the criterion upon which the model

https://doi.org/10.26219/heal.aueb.6189

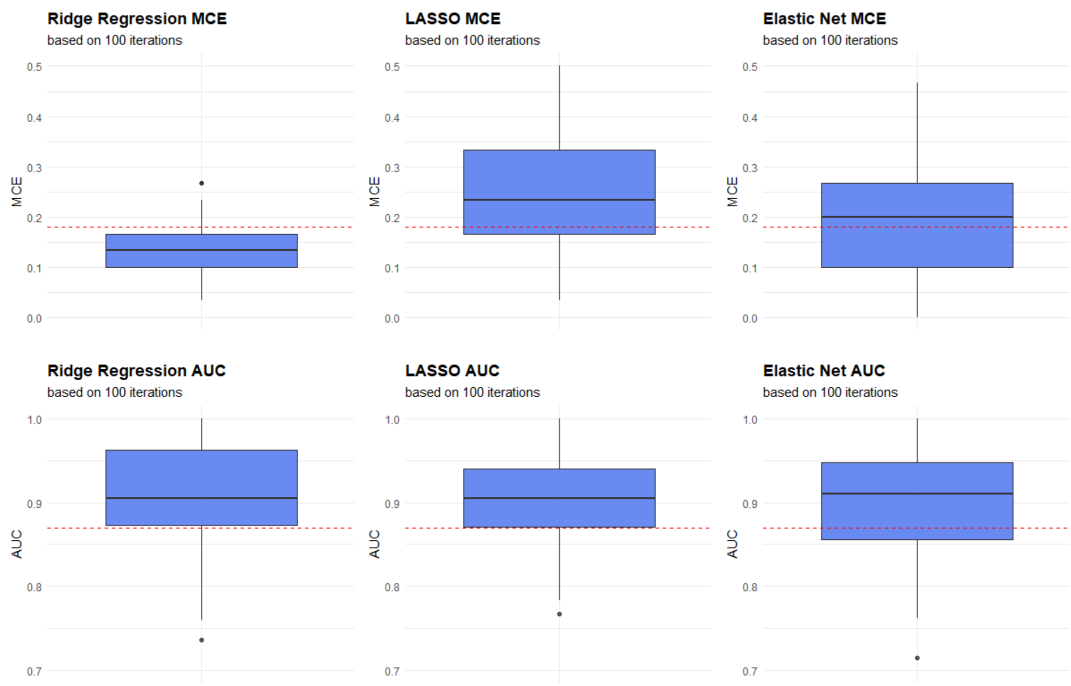

Figure 3.3: Penalization Metrics for Dataset I (MCE, AUC)

selection process is grounded for subsequent interpretation hinges on specific performance metrics. Specifically, the model selected for interpretation is the one derived from an iteration that yielded relatively lower MCE and concurrently exhibited a higher AUC for each of the three shrinkage methods. Since the task is classification, for easier interpretability the coefficients are presented in the exponential scale.

The coefficients displayed in their exponential form bear a distinctive interpretive significance within the context of logistic regression. These coefficients directly encapsulate their respective predictor variables' contribution to the odds of a tumor being malignant. In this context, coefficients with values greater than 1 signify that as the values of the corresponding predictors increase, the greater the odds of a patient are for having a malignant tumor correspondingly. Conversely, coefficients with values lower than 1 convey that as the values of these predictors

https://doi.org/10.26219/heal.aueb.6189

Predictor	Ridge Regression	LASSO	Elastic Net
area	1.45		1.30
compactness	1.54	1.32	1.29
fractal dimension	0.99		0.97
perimeter	1.55	1.42	1.36
radius	0.95		0.93
smoothness	0.98		0.99
symmetry	1.12		1.07
texture	1.21		1.03

Table 3.3: Coefficients Summary per Penalization Technique for Dataset I

increase, the lower the odds of a patient are for having a malignant tumor. It is crucial to emphasize that coefficients approaching or approximating 1 signify that the associated predictor exerts negligible influence in predicting the presence of a malignant tumor. In essence, they neither augment nor diminish the odds, thus having minimal to no impact on the predictive outcome in relation to malignant tumors.

As delineated in Table 3.3, it is evident that LASSO regularization, when applied, has selectively retained only two predictors, specifically, the compactness and the perimeter of the tumor (coefficients missing are assumed to be zero). Conversely, in the context of both Ridge Regression and Elastic Net estimations, the penalization techniques have spotlighted a consistent finding. Notably, the most salient predictors, as indicated by the magnitude of their coefficients, are once again the compactness and perimeter variables, but also the area of the tumor. These predictors exhibit the most substantial absolute coefficient values, affirming their pivotal role in influencing the predictive outcome within the context of the penalization approaches. In the Appendix section, readers will find a collection of distribution plots illustrating the values of predictor coefficients as observed after 100 iterations in their original scale. These plots offer a visual representation of the coefficient distributions for a select subset of predictors, allowing for a comprehensive assessment of their significance.

https://doi.org/10.26219/heal.aueb.6189

A discernible discrepancy in significance becomes readily apparent when contrasting, for instance, the compactness and smoothness predictors. In the case of compactness, the values of the coefficients, when viewed in the original scale, consistently manifest above-zero values. This observation signifies a tangible contribution to the model's predictive accuracy. Conversely, for the smoothness predictor, the coefficient values tend to cluster around zero. This clustering pattern suggests that the smoothness predictor exerts minimal influence on the predictive accuracy of the model and is, therefore, of limited significance in discerning malignant tumors.

An intriguing avenue of analysis lies in quantifying the significance of each predictor across the entirety of 100 iterations. This approach allows for the determination of how frequently the coefficients for each predictor deviate from zero, indicative of their respective levels of importance within the modeling context. In the case of LASSO regularization, this quantification task is relatively straightforward due to its inherent capacity for introducing sparsity by decisively shrinking coefficients to exact zero. Consequently, LASSO reveals the significance of each predictor by virtue of the exactness of this zeroing process.

Conversely, for Ridge Regression and Elastic Net, where coefficients are subjected to shrinkage but may not precisely converge to zero, a discerning threshold must be introduced. Specifically, a threshold value, such as 0.05, can be established. Any predictor whose absolute coefficient value falls below this threshold will be pragmatically set to zero. Employing this methodological approach enables the counting of instances wherein the coefficient values for each predictor do not reach zero, providing a quantitative metric of their significance within the model. The results are summarized in the Table 3.4.

Employing a threshold criterion that necessitates a predictor to exhibit non-zero coefficients in at least 80% of the total iterations, across a minimum of two penalization techniques, constitutes a prudent and rigorous approach to establish the significance of predictors. While this method bears a subjective component, it

https://doi.org/10.26219/heal.aueb.6189

Predictor	Ridge Regression	LASSO	Elastic Net
area	100	12	96
compactness	100	60	100
fractal dimension	56	26	39
perimeter	100	92	99
radius	71	27	72
smoothness	61	27	60
symmetry	76	25	80
texture	91	27	50

Table 3.4: Occurrences of non-zero coefficients over 100 iteration for Dataset I

inherently offers a layer of robustness to the results by demanding consistency across multiple regularization techniques. Under this criterion, the variables area, perimeter, and compactness emerge as noteworthy candidates for significance. These particular predictors satisfactorily meet the prescribed criteria by consistently maintaining non-zero coefficients in at least 80% of the total iterations and spanning across at least two different penalization techniques.

In order to further enrich the analysis, an extension of this approach entails examining the utilization of predictors across all 100 iterations within the three penalization techniques. Specifically, this involves observing how many predictors are actively included in each iteration of the modeling process. To facilitate this exploration, a threshold value of 0.05, as previously described, can be applied. By implementing this threshold, we can delineate the number of selected predictors in each iteration for all three penalization techniques.

In the context of Ridge Regression, it is evident that in the majority of iterations, either 6 or 7 predictors were retained, resulting in models of relatively larger complexity. Conversely, LASSO exhibited a preference for more parsimonious models, with the majority of iterations retaining only 1 or 2 predictors. Additionally, the Elastic Net approach consistently maintained 5 or 6 predictors across most iterations. Detailed visualization of these results can be referenced in the Appendix.

https://doi.org/10.26219/heal.aueb.6189

3.4 Early Stopping

Classification Trees

In this section, we delve into the application of early stopping regularization approaches, commencing with the Classification and Regression Trees (CART) algorithm. To facilitate this analysis, we employ the `tidymodels` framework. `Tidymodels` is a comprehensive and versatile R package that streamlines the process of creating and evaluating machine learning models, making it particularly well-suited for the task at hand. Within the context of CART regularization, our objective includes hyperparameter tuning, a critical step for optimizing model performance. Since classification trees align with the concept of early stopping regularization, it entails finding the optimal number of splits in the final, fine-grained tree structure.

To achieve this, we engage in the tuning of the cost complexity parameter, a crucial determinant for tree pruning, and the tree depth, which refers to the maximum depth or levels the tree can attain. Tree depth, in essence, defines the number of decision nodes or levels within the tree structure, which directly impacts the granularity of the model's predictions. The higher the tree depth, the more intricate and detailed the decision boundaries become.

In tandem with the hyperparameter tuning process, we continue to utilize the cross-validation resampling technique, considering the inherent randomness in the modeling process. This methodology ensures the robustness and reliability of our results by repeating the modeling process across 100 iterations and subsequently aggregating the outcomes.

In addition, it's important to mention that standardization of predictors is not typically required for the machine learning algorithms. These techniques are inherently robust to variations in predictor scales, and their performance is not significantly impacted by differences in variable scaling.

https://doi.org/10.26219/heal.aueb.6189

To comprehensively assess the functionality of the algorithm, we specify the task as classification, with the splitting criterion based on the information gain. Subsequently, a range of logical combinations for the cost complexity parameter and tree depth will be explored to grow respective trees, with the log loss[2] function being recorded at each iteration. This venture will be implemented utilizing the `dials` package in R, which contains infrastructure to create and manage values of tuning parameters for the tidymodels packages. It provides a reasonable and sensible combinations of values for hyperparameter tuning, but we will also make adjustments to these predefined values based on the specific characteristics of our dataset as following:

cost complexity	**tree depth**
0.0000000001	1
0.0000000178	2
0.00000316	3
0.000562	4
0.1	5

Table 3.5: Parameter Grid for Optimizing Regression Trees Hyperparameters for Dataset I

As Table 3.5 is shown, in each iteration $5*5 = 25$ different classification trees will be constructed. The culmination of this exploration will reveal the combination of hyperparameter values that minimizes the log loss, serving as the tuned values for our model. After conducting 100 iterations, the cost complexity parameter converged to the value of 1e-10 in 86 iterations, 3.16e-6 in 4 iterations, while in the remaining 10 iterations, it converged to 0.1. In terms of tree depth, the results showed that in 57 iterations, a decision stump with one split was favored, in 40 iterations a depth of 2 was selected, and in 3 iterations a depth of 3 was chosen. Notably, depths of 4 or 5 were not favored in any of the iterations. It is sensible that with a small tree depth, a very small cost complexity parameter value is anticipated, and may no pruning is needed at all.

[2] Log loss is a common evaluation metric for binary classification and can be computed as $ll = -\frac{1}{N}\sum_{i=1}^{N}[y_i ln p_i + (1-y_i)ln(1-p_i)]$

https://doi.org/10.26219/heal.aueb.6189

During the tuning process, it was observed that several combinations of hyperparameters led to similar results in terms of the estimated log loss. Specifically, configurations involving tree depths of 1 and 2 almost consistently yielded similar log loss values. In accordance with the methodology of the tidymodels framework, when faced with such equivalences, the simpler model is preferred. Consequently, the decision was made to select the decision stump, characterized by a minimal significance cost complexity parameter. It is imperative to consider this aspect when interpreting the results obtained. A visualization of a classification tree with depth 2 can be found in the Appendix.

The results obtained from the CART algorithm are moderate. The MCE is 0.17, the AUC is 0.82, TPR is 0.77, and the TNR is 0.86. While the MCE is slightly better than that of logistic regression, the AUC and TPR have not improved significantly. To visually compare these results with the baseline measurements and the best-performing technique, corresponding boxplots will be displayed.

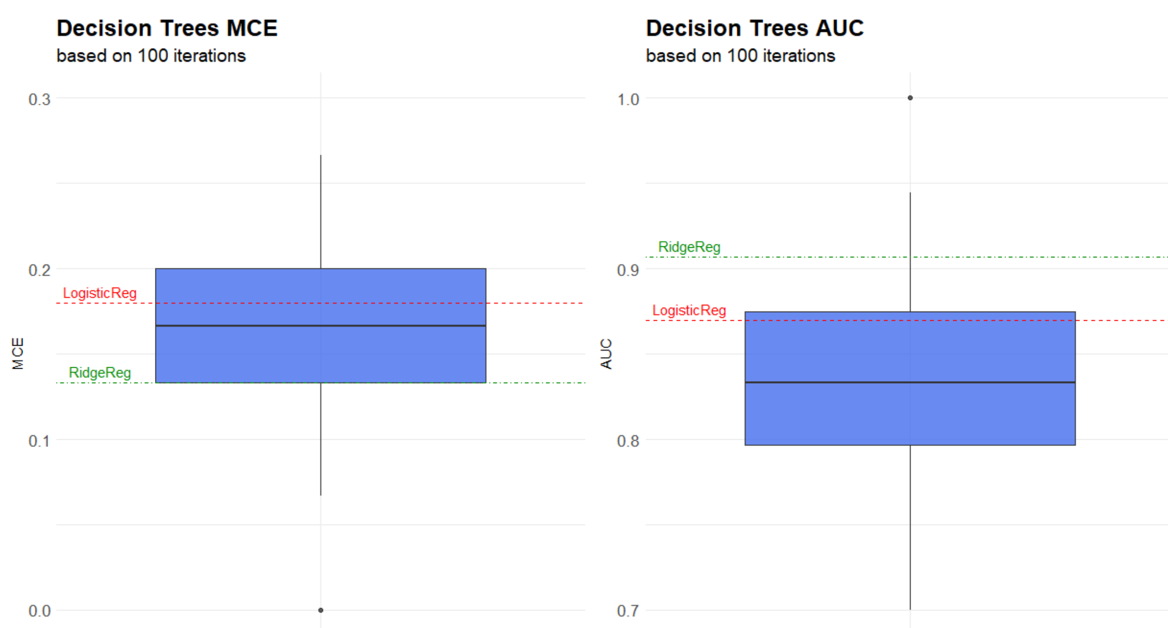

Figure 3.4: Classification Trees Metrics for Dataset I (MCE, AUC)

https://doi.org/10.26219/heal.aueb.6189

In these boxplots, the dashed line represents the metrics derived from logistic regression, and the dot-dashed lines represent the metrics derived from ridge regression, as it has outperformed all other techniques. This approach will be used throughout the remainder of the application context.

We will now investigate the individual contributions of each predictor to the final accuracy predictions. For this purpose, we make use of the `vip` package and employ the model-based variable importance measure provided by the `rpart` engine. In particular, this importance measure calculates the overall importance of a variable as the sum of the goodness of split measures for each split in which it was the primary variable, along with the goodness for all splits in which it served as a surrogate variable.

Furthermore, the package rescales the scores to ensure they sum up to 100 and disregards predictors whose proportion is less than 1%. It is essential to exercise caution when interpreting the final results, as they represent an aggregation of 100 iterations. Therefore, emphasis should be placed on the magnitude of the bars rather than on the individual scores.

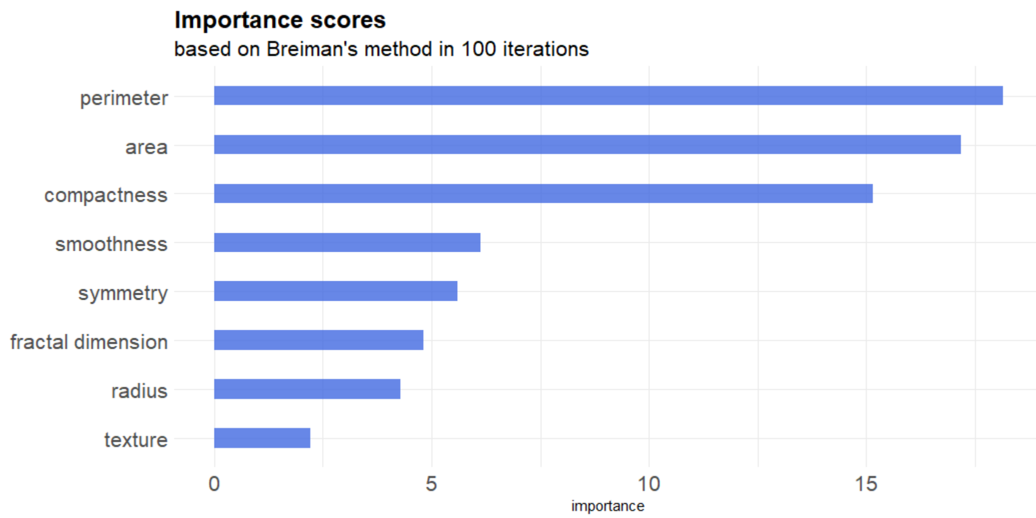

Figure 3.5: Relative Importance of Predictors in Classification Trees for Dataset I

https://doi.org/10.26219/heal.aueb.6189

As illustrated in Figure 3.5, a clear and discernible pattern emerges when examining the importance of predictors for accurate predictions. In ascending order of importance, the most influential predictors are observed to be compactness, area, and perimeter. These findings align remarkably with the observations made during our exploration of penalization techniques earlier in this study.

Gradient Boosted Classification Trees

We now conclude our exploration of early stopping regularization approaches with the implementation of the variant of the boosting algorithm, the gradient boosted classification trees. For this endeavor we will utilize the `mlr3` and `gbm` packages in R. The process begins by specifying the task as classification, utilizing cross-validation as the resampling method, and employing the log loss function as the performance criterion. This performance measure will also guide our tuning process, where we will optimize two key hyperparameters: the learning rate ν and the total number of boosting rounds (boosted trees). Since we need to establish criteria for stopping the iteration process based on our performance metric, tuning the learning rate and identifying the optimal number of boosting rounds will enable us to execute this essential aspect of the algorithm, achieving the early stopping regularization effect described.

The specified ranges are [0.01, 0.3] for the learning rate and [5, 100] for the boosted rounds. While these values might seem somewhat arbitrary, they are selected to be both sensible and reasonable, taking into consideration the characteristics of our dataset, including the number of observations and predictors. The chosen range for the learning rate provides flexibility in exploring different levels of adaptation during boosting, and the range for the boosted rounds allows us to examine a variety of boosted trees.

After conducting 100 iterations of the gradient boosting algorithm, our analysis unveiled a consistent preference for smaller values in the total number of boosted classification trees within the ensemble. Specifically, in 38 iterations, the algorithm selected 11 trees, while in 18 and 16 iterations, it opted for 18 and 25 trees, respectively. Moreover, the learning rate predominantly fell within the range of [0.19, 0.29]. These observations suggest that the dataset's dimensionality benefits from a more moderate number of boosting rounds and learning rate. The graphical representation of the tuned results is available in the Appendix

Regarding the metrics obtained from the gradient boosted classification trees algorithm, the results fall within the range of moderate performance. Specifically, the MCE is equal 0.16, the AUC is equal to 0.91 while the TPR and TNR are equal to 0.87 and 0.77 respectively. While these results indicate an improvement over the simple logistic regression, they slightly lag behind the dominant technique, which is Ridge Regression. To facilitate a comprehensive comparison, the results are once again visualized, with corresponding lines for comparison purposes.

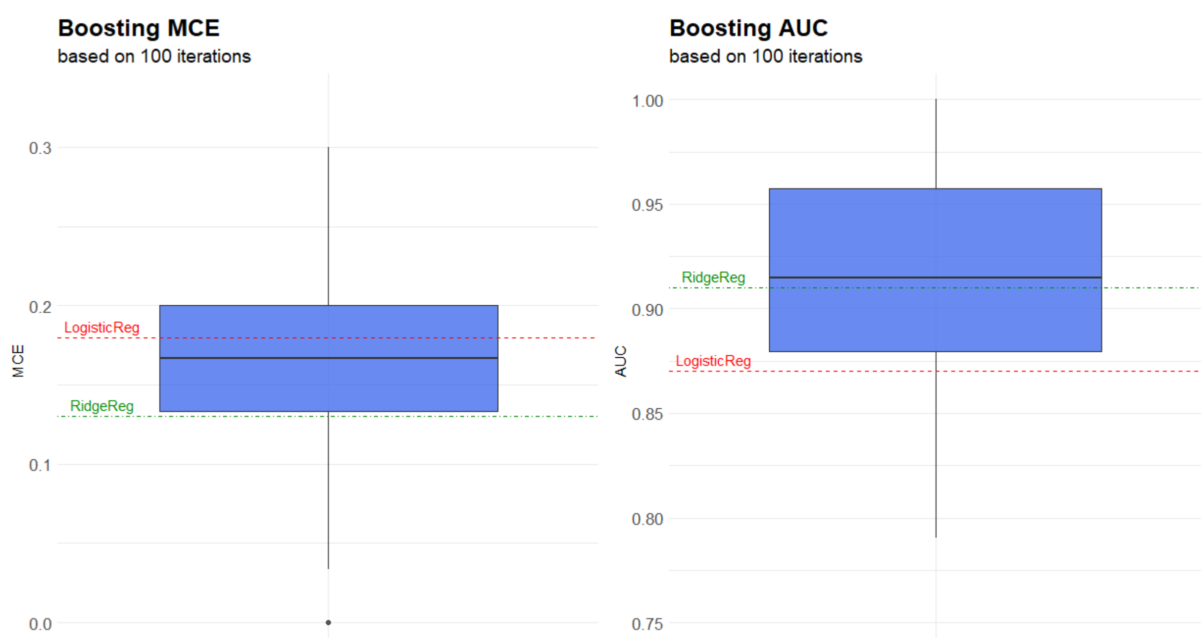

Figure 3.6: Gradient Boosted Classification Trees Metrics for Dataset I (MCE, AUC)

https://doi.org/10.26219/heal.aueb.6189

In the subsequent analysis, we will delve into the importance scores generated by the model based approach. The measure of improvement employed is the MCE, and therefore, the scores obtained in each iteration represent the improvements attributed to the corresponding predictor when utilized as a splitting variable in the tree ensemble. It is urgent we point out, the utilization of the surrogate variables is particularly beneficial in the context of single decision trees, where the number of opportunities for variables to participate in splitting is limited by the tree's size, its necessity diminishes in boosting. In boosting, the abundance of splitting opportunities significantly increases, reducing the need for surrogate unmasking. The final importance scores are computed as the average of these scores across 100 iterations.

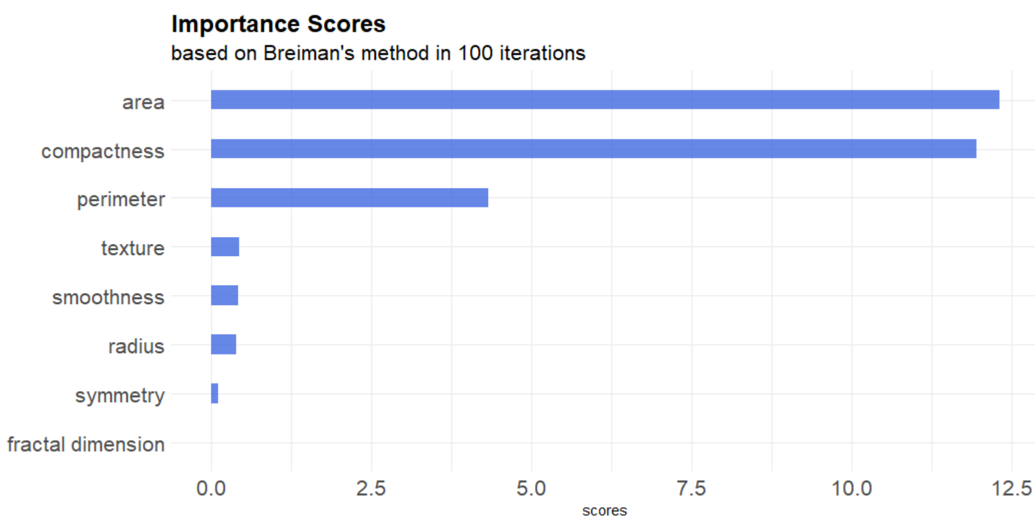

Figure 3.7: Relative Importance of Predictors in Gradient Boosted Classification Trees for Dataset I

As depicted Figure 3.7, the most influential predictors are the area, compactness, and perimeter. Area and compactness of tumor patients' data received similar high scores above 10, while the remaining predictors scored below 1. Notably, the importance of the fractal dimension was computed as zero, suggesting it was never selected as a splitting variable in the gradient boosting tree ensemble.

https://doi.org/10.26219/heal.aueb.6189

3.5 Ensembling

In our pursuit of machine learning regularization approaches, we are now turning our attention to the implementation of the ensembling approaches, and in particularly on the Random Forest algorithm for classification task. To accomplish this, we have harnessed the capabilities of the `randomForestSRC` R-package along with the `MLmetrics` package, which implements the random forest algorithm introduced by Breiman, 2001. The primary hyperparameter to be tuned in this context is the number of predictors considered at each splitting step. Additionally, we will optimize the number of total trees, the node depth, and the minimum node size. For clarity, let's briefly define the terms `node depth` and `minimum node size`. Node depth refers to the maximum number of levels or splits a decision tree can attain. On the other hand, the minimum node size denotes the minimum number of observations required within a node for further splitting to occur.

In each of the 100 iterations, we will employ a 10-fold cross-validation resampling method on the training observations. Subsequently, we will randomly select 50 values from a set of potential values for the four hyperparameters. The mean log loss will be computed for each iteration, and the hyperparameter values resulting in the lowest log loss will be designated as the optimal tuned values. Throughout this process, the splitting criterion assigned is the entropy. In particularly, as far as the hyperparameters tuning is concerned, the range of possible values for the hyperparameters has been partially derived from the primary research paper that has inspired this thesis. These values have been thoughtfully selected to ensure that the hyperparameter tuning process explores a range of sensible and relevant options for optimizing the random forest algorithm.

The results obtained from the tuning process for the random forest algorithm exhibit distinct patterns. Notably, the number of predictors considered at each split predominantly favors a specific range, with 6 and 7 predictors being the most frequently favored choices. Additionally, the minimum node size required for initiat-

https://doi.org/10.26219/heal.aueb.6189

hyperparameter	range
number of predictors in each split	[2,7]
total number of trees	[250,1000]
minimum node size	[2,10]
node depth	[7,20]

Table 3.6: Parameter Grid for Optimizing Random Forest Hyperparameters in Dataset I

ing subsequent splits tends to cluster around lower values, particularly within the range of [2, 6]. Conversely, when considering node depth, the algorithm demonstrates a preference for higher values within the predefined range, with a notable focus around a depth value of 18. Furthermore, the analysis indicates a moderate number of trees being generated, as evidenced by a mean value of 543 across the 100 iterations. These results are visually presented in the appendix.

The ensemble of trees, as derived from the random forest algorithm, has produced the following results: a MCE of 0.20, an AUC of 0.85, a TPR of 0.91 and a TNR of 0.63. The obtained results are not satisfactory, as both the MCE and AUC metrics performed worse than the baseline measurements.

In addition, it is of paramount importance to scrutinize the contributions of individual predictor variables to the final predictions, a task facilitated by the capabilities of the random forest algorithm. To accomplish this, we harness the specific feature that the algorithm, namely the evaluation of trees on the Out-Of-Bag (OOB) samples. In this endeavor, we will compute the variance importance scores based on the permutation method, which involves permuting the OOB cases.The importance scores to be derived are calculated based on the difference in the missclassification error. To ensure comprehensive and robust results, we will store the importance scores in each of the 100 iterations and subsequently aggregate them.

The `randomForestSRC` package extends its functionality further by offering an additional argument. In addition to computing importance scores for overall accurate predictions, it provides the capability to compute importance scores for each

 https://doi.org/10.26219/heal.aueb.6189

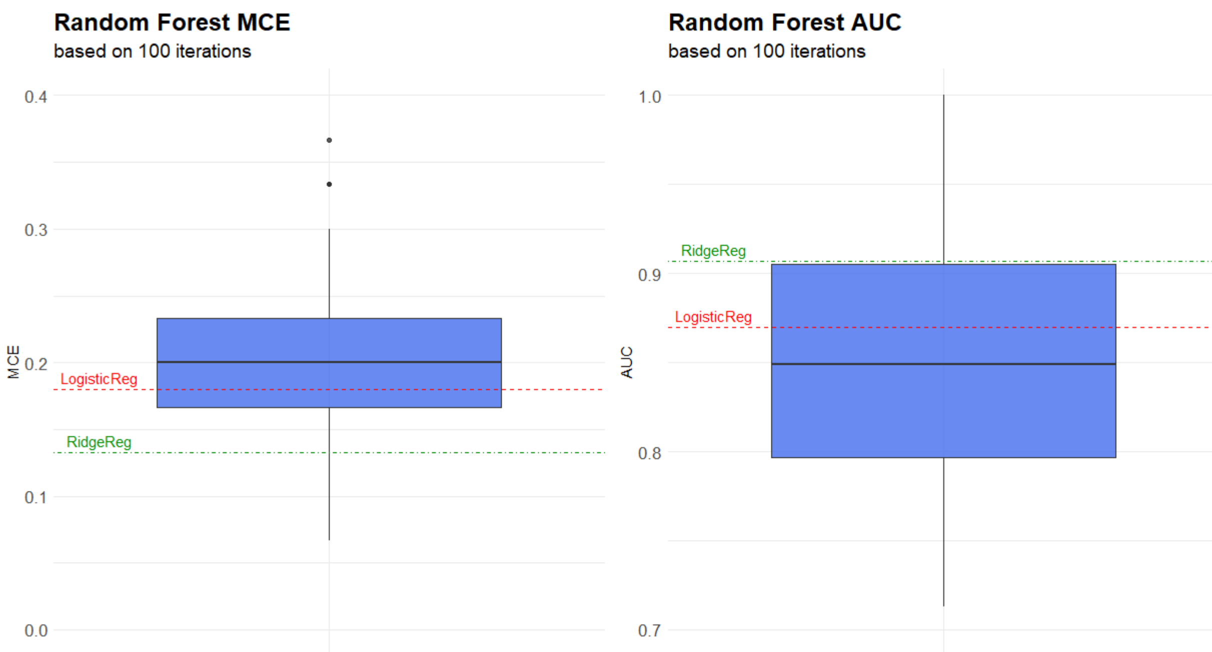

Figure 3.8: Random Forest Metrics for Dataset I (MCE, AUC)

predictor with respect to each class label individually. This fine-grained analysis, known as Conditional Variable Importance, can be achieved by considering only those cases associated with the specified class labels, which in our case are `malignant` and `benign`. This approach will allow us to gain a deeper understanding of how each predictor contributes to the classification of these specific class labels within the random forest framework. The results concerning the importance scores for the overall prediction accuracy are summarized in the Figure 3.9, while the VIMP plots can be found in the Appendix.

We need to point out that, if a variable has very little predictive power permuting the OOB cases may lead to a slight increase in accuracy duo to random noise. This can give rise to small negative scores, as it can be observed in our case, which can be assumed to be equal to zero.

 https://doi.org/10.26219/heal.aueb.6189

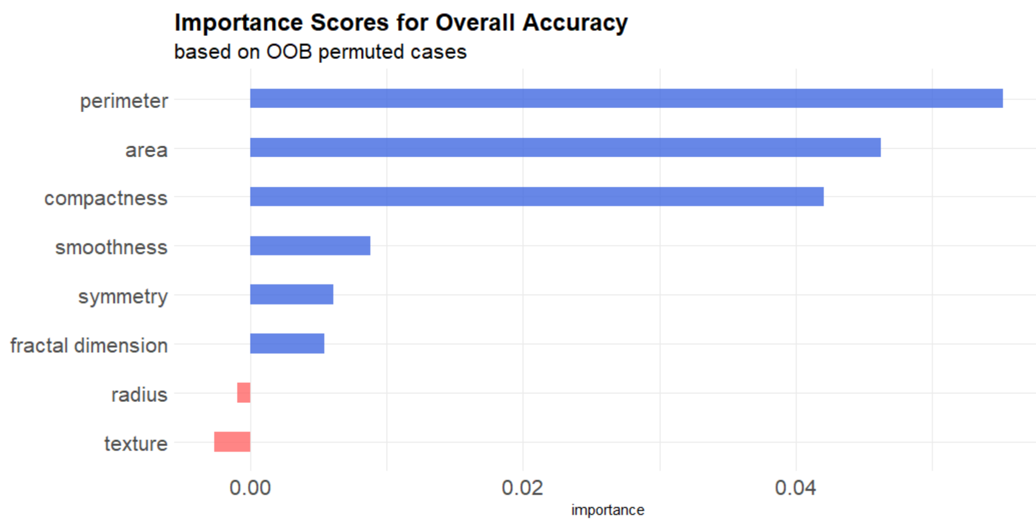

Figure 3.9: Relative Importance of Predictors in Random Forest for Dataset I

Whether we focus on overall predictions or specifically on predictions for the `malignant` and `benign` class labels, it becomes evident that the most significant predictor variables are consistently the same: perimeter, area, and compactness. These predictors consistently exhibit a high level of importance in shaping the accuracy of the predictions.

https://doi.org/10.26219/heal.aueb.6189

3.6 Results

In summary, our exploration of various regularization techniques has shed light on the performance and characteristics of the models under scrutiny. Ridge Regression emerged as the top performer, excelling in terms of achieving the lowest MCE and the highest AUC when compared to LASSO, Elastic Net, Classification Trees, Random Forest, and Gradient Boosted Trees. However, it's worth noting that LASSO, despite its higher MCE, stood out with the highest TPR, a crucial factor in cases where accurate predictions of malignant tumors are vital. The influential predictors consistently identified across these techniques were the area, perimeter, and compactness of the tumor, reaffirming their significance in predictive modeling.

Furthermore, our investigation into Classification Trees revealed moderate results. While CART algorithm showed lower MCE than logistic regression, it fell short of Ridge Regression's superior performance. Notably, the AUC for CART remained relatively low, emphasizing its limitations in accurately distinguishing between benign and malignant tumors. On the other hand, the Random Forest algorithm produced suboptimal results, failing to achieve competitive MCE, AUC, and TNR metrics when compared to the other models in our analysis. Finally, the Gradient Boosted Classification Trees delivered moderate results, offering a slight improvement over simple logistic regression but falling short of Ridge Regression's outstanding performance.

Collectively, our findings underscore the importance of the area, compactness, and perimeter measurements in accurately predicting malignant tumors and highlighting the efficient utilization of regularization in clinical biostatistics. The table below presents a comprehensive overview of all the techniques' results, facilitating a direct comparison of their performance metrics and insights to aid in assessing their respective strengths and weaknesses.

https://doi.org/10.26219/heal.aueb.6189

Regularization	Technique	MCE	AUC	TPR	TNR
None	Logistic Regression	0.18	0.87	0.75	0.84
Statistical	Ridge Regression	0.13	0.91	0.93	0.77
	LASSO	0.25	0.90	0.96	0.70
	Elastic Net	0.19	0.90	0.95	0.60
Machine Learning	Classification Trees	0.17	0.82	0.78	0.86
	Random Forest	0.20	0.85	0.91	0.63
	Gradient Boosted Classification Trees	0.16	0.91	0.87	0.77

Table 3.7: Overall Metrics of Regularization Techniques for Dataset I

https://doi.org/10.26219/heal.aueb.6189

Chapter 4

Application in Prostate Cancer Dataset II

4.1 Introduction

The second data set under consideration, upon which the regularization techniques will be applied, originates from a study conducted by Stamey et al. This study aimed to explore the levels of prostate-specific antigen (PSA) in 97 patients who were scheduled to undergo radical prostatectomy. PSA is a well-known biomarker in prostate cancer diagnosis, as it is a protein produced by the prostate gland. Typically, the PSA test, a blood test, is administered to measure the concentration of PSA in a blood sample, helping in the detection and monitoring of potential prostate tumors. In this dataset, the authors sought to examine the correlation between the logarithm of PSA levels and eight clinical variables. Notably, the majority of these predictors are numeric, with seven out of the eight being continuous variables, while the remaining predictor is binary. In the subsequent sections, we will delve into a detailed exploration of these predictors, their relationships, and the application of regularization techniques to enhance our understanding of how these variables collectively impact PSA levels and, by extension, prostate cancer diagnosis. Let us begin with the acquaintance with the predictors:

https://doi.org/10.26219/heal.aueb.6189

- Cancer volume: Captures the maximum size of the invasive component of the primary tumor on the pathological examination. It is measured in a log scale.

- Prostate weight: Measures the weight of the prostate gland. It is measured in a log scale

- Age: Age of the patient .

- Benign prostatic hyperplasia amount: Measures the prostate gland enlargement. The patient's prostate gland is enlarged but not cancerous. It is measured in a log scale.

- Seminal vesicle invasion: Captures the presence of the prostate cancer in the areolar connective tissue around the seminal vesicles and outside the prostate gland.

- Capsular penetration: Captures the spread of the prostate cancer to the outer wall of the gland. It is measured in a log scale.

- Gleason score: Results from the Gleason grading system and refers to how abnormal the prostate cancer cells look and how likely the cancer is to advance and spread. A lower Gleason grade means that the cancer is slower growing and not aggressive. Ranges between 1 and 5.

- Percent of Gleason scores 4 or 5: Capture the percentage of the prostate cells in which has been assigned a Gleason score equal to 4 or 5.

Intuitively, we anticipate certain patterns in the relationships between the predictors and the logarithm of PSA levels in this dataset. Specifically, we expect that as the values of predictors such as cancer volume, prostate weight, capsular penetration, age, Gleason score, and the percent of Gleason scores increase, the logarithm of PSA levels will also tend to increase. This expectation aligns with the notion that these clinical factors may have a direct or indirect impact on PSA production

https://doi.org/10.26219/heal.aueb.6189

or secretion. Similarly, for a patient with a positive observation regarding seminal vesicle invasion, we anticipate higher logarithmic PSA values. On the other hand, when considering the predictor amount of benign hyperplasia, our expectations are less conclusive. While we may anticipate that higher values of this predictor could be associated with higher logarithmic PSA values, the relationship is not as straightforward as those assumed with the previous predictors. This variable may exhibit more nuanced or complex interactions with PSA levels, and as such, we approach her with a degree of uncertainty.

In a manner similar to the previous chapter, we will employ various regularization techniques described in Chapter 2, in order to analyze the dataset and assess their impact on predictive performance. Our primary objective is to compare the results obtained from these regularization techniques against a baseline measurement derived from a ”simple” model, which, in this case, takes the form of a multiple linear regression model. To evaluate and contrast the effectiveness of these methods, we will rely on two key performance metrics: the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). The regularization technique that outperforms the multiple linear regression model in terms of predictive accuracy - exhibits the lowest MSE and MAE - will be considered as the optimal choice for this particular dataset. These metrics will enable us to gauge the predictive accuracy and robustness of each technique .

To maintain consistency with the regularization approaches, we will conduct 100 iterations, randomly splitting the dataset into a training set consisting of 67 observations and a test set with the remaining 30 observations. This iterative approach will help us account for the inherent randomness in the splitting process. Our initial findings reveal that the mean baseline measures derived from the multiple linear regression model yield an MSE of 0.60 and an MAE value of 0.58.

The implementation of the regularization algorithms for this dataset will follow a consistent framework, much like the one employed in Chapter 3.

https://doi.org/10.26219/heal.aueb.6189

4.2 Exploratory Data Analysis

We will begin our exploratory data analysis, providing descriptive statistics to offer an initial understanding of the predictor measurements. Our primary target variable of interest is the logarithm of PSA. In the following table, we present descriptive statistics for the seven numeric predictors along with the response variable. It's important to note that the data in this analysis are presented according to the study's scale where taken, thus the predictors prostate weight, cancer volume, benign hyperplasia amount, capsular penetration, and PSA, are in a logarithmic scale.

Predictor	mean	median	standard deviation	min	max
PSA	2.48	2.59	1.15	-0.43	5.58
Cancer volume	1.35	1.45	1.18	-1.35	3.82
Prostate weight	3.63	3.62	0.43	2.37	4.78
Age	63.87	65.00	7.45	41.00	79.00
Benign hyperplasia amount	0.10	0.30	1.45	-1.38	2.33
Capsular penetration	-0.18	-0.79	1.40	-1.38	2.90
Gleason	6.75	7.00	0.72	6.00	9.00
Perc. gleason scores 4 or 5	24.38	15.00	28.20	0.00	100.00

Table 4.1: Descriptives Statistics for Dataset II

Let's delve deeper into the descriptive statistics of these variables. The mean value of PSA for the 97 patients is approximately 11.94 when expressed in the original scale. In terms of age, the average age of the patients is around 64 years, with ages ranging from 41 to 79 years old. The Gleason score, which is a crucial factor in prostate cancer diagnosis, has a mean value of 6.75 and falls within the range of 6 to 9, as discussed in the introduction of this chapter. Moving on to cancer volume, the mean value is approximately 3.86 in the logarithmic scale, with a standard deviation of 1.18. Prostate weight, another vital predictor, has an average value of about 37 grams in the original scale. In the logarithmic scale, these observations exhibit a standard deviation of approximately 0.43.

https://doi.org/10.26219/heal.aueb.6189

Just as in the previous dataset analysis, we will visualize the distribution of the numeric predictors and the response variable using density plots. These plots provide valuable insights into the underlying distribution of each predictor. In the following Figure, the solid line represents the observed distribution, while the dashed line reflects the normal distribution with parameters estimated based on the values of the respective predictor. By examining these plots, we can identify any deviations from the normal distribution and detect potential skewness in the data.

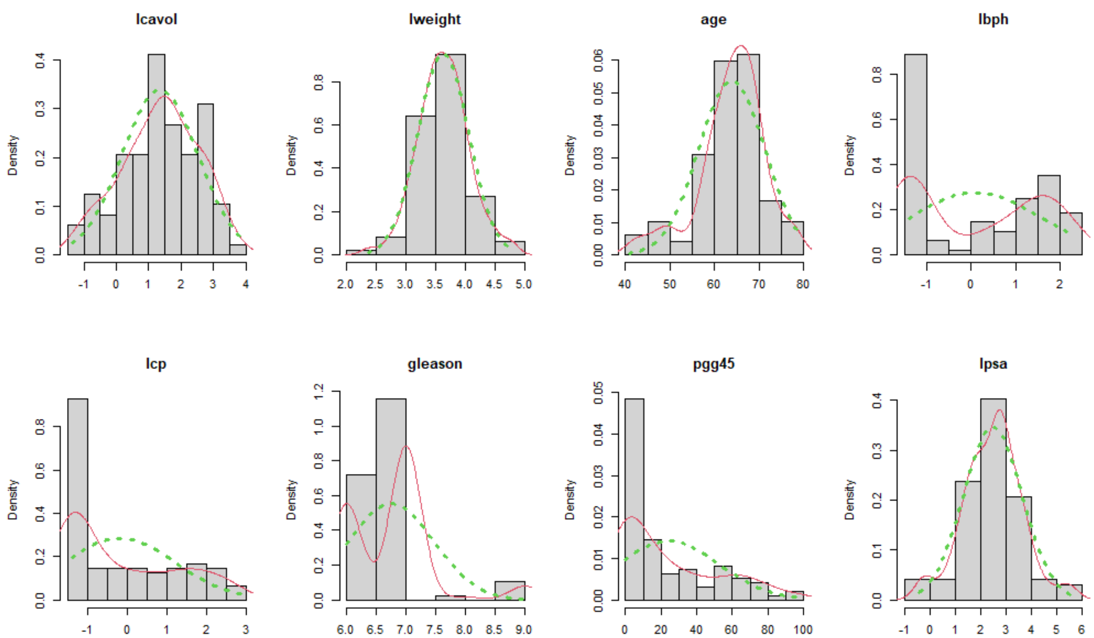

Figure 4.1: Distribution Comparison: Empirical vs. Normal for Dataset II

Regarding the target variable, the logarithm of PSA appears to follow a nearly normal distribution, suggesting that the original scale values might exhibit a right-skewed distribution. Among the predictors, cancer volume, prostate weight, and age show distributions that more closely resemble normality. The remaining predictors exhibit significant right-skewness, which may need to be addressed during modeling to ensure the assumptions of certain algorithms are met.

https://doi.org/10.26219/heal.aueb.6189

The binary predictor variable ”seminal vesicle invasion” reveals an interesting relationship within the dataset. Among the 97 patients, 21 of them had cancer cells extracted from the prostate gland, while the remaining 76 did not exhibit this condition. To explore this relationship further, we conducted an analysis comparing the target variable (logarithm of PSA) and the other numeric predictors across the two categories defined by seminal vesicle invasion. As expected, patients with positive seminal vesicle invasion had significantly higher PSA values than those without. Additionally, substantial differences were observed in the predictors of cancer volume, capsular penetration, and Gleason scores. Patients with the presence of prostate cancer around the seminal vesicles exhibited higher values for these predictors, aligning with the anticipated relationship between these variables. For a detailed visual representation of these findings, boxplots have been provided in the Appendix (A.7). These insights help elucidate the potential influence of seminal vesicle invasion on the predictive factors under consideration.

In the aftermath of our exploratory data analysis and the examination of relationships between the categorical variable ”seminal vesicle invasion” (svi) and the predictors, we turned our attention to understanding the associations among the numerical predictors themselves. Given our expectation of monotonic relationships rather than strictly linear ones, we opted for Spearman correlation coefficients to assess these connections. Our analysis uncovered several noteworthy findings:

- A very strong and anticipated relationship existed between Gleason scores and the percentage of cells assigned values of 4 or 5 (correlation coefficient: 0.87)
- Positive relationships were identified between:
  - Gleason scores and capsular penetration (correlation coefficient: 0.579)
  - Percentage of cells assigned Gleason scores of 4 or 5 and capsular penetration (correlation coefficient: 0.663)
  - Capsular penetration and cancer volume (correlation coefficient: 0.663)

 https://doi.org/10.26219/heal.aueb.6189

- Moderate relationships were observed between various other predictors, with correlation coefficients below 0.50.

Finally, concerning the target variable (logarithm of PSA), we found the following relationships:

- Cancer volume exhibited the highest positive correlation (correlation coefficient: 0.70)
- Moderate relationships were identified with predictors:
  - Capsular penetration (correlation coefficient: 0.52)
  - Percentage of cells assigned Gleason scores of 4 or 5 (correlation coefficient: 0.52)
  - Gleason scores (correlation coefficient: 0.47)

These Spearman correlation coefficients provide valuable insights into the relationships among the predictors and the target variable, helping us understand the potential influence of each variable in our modeling process.

	lcavol	*lweight*	*age*	*lbph*	*lcp*	*gleason*	*pgg45*	*lpsa*
lcavol								
lweight	0.291							
age	0.192	0.402						
lbph	0.014	0.493	0.341					
lcp	0.663	0.182	0.151	0.023				
gleason	0.487	0.144	0.252	0.107	0.579			
pgg45	0.497	0.171	0.270	0.100	0.663	0.873		
lpsa	0.700	0.454	0.215	0.171	0.521	0.473	0.520	

Computed correlation used spearman-method with listwise-deletion.

Table 4.2: Pairwise Spearman's Correlation Coefficients for Dataset II. The Coefficients in Bold Font Indicate Statistically Significant Correlations.

https://doi.org/10.26219/heal.aueb.6189

4.3 Penalization

Our journey into regularization techniques begins with the penalization approach, where we will apply a systematic process to prepare the data. In particular, data standardization will be employed to ensure uniformity across variables, a crucial step as penalization techniques are sensitive to variations in variable scales. The application of the penalization algorithms will be facilitated using the `mlr3` package, as we have already mentioned. The hyperparameters in focus for tuning are lambda and alpha. The penalized models will again constructed in 100 iterations, while as a resampling technique, we have chosen a 10-fold CV approach. This method allows us to select the most suitable values for the hyperparameters, optimizing the regularization models' predictive capabilities effectively.

In our rigorous tuning process for the penalization approaches, we obtained valuable insights from the 100 iterations. For Ridge Regression, the lambda hyperparameter exhibited varying degrees of importance. It was selected as 0.001 in 8 iterations, 0.084 in 57 iterations, 0.17 in 27 iterations, 0.27 and 0.374 once. This variability in lambda values suggests that Ridge Regression is relatively flexible and can adapt its regularization strength to different subsets of the data. In the LASSO regularization technique, the tuning process produced outcomes characterized by similar hyperparameter values, albeit with varying frequencies. Specifically, the lambda hyperparameter was chosen as 0.001 in 47 iterations, 0.084 in 50 iterations, and 0.17 in the remaining 3 iterations.

In the context of Elastic Net, the results exhibit a distinct pattern. Over the course of 100 iterations, only two combinations of hyperparameters emerge as prominent. Specifically, in 64 iterations, alpha assumes a value of 0, accompanied by a corresponding lambda of 0.293. This preference for L2 norm penalization is evident in the majority of cases, highlighting its significance. In contrast, alpha takes on a value of 1 in 35 iterations, introducing L1 norm penalization with a corresponding lambda of 0.001. Finally, one remaining iteration yields a combination with

https://doi.org/10.26219/heal.aueb.6189

alpha at 0.777 and lambda at 0.001, demonstrating a unique situation. The results yielded after 100 iterations are aggregated and presented below:

- Ridge Regression: MSE=0.56 and MAE=0.57
- Lasso: MSE=0.57 and MAE=0.59
- Elastic Net: MSE=0.45 and MAE=0.52

As it can be observed, the results from our penalization techniques indicate that all three approaches have improved predictive accuracy when compared to the baseline metrics derived from multiple linear regression. However, Elastic Net stands out as the top performer, boasting the lowest MSE and MAE values among the three statistical approaches. To provide a comprehensive visual representation of these results, the Figure 4.2 illustrates the performance metrics for each penalization technique, with dashed lines representing the measures from the linear regression model as a reference point.

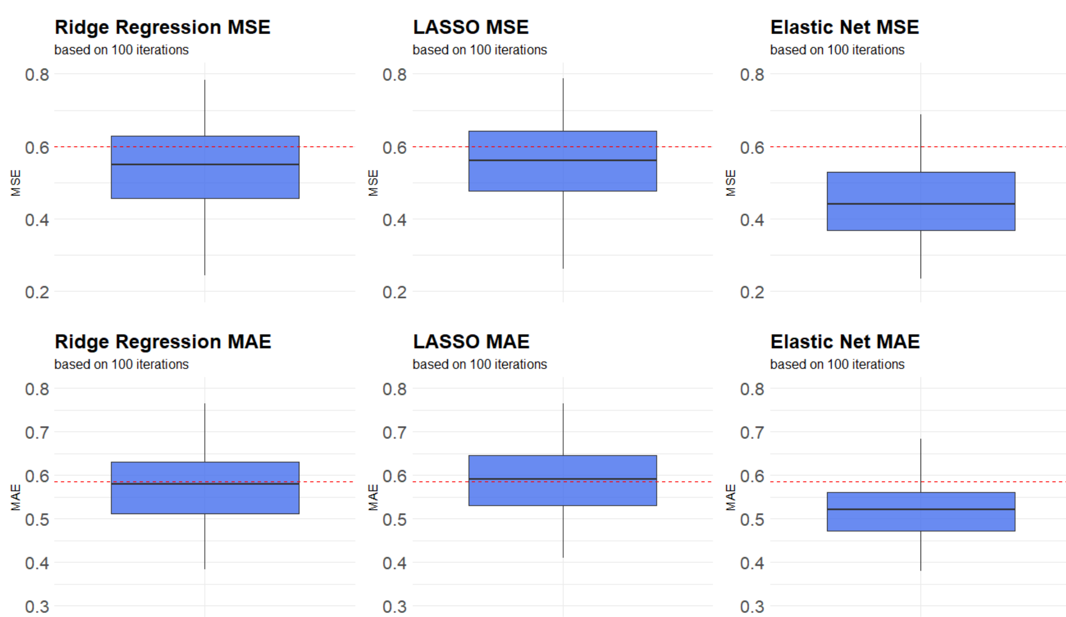

Figure 4.2: Penalization Metrics for Dataset II (MSE, MAE)

https://doi.org/10.26219/heal.aueb.6189

The interpretation of models constructed using the penalization techniques is straightforward, as the contribution and influence of each predictor to the final prediction are reflected by the corresponding coefficient values. In this analysis, we focus on the models that yielded the best prediction measurements out of 100 iterations, as determined by MSE and MAE. These models provide the most appropriate basis for interpretation. Table 4.3 presents the coefficients obtained from the models in each technique that yielded the best predictive performance.

Predictor	**Ridge Regression**	**LASSO**	**Elastic Net**
cancer volume	0.50	0.66	0.67
prostate weight	0.27	0.20	0.23
age	-0.13		-0.20
benign hyperplasia amount	0.10	0.02	0.22
seminal vesicle invasion	0.64	0.43	1.03
capsular penetration	-0.00		-0.09
Gleason	0.07	0.01	0.13
Gleason scores 4 or 5	0.07		-0.00
MSE	0.24	0.26	0.23
MAE	0.38	0.40	0.38

Table 4.3: Coefficients Summary per Penalization Technique for Dataset II

As observed, LASSO regularization retained 5 predictors out of the 8 available and effectively shrunk the coefficients of the variables age, capsular penetration and pgg45 to zero. Conversely, both Ridge Regression and Elastic Net regularization retained all the predictors in this specific iteration. Interpreting these coefficients is relatively straightforward, as the magnitude of each coefficient reflects its contribution to the final prediction. It's important to note that since some predictors and the response variable are in logarithmic scales, the interpretation of the coefficients takes into account the logarithmic transformation. For instance, in the context of Elastic Net, if we increase the logarithm of the cancer volume for a patient by 1 standard deviation, the predicted PSA (in a logarithmic scale) will be increased by 0.41. On the other hand, if a patient has seminal vesicle invasion, the predicted PSA (in a logarithmic scale) will be increased by 1.03, and so forth.

https://doi.org/10.26219/heal.aueb.6189

Overall, as shown in the Table 4.2, the predictors cancer volume, prostate weight, and seminal vesicle invasion have received the largest absolute value coefficients as indicated by the penalization techniques. This suggests that these predictors play a significant role in predicting the logarithmic value of PSA, with cancer volume and seminal vesicle invasion having the most substantial impact.

To gain further insights into the importance and contribution of each predictor, we conducted an analysis to count the number of iterations in which the coefficient of each predictor was non-zero. This analysis helps us understand how frequently each predictor was retained in the final model across multiple iterations. In the case of LASSO regularization, this process is straightforward, as this technique inherently promote sparsity, resulting in coefficients that are either non-zero or zero. For Ridge Regression, where coefficients tend to be nonzero due to its nature, we applied a threshold of an absolute value less than 0.005 to determine whether a predictor was effectively contributing to the model in each iteration. The results are summarized in the next Table.

Predictor	**Ridge Regression**	**LASSO**	**Elastic Net**
cancer volume	100	100	100
prostate weight	100	100	100
age	100	52	100
benign hyperplasia amount	99	85	100
seminal invasion	100	100	100
capsular penetration	95	49	100
Gleason	95	57	100
Gleason scores 4 or 5	97	73	0

Table 4.4: Occurrences of non-zero Coefficients over 100 Iterations for Dataset II

As presented in Table 4.3, our analysis reveals compelling insights into the importance of various predictors within each penalization approach. Notably, cancer volume, prostate weight, and seminal vesicle invasion emerge as the most influential variables, consistently retained in all 100 iterations across all three penalization techniques. LASSO, known for its ability to induce sparsity, produced more

https://doi.org/10.26219/heal.aueb.6189

parsimonious models by retaining fewer predictors than the other techniques. Intriguingly, Elastic Net consistently retained all predictors except from the predictor ”Percentage of Gleason scores equal to 4 or 5”.

Furthermore, our investigation extends to examining how many predictors, regardless of their identities, each penalization technique retained in each iteration. This approach provides an overarching perspective on the sparsity or inclusiveness of the models. In Ridge Regression a total of 87 iterations retained all 8 predictors while in the remaining 13 iterations, one predictor was omitted from the models. Conversely, for LASSO, all 8 predictors were included in the models for 45 iterations. In 31 iterations, 5 predictors were retained, while in 12 iterations, 4 predictors were selected. Apparently according to Table 4.4, Elastic Net utilized exactly 7 predictors in each iteration.

Figure 4.3: Frequency of Predictors Retained by Penalization Techniques Across 100 Iterations in Dataset II

4.4 Early Stopping

Regression Trees

We will commence the implementation of the machine learning techniques by employing the CART (Classification and Regression Trees) algorithm to construct regression trees. As we have done previously in the preceding chapter, we will utilize the `tidymodels` framework along with the `rpart` engine and `vip` package. In addition the tuning of the hyperparameters will once again be implemented using the `dials` package.

The cost complexity parameter and the range of possible tree depth values, from which optimal settings will be selected, are defined as follows:

cost complexity	tree depth
0.0000000001	1
0.0000000781	2
0.00000525	3
0.00475	4
0.1	5

Table 4.5: Parameter Grid for Optimizing Regression Trees Hyperparameters in Dataset II

After conducting 100 iterations for tuning the CART algorithm, the results revealed that the cost complexity parameter (`cp`) predominantly converged to the minimum value within its specified range, which is 1e-10, in 94 iterations. Conversely, only 6 iterations led to a `cp` value of 0.1. This outcome complements the findings related to tree depth, where 43 iterations resulted in a tree depth of 3, 34 iterations led to a tree depth of 4 and 18 iterations led to a tree depth of 2.

This combined information suggests that a very small cost complexity parameter is favored, in line with the relatively shallow tree depth observed, indicating that substantial pruning may not be necessary. In the Appendix can be found a visualization of a regression tree constructed in the last iteration with depth size 3.

https://doi.org/10.26219/heal.aueb.6189

After conducting 100 iterations with the regression tree algorithm, the obtained performance metrics revealed an MSE of 0.84 and an MAE of 0.75. These results, similar to those observed in the previous dataset, fall short of the desired levels of predictive accuracy, indicating suboptimal model performance. In the remainder of this section, the Elastic Net metrics will be displayed in the boxplots, denoted by the dot-dashed line, as the optimal values, while the baseline metrics derived from multiple linear regression will be represented by a dashed line.

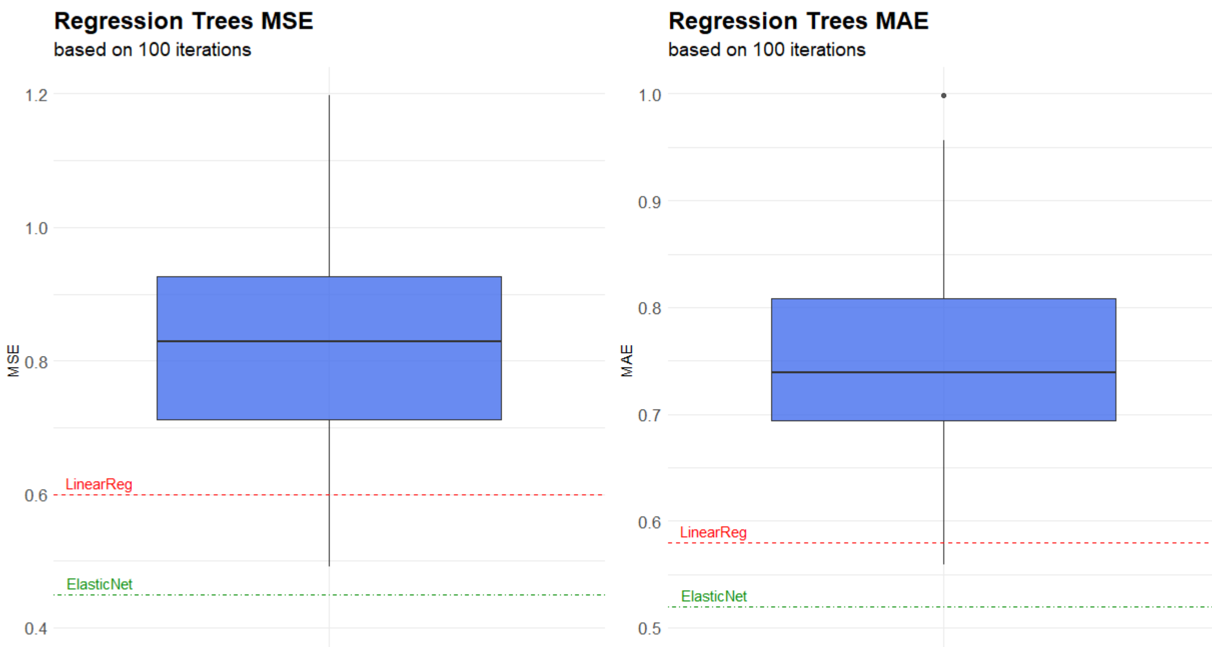

Figure 4.4: Regression Trees Metrics for Dataset II (MSE, MAE)

We will now proceed with the analysis of the predictors' contribution to the final prediction, as determined by the CART algorithm, and specifically, as calculated using the `rpart` engine. This contribution is represented by the sum of squared improvements of the accuracy metric, which in our case is the MSE. The aggregated results, obtained after conducting 100 iterations, are presented below.

https://doi.org/10.26219/heal.aueb.6189

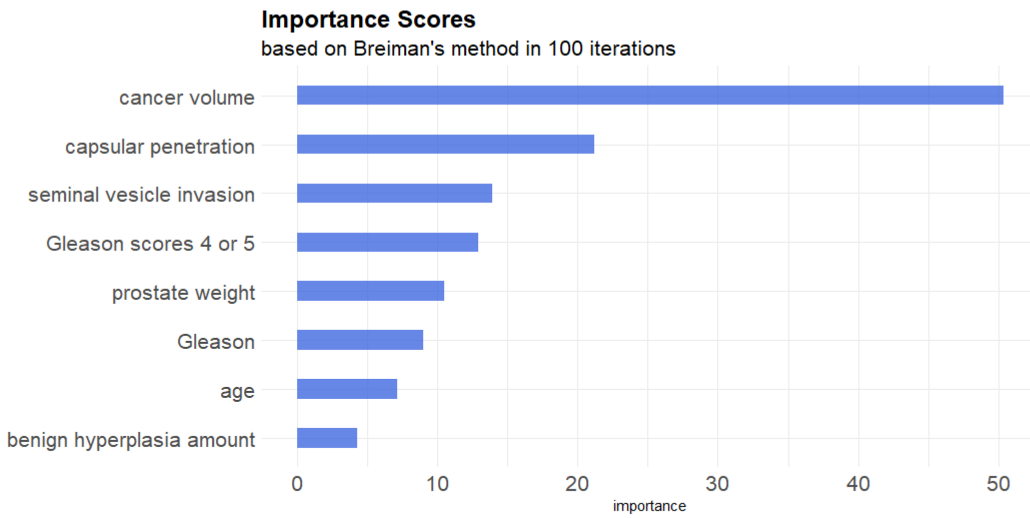

Figure 4.5: Relative Importance of Predictors in Regression Trees for Dataset II

The importance scores derived from the CART algorithm exhibit some differences compared to those obtained from the penalization approaches. While there is alignment in identifying two of the most important predictors so far, with both approaches highlighting the significance of cancer volume and seminal vesicle invasion, the second most important predictor identified by the regression tree algorithm is capsular penetration.

Gradient Boosted Regression Trees

We will now proceed with the implementation of the variant of the boosting algorithm, and in particular with the gradient boosted regression trees. Similar to the previous dataset, the boosting algorithm will be implemented using the `mlr3` and `gbm` packages in R. The hyperparameters to be tuned in this case are the learning rate (ν) and the total number of boosting rounds. The search space for tuning will cover a reasonable range of values and it is specified as [10,100] for the number of boosted trees and [0.01,0.3] for the learning rate. This tuning process will help us identify the optimal combination of hyperparameters for achieving the best predic-

 https://doi.org/10.26219/heal.aueb.6189

tive performance. Based on the results obtained from the tuning process, valuable insights into the hyperparameters were revealed. Among the 100 iterations, it was evident that in 50 of them, the optimal number of boosted trees tended to favor smaller values, specifically 10, 16, and 22. This observation suggests that the boosting algorithm does not necessarily require an extensive number of boosting rounds to minimize the Mean Squared Error (MSE).

Additionally, the learning rate exhibited a range of values spanning from 0.1 to 0.3, with a median value of 0.2. The results can be seen visually in the Appendix. After 100 iterations, the MSE was found to be 0.73, and the MAE was 0.67. These metrics were notably worse than the baseline measurements obtained from the multiple linear regression model, further highlighting the insufficiency of this machine learning technique in predicting the logarithm of the PSA for this specific dataset. The results are visually presented in Figure 4.6 for better comparison.

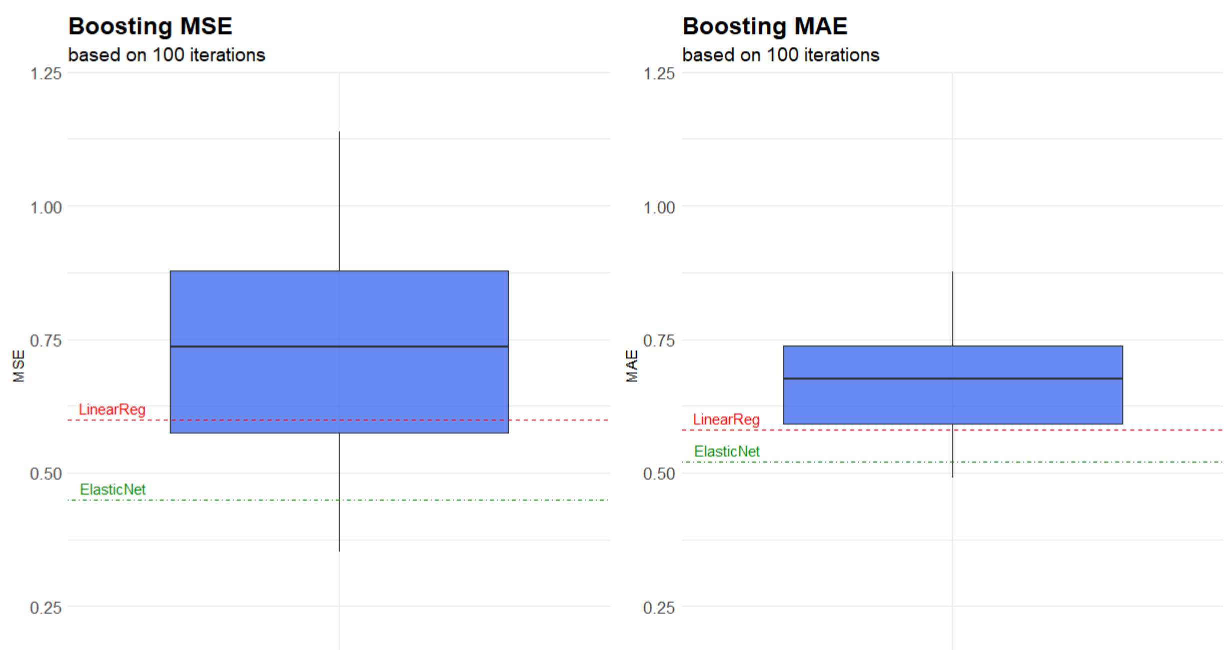

Figure 4.6: Gradient Boosted Regression Trees Metrics for Dataset II (MSE, MAE)

https://doi.org/10.26219/heal.aueb.6189

To conclude this Chapter on the application of regularization techniques, we sought to identify the most influential predictors derived from the boosted regression trees algorithm. The importance scores, based on Breiman's method, were computed in each of the 100 iterations, and the aggregated results are presented in the Figure 4.7. Once again, the results are scaled according to the gbm package.

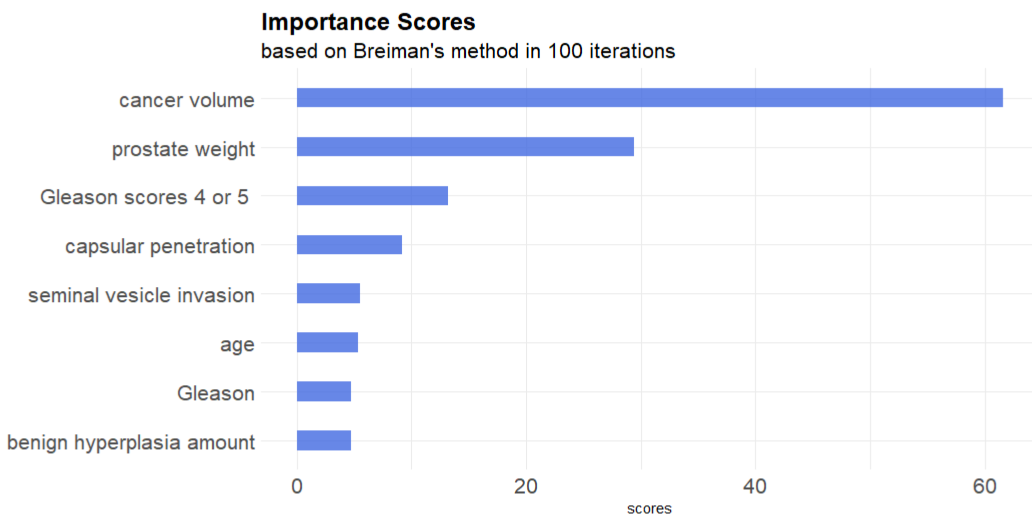

Figure 4.7: Relative Importance of Predictors in Gradient Boosted Regression Trees for Dataset II

Upon careful examination of the results, it is evident that the variant of the boosting algorithm highlighted the cancer volume as the most substantial predictor, reiterating its importance. Following closely behind in terms of significance were the predictors prostate weight and the percentage of Gleason scores 4 or 5. Conversely, the predictors age and benign hyperplasia amount consistently received the lowest importance scores.

https://doi.org/10.26219/heal.aueb.6189

4.5 Ensembling

In this subsection, as we delve into the Random Forest algorithm for this regression task, we will be employing the `randomForestSRC` and `MLmetrics` packages in R. The overall approach follows the same principles as outlined in the previous chapter, with minor adjustments to suit the regression task. Specifically, we will set the split rule to be based on MSE, given the nature of our task. We will also focus on tuning four key hyperparameters: the number of total trees, the number of predictors considered for each split, the minimum node size, and the maximum node depth. It's important to note that the selected ranges for these hyperparameter values are not arbitrary but rather sensible, taking into account the dimensionality and characteristics of our dataset. The specific ranges for the hyperparameters are detailed in the table below: After conducting 100 iterations of the random forest

hyperparameter	**range**
number of predictors in each split	[2,7]
total number of trees	[50,500]
minimum node size	[1,10]
node depth	[2,15]

Table 4.6: Parameter Grid for Optimizing Random Forest Hyperparameters for Dataset II

algorithm, we observed some interesting patterns in the tuning of hyperparameters. It appears that the algorithm consistently favored using 6 or 7 predictors at each split, indicating a preference for considering the majority of available predictors to make decisions. Regarding the optimal node size, we found that moderate values, such as 3, 6, and 7, were selected most frequently, suggesting that the algorithm often prefers to split nodes into subsets of these sizes. In contrast, the node depth exhibited significant variability, with various values from the specified range being chosen multiple times. Moreover, the median number of total trees grown was 181 trees. These insights provide valuable information about how the random forest algorithm adapts to our regression task and can be displayed in the Appendix.

https://doi.org/10.26219/heal.aueb.6189

Despite conducting thorough tuning and multiple iterations, the random forest algorithm's predictive accuracy for this regression task did not meet our expectations. The MSE reached 0.63, and the MAE 0.64, both of which are not satisfactory. In fact, these results were even worse than our baseline measurements obtained from the linear regression model. Those findings are summarized visually in Figure 4.6.

Figure 4.8: Random Forest Metrics for Dataset II (MSE, MAE)

We will conclude this subsection, presenting the importance scores derived from the random forest algorithm using the permutation method. In this approach, the OOB observations are permuted, and the resulting scores are calculated. Across the 100 iterations, we've stored and aggregated these results to obtain a comprehensive view of predictor importance. The outcome of this process is depicted in Figure 4.9.

https://doi.org/10.26219/heal.aueb.6189

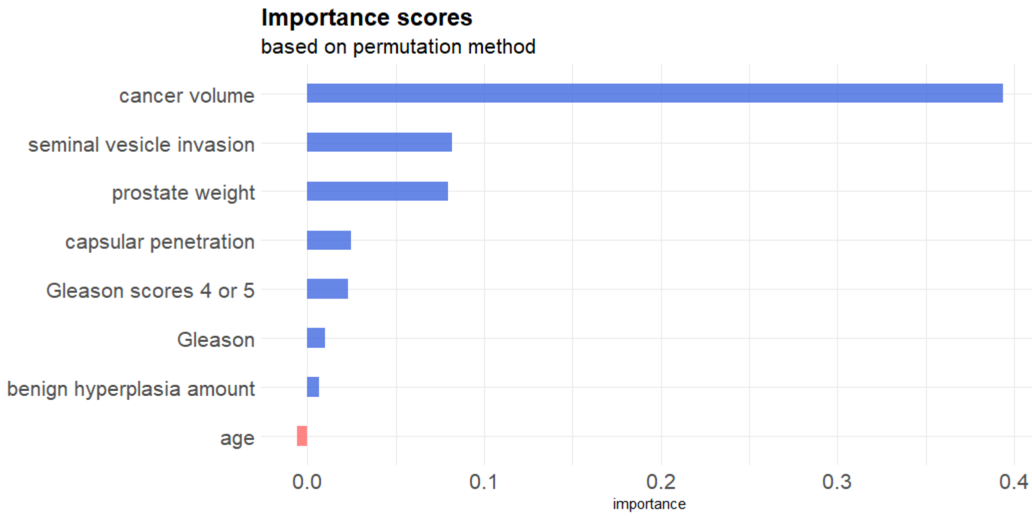

Figure 4.9: Relative Importance of Predictors in Random Forest for Dataset II

As we can observe, there is substantial agreement between the random forest algorithm and the previous regularization techniques. The predictors cancer volume, prostate weight and seminal vesicle invasion consistently emerge as the most influential predictors for prediction accuracy, with the cancer volume outstanding in each of the techniques. The similarity in the predictor scores across the different techniques reinforces the importance of these variables in predicting the (logarithm of) PSA. The agreement between methods provides additional confidence in the significance of these three predictors.

https://doi.org/10.26219/heal.aueb.6189

4.6 Results

Our primary objective in this research was to develop predictive models using regularization techniques for estimating the logarithm of PSA levels based on eight clinical predictors, despite the seemingly limited necessity for such techniques within our dataset comprising 97 observations and 8 predictors. The outcomes of our investigation offer valuable insights into the prospective utility of these methodologies.

The investigation into penalization approaches has yielded noteworthy results, surpassing the baseline measurements in predictive accuracy. Among the three penalization techniques, namely Ridge Regression, Lasso, and Elastic Net, all demonstrated substantial improvements. It is imperative to highlight that Elastic Net excelled in this context, exhibiting the most remarkable performance. This was evidenced by the achievement of the lowest MSE and MAE with values of 0.45 and 0.52, respectively.

On the other hand, the machine learning approaches, including Regression Trees, Random Forest, and Gradient Boosted Trees, did not perform as well, producing even worse measures than the baseline model. These results suggest that there may not be any complex, non-linear relationships between the clinical predictors and PSA levels in this dataset (plus the majority of the predictors are measured in a log scale) and the dataset's dimensions is restrictive. The results across the techniques are summarized in Table 4.6.

Regarding the identification of important predictors, all regularization techniques almost consistently highlighted three key variables: prostate weight, seminal vesicle invasion, and cancer volume. These predictors emerged as the most influential factors for predicting PSA levels. Therefore, we conclude that a strong relationship exists between these clinical variables and PSA. To achieve accurate predictions, it is crucial to consider the measurements of these specific predictors.

Regularization	Technique	MCE	MAE
None	Linear Regression	0.60	0.58
Statistical	Ridge Regression	0.56	0.57
	LASSO	0.57	0.58
	Elastic Net	0.45	0.52
Machine Learning	Regression Trees	0.82	0.74
	Random Forest	0.63	0.64
	Gradient Boosted Regression Trees	0.72	0.67

Table 4.7: Overall Metrics of Regularization Techniques for Dataset II

In light of our comprehensive exploration of regularization techniques through two distinct applications and a thorough review of existing literature, we can distill the key strengths and weaknesses associated with each of the regularization methodologies featured in this thesis. To succinctly convey this information, we have organized our findings into three columns representing the techniques Penalization, Early Stopping and Ensembling. Each row corresponds to a pivotal capability or characteristic inherent to these techniques, the evaluation of which is delineated as follows:

- Green color: Signifies a favorable capability or characteristic exhibited by the respective technique
- Red color: Indicates an unfavorable aspect or limitation associated with the technique
- Orange color: Denotes a neutral or mixed attribute without strong positive or negative implications.

To further elucidate this framework, we have employed distinctive symbols to denote the outcome for each technique and capability:

- A $\surd$ symbol signifies that the technique successfully satisfies the specified capability or characteristic

- A $\times$ symbol denotes that the technique does not meet the criteria for the capability or characteristic
- A $-$ symbol suggests that the technique neither fully satisfies nor fails to satisfy the given capability or characteristic.

This comprehensive evaluation framework allows for a concise representation of the nuanced strengths and weaknesses inherent in each regularization technique.

Characteristic	**Penalization**	**Early Stoping**	**Ensembling**
Mixed Type Data	$\surd$	$\surd$	$\surd$
Handling Collinearity	$-$	$\surd$	$\surd$
Standardization	$\surd$	$\times$	$\times$
Parametric Assumption	$\surd$	$\times$	$\times$
Flexibility	$\times$	$\surd$	$\surd$
Interpretability	$\surd$	$-$	$-$
Predictive Power	$\surd$	$\times$	$\times$
Customizable Hyperparameters	$\surd$	$\surd$	$\surd$
Capture Complex Relationships	$\times$	$\surd$	$\surd$
Handling Missing Values	$\times$	$\surd$	$\surd$
Computational Burden	$\surd$	$-$	$\times$
Multi Model Inference	$\times$	$\surd$	$\surd$

Table 4.8: Comparison of Regularization Techniques

In accordance with the information presented in Table 4.7, for example the ”Standardization” represent a distinctive characteristic or procedural requirement that various techniques may either necessitate or not. In the context of penalization, denoted by a $\surd$, it indicates an obligatory procedure that may potentially consume considerable time and, consequently, is depicted by the color red. Conversely, machine learning techniques do not mandate a standardization preprocessing step, symbolized by a $\times$ and are represented in the color green. This interpretation extends to the entirety of the table's contents.

https://doi.org/10.26219/heal.aueb.6189

Chapter 5

Conclusion & Future Work

This dissertation provides a comprehensive overview of the regularization techniques applied in clinical biostatistics. The examined techniques belong to the broader spectrum of statistical and machine learning algorithms, and their relationships with the regularization approaches, namely Penalization, Early Stopping and Ensembling, are extensively discussed. The primary objective was to facilitate the application of regularization in clinical biostatistics, thereby enhancing its implementation for medical purposes.

The techniques were implemented using the R software on two available datasets, both related to prostate cancer patients. In the first dataset, our aim was to predict whether a patient's tumor is benign or malignant based on eight different predictor variables. Following the implementation of all regularization approaches for this classification task, Ridge Regression (which belongs to Penalization) emerged as the top-performing technique, achieving the most favorable predictive measures, including Mean Classification Error (MCE) and Area Under the Curve (AUC).

In the second dataset, we examined eight clinical variables and their potential relationship with the logarithm of prostate-specific antigen (PSA), a well-known biomarker. Once again, the Penalization approach outperformed all other techniques, with Elastic Net producing the smallest Mean Squared Error (MSE) and

https://doi.org/10.26219/heal.aueb.6189

Mean Absoluter Error (MAE). Interestingly, machine learning approaches yielded less satisfactory results in this particular dataset.

It's worth noting that both datasets, at first glance, may not be considered highly dimensional since they consist of only eight predictors and 100 and 97 observations respectively, hence reasonably can someone assume that regularization may constitute an unnecessary procedure. Future research could explore the application of regularization in high-dimensional datasets or in the field of bioinformatics, where a multitude of genes and strategic regularization could potentially yield significant findings. Additionally, machine learning algorithms might be more advantageous in non-parametric scenarios and when the underlying relationship between predictors and the dependent variable is exceedingly complex.

In conclusion, this work highlights the significance of regularization techniques in clinical biostatistics, as demonstrated by its ability to enhance predictive results concerning prostate cancer. The study underscores the effectiveness of the statistical regularization in scenarios where the inherent relationship within the dataset lacks complexity and the potential for further exploration in this domain, especially in high-dimensional datasets and non-parametric contexts. Overall, the application of regularization techniques opens up a realm of possibilities for future research and advancement in clinical biostatistics.

https://doi.org/10.26219/heal.aueb.6189

References

Breiman, L. (1996). Bagging predictors.*Machine Learning*, 24, 123-140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5-32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees. *Monterey, CA: Wadsworth.*

Fahrmeir, L., Kneib, T., & Lang, S. (2004). Penalized structured additive regression for space-time data: a Bayesian perspective. *Statistica Sinica*, 14, 731-761.

Freund, Y., & Shapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119-139.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29, 1189-1232.

Friedman, J. H. (1999). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38, 367-378.

Friedman, J. H., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion). *The annals of statistics*, 28, 337-407.

Greenwell, B., Boehmke, B., Cunningham, J, & Gbm Developers. (2020). gbm: Generalized Boosted Regression Models. https://CRAN.R-project.org/package=gbm. R package version 2.1.8.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2001). The elements of statistical learning. *Springer* series in statistics New York.

Hastie, T., Tibshirani, R., & Wainwright, M. (2015). Statistical learning with sparsity - the lasso and generalizations. *CRC prss*.

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: biased estimation for

 https://doi.org/10.26219/heal.aueb.6189

nonorthogonal problems. *Technometrics*, 12, 55-67.

Lang, M., Binder, M., et al. mlr3: A modern object-oriented machine learning framework in R.*Journal of Open Source Software*, 4.44, 1903

Mayr, A., Binder, H., Gefeller, O., & Schmid, M. (2014). The evolution of boosting algorithms - from machine learning to statistical modelling. *Methods of information in medicine*, 53, 419-427.

Park, T., & Casella, G. (2008). The Bayesian lasso. *Journal of the American Statistical Association*, 103, 681-686.

Ridgeway, G. (2007). Generalized Boosted Models: A guide to the gbm package. *Update*, 1.

Saifi, S. (2021). Prostate cancer dataset. https://www.kaggle.com/sajidsaifi/prostate-cancer, 2021.

Stamey, T. A., Kabalin, J. N., & McNeal, J. E. (1989). Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate radical prostatectomy treated patients. *J Urol*, 141, 1076-1083.

Strobl, C., Boulesteix, A. L., & Zeilis A. (2007). Bias in random forest variable importance measures: illustrations, sources and a solution. *BMC Bioinformatics*, 8, 1-25.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58, 267-288.

Tutz, G., & Binder, H. (2006). Generalized additive modeling with implicit variable selection by likelihood-based boosting. *Biometrics*, 62, 961-971.

Ishwaran, H., & Kogalur, U. B. (2023). Fast unified random forests. https://CRAN.R-project.org/package=randomForestSRC.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 67, 301-320.

https://doi.org/10.26219/heal.aueb.6189

Appendix A

Figures

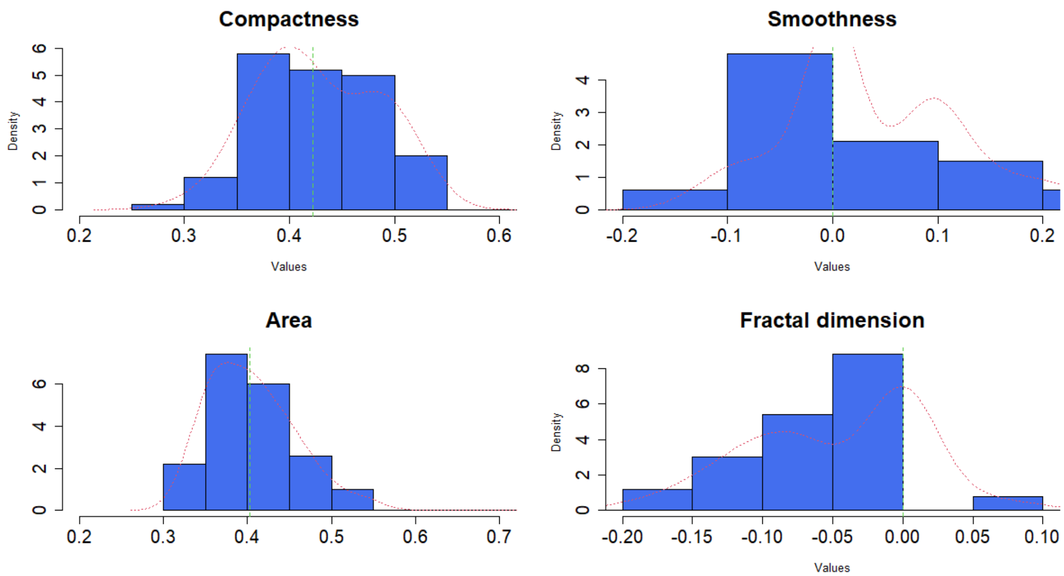

Figure A.1: Variability of Penalized Coefficients: Ridge Regression Results for Four Key Predictors Across 100 Iterations for Dataset I

https://doi.org/10.26219/heal.aueb.6189

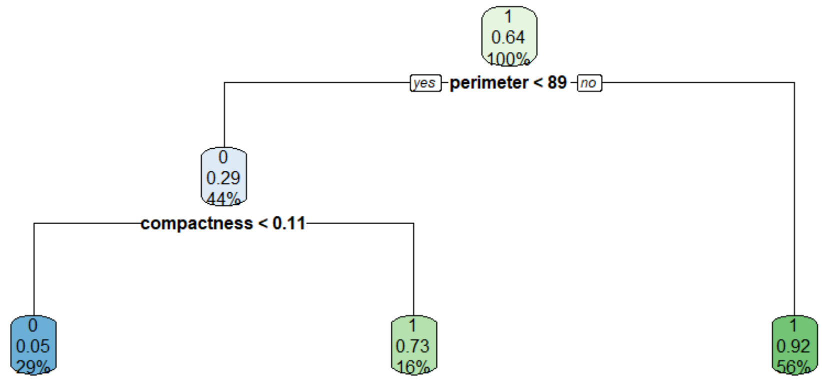

Figure A.2: Indicative Classification Tree With Depth 2 for Dataset I

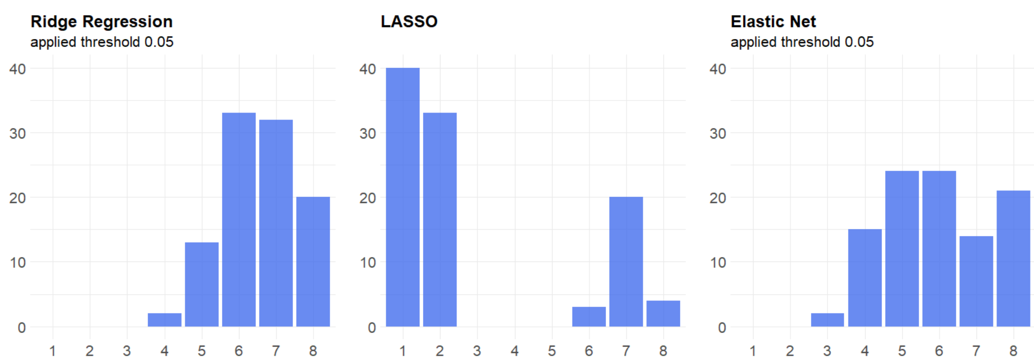

Figure A.3: Frequency of Predictors Retained by Penalization Techniques Across 100 Iterations In Dataset I

https://doi.org/10.26219/heal.aueb.6189

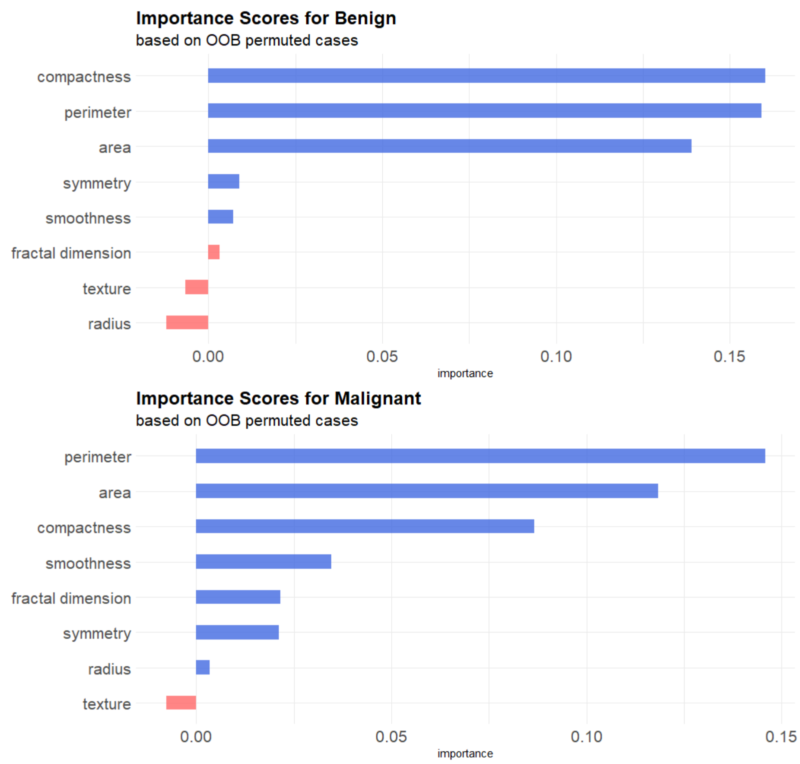

Figure A.4: Random Forest Conditional Features Relevance for Dataset I

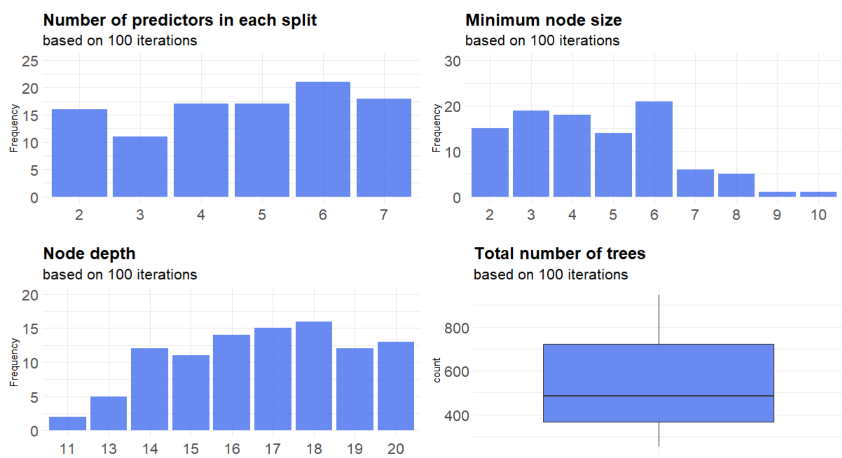

Figure A.5: Random Forest Hyperparameters Tuning Results Across 100 Iterations for Dataset I

https://doi.org/10.26219/heal.aueb.6189

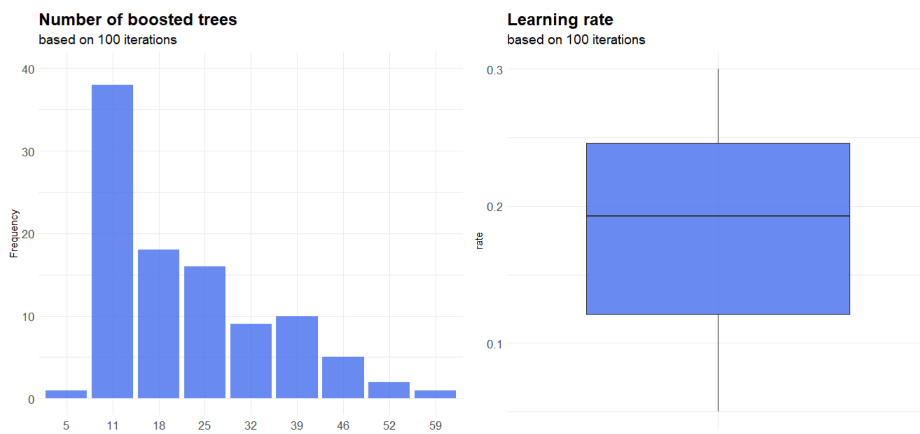

Figure A.6: Gradient Boosted Regression Trees Hyperparameters Tuning Results Across 100 Iterations for Dataset I

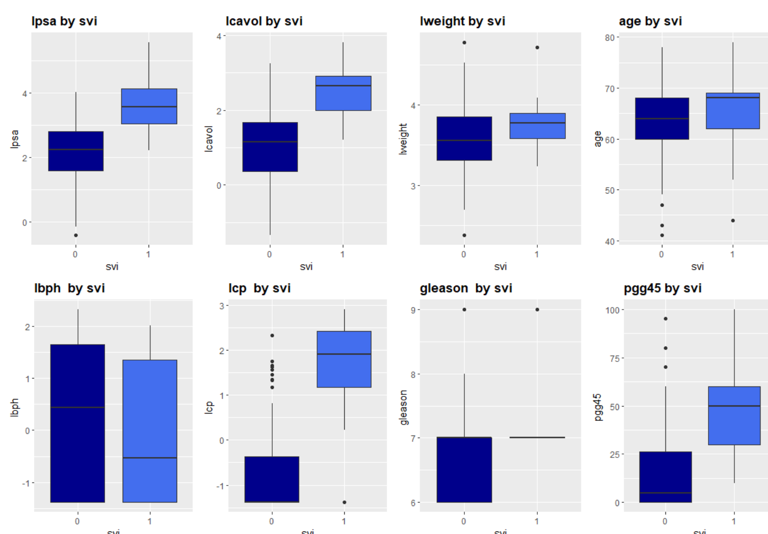

Figure A.7: Exploring Relationships: SVI Factor vs. Numerical Variables via Boxplot Analysis

https://doi.org/10.26219/heal.aueb.6189

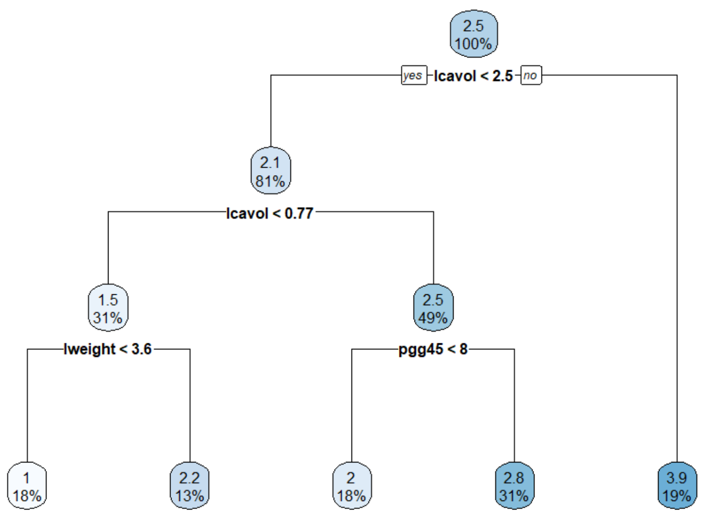

Figure A.8: Indicative Regression Tree With Depth 3 for Dataset II

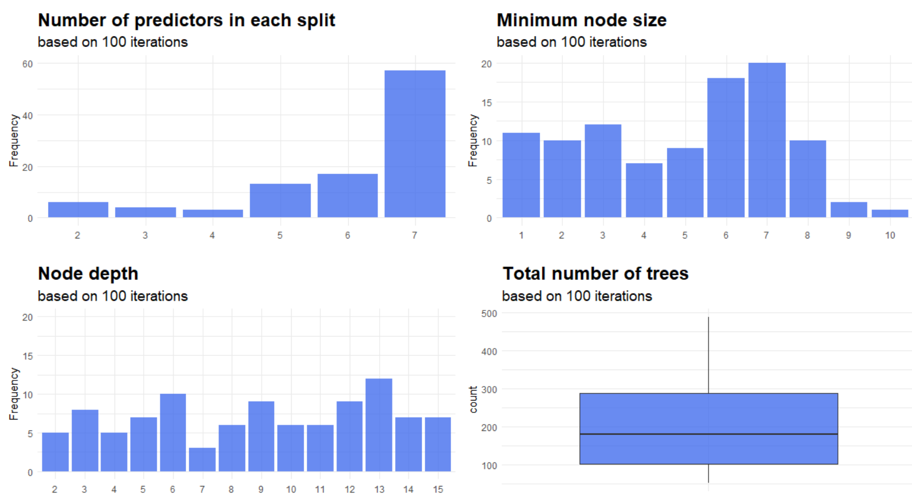

Figure A.9: Random Forest Hyperparameters Tuning Results Across 100 Iterations for Dataset II

https://doi.org/10.26219/heal.aueb.6189

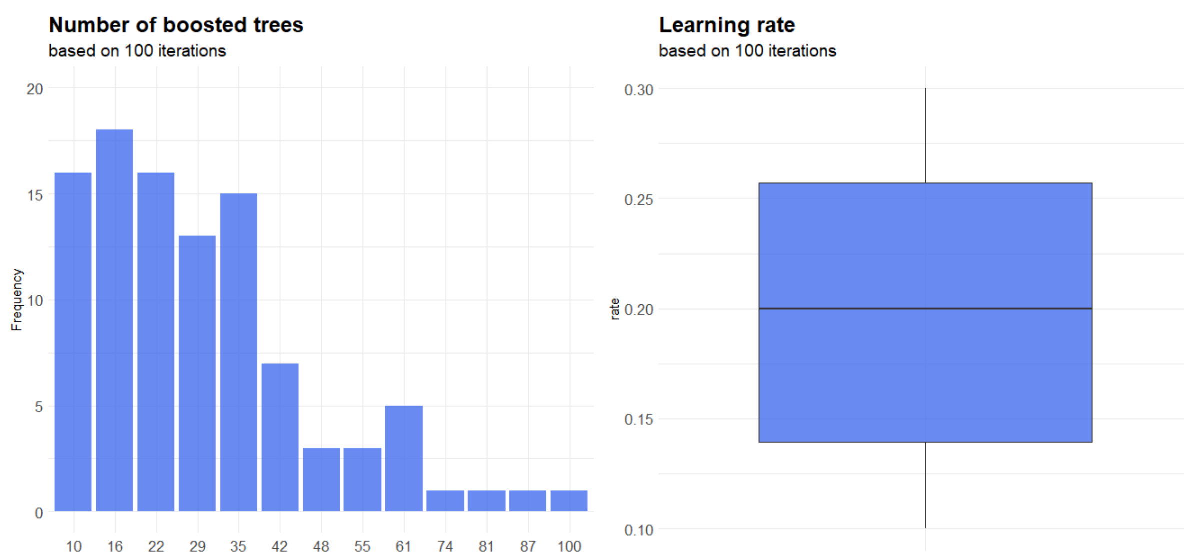

Figure A.10: Gradient Boosted Classification Trees Hyperparameters Tuning Results Across 100 Iterations for Dataset II

https://doi.org/10.26219/heal.aueb.6189

Appendix B

Indicative R Script for Prostate Cancer Dataset II

```
# Exploratory Data Analysis  ---

# load the data
df <- read.delim("C:/Users/Panos/Desktop/Master_Thesis/data
   /Hastie_Prostate_Cancer.txt")
df <- df[,-10]
head(df)
str(df)
# descriptive statistics
for (col in 1:ncol(df)) {
  df[, col] <- as.numeric(df[, col])}
str(df)
num_only <- df[,-5]
num_only
head(num_only)
summary(num_only)
round(apply(num_only,2,sd),2)
```

https://doi.org/10.26219/heal.aueb.6189

```
table(df$svi)
# binary boxplots
custom_colors <- c("0" = "blue4", "1" = "royalblue2")
b1 <- ggplot(df, aes(x = as.factor(svi), y = lpsa, fill =
   as.factor(svi)),alpha=0.8) +
  geom_boxplot() +
  labs(title = "lpsa by svi", x = "svi",  y = "lpsa") +
  scale_fill_manual(values = custom_colors) +
  theme(legend.position = "none") +  theme(plot.title =
     element_text(face = "bold"))
b2 <- ggplot(df, aes(x = as.factor(svi), y = lcavol, fill =
    as.factor(svi)),alpha=0.8) + geom_boxplot() +labs(title
    = "lcavol by svi",  x = "svi",  y = "lcavol") +
  scale_fill_manual(values = custom_colors) + theme(legend.
     position = "none") +
  theme(plot.title = element_text(face = "bold"))
b3 <- ggplot(df, aes(x = as.factor(svi), y = lweight, fill
   = as.factor(svi)),alpha=0.8) + geom_boxplot() +labs(
   title = "lweight by svi",
x = "svi",   y = "lweight") + scale_fill_manual(values =
   custom_colors) + theme(legend.position = "none") + theme
   (plot.title = element_text(face = "bold"))
b4 <- ggplot(df, aes(x = as.factor(svi), y = age, fill = as
   .factor(svi)),alpha=0.8) + geom_boxplot() + labs(title =
    "age by svi",  x = "svi", y = "age") + scale_fill_
   manual(values = custom_colors) + theme(legend.position =
    "none") + theme(plot.title = element_text(face = "bold"
   ))
b5 <- ggplot(df, aes(x = as.factor(svi), y = lbph , fill =
   as.factor(svi)),alpha=0.8) +
  geom_boxplot() + labs(title = "lbph  by svi",  x = "svi",
      y = "lbph ") +
```

https://doi.org/10.26219/heal.aueb.6189

```
  scale_fill_manual(values = custom_colors) + theme(legend.
     position = "none") +
  theme(plot.title = element_text(face = "bold"))
b6 <- ggplot(df, aes(x = as.factor(svi), y = lcp , fill =
   as.factor(svi)),alpha=0.8) +
  geom_boxplot() + labs(title = "lcp  by svi",  x = "svi",
     y = "lcp ") +
  scale_fill_manual(values = custom_colors) + theme(legend.
     position = "none") +
  theme(plot.title = element_text(face = "bold"))
b7 <- ggplot(df, aes(x = as.factor(svi), y = gleason , fill
    = as.factor(svi)),alpha=0.8) + geom_boxplot() + labs(
   title = "gleason  by svi",
x = "svi", y = "gleason ") + scale_fill_manual(values =
   custom_colors) +
theme(legend.position = "none") + theme(plot.title =
   element_text(face = "bold"))
b8 <- ggplot(df, aes(x = as.factor(svi), y = pgg45, fill =
   as.factor(svi)),alpha=0.8) +
geom_boxplot() + labs(title = "pgg45 by svi", x = "svi", y
   = "pgg45 ") +
  scale_fill_manual(values = custom_colors) + theme(legend.
     position = "none") +
  theme(plot.title = element_text(face = "bold"))
bars <- ggarrange(b1,b2,b3,b4,b5,b6,b7,b8,ncol=4,nrow=2)
# distribution of all predictors plus the response variable
    for the normality assumptions
p <- ncol(num_only)
y <- num_only
par(mfrow=c(2,4))
for (i in 1:p){
  hist(y[,i], main=names(y)[i], probability=TRUE,xlab = "")
```

```
    lines(density(y[,i]), col=2)
    index <- seq( min(y[,i]), max(y[,i]),
                  length.out=100)
    ynorm <- dnorm( index, mean=mean(y[,i]),
                    sd(y[,i]) )
    lines( index, ynorm, col=3, lty=3, lwd=3 ) }
# correlations and correlation matrix
library(corrplot)
library(sjPlot)
sjp.corr(num_only,corr.method = "pearson")
corrplot(cor(num_only),method = "ellipse")
corrplot(cor(num_only),method = "number",type = "lower")
tab_corr(num_only,corr.method = "pearson",show.p = FALSE,p.
    numeric = TRUE,fade.ns = TRUE,digits = 3,triangle = "
    lower")
tab_corr(num_only,corr.method = "spearman",show.p = FALSE,p
    .numeric = TRUE,fade.ns = TRUE,digits = 3,triangle = "
    lower")

# Penalization Approaches ---
# split train and test
set.seed(1999)
train <- sample(1:length(df[,1]),0.70*length(df[,1]))
test <- setdiff(1:length(df[,1]),train)

#  Linear Regression using mlr3 package ---
# algorithm
iter <- 100
task = TaskRegr$new(id="LinReg",backend = df,target = "lpsa
    ")
learner = lrn("regr.lm")
set.seed(199)
```

https://doi.org/10.26219/heal.aueb.6189

```
lr_list <- list()
resample <- rsmp("holdout",ratio=7/10)
for ( i in 1 : iter) {
  lr_list[[i]] <- resample(task,learner,resample,store_
     models=TRUE)}
linear_reg <- data.frame("MSE" = sapply(1:iter, function(x)
  lr_list[[x]]$aggregate(msr("regr.mse"))), "MAE" = sapply
     (1:iter, function(x)
    lr_list[[x]]$aggregate(msr("regr.mae"))))
# linear metrics
linear_metrics <- apply(linear_reg,2,mean)
# standardization
df$lcavol <- (df$lcavol - mean(df$lcavol))/sd(df$lcavol)
df$lweight <- (df$lweight - mean(df$lweight))/sd(df$lweight
   )
df$age <- (df$age - mean(df$age))/sd(df$age)
df$lbph <- (df$lbph - mean(df$lbph))/sd(df$lbph)
df$lcp <- (df$lcp - mean(df$lcp))/sd(df$lcp)
df$gleason <- (df$gleason - mean(df$gleason))/sd(df$gleason
   )
df$pgg45 <- (df$pgg45  - mean(df$pgg45 ))/sd(df$pgg45)
df$lpsa <- scale(df$lpsa,center=TRUE,scale=FALSE)
# Ridge Regression using mlr3 package --
measure <- msr("regr.mse")
terminator <- trm("none")
param_set = ps(
  lambda = p_dbl(log(1), log(10), trafo = function(x) exp(x
     )-0.999))
iter <- 100
ridge_mse <- rep(0,iter)
ridge_mae <- rep(0,iter)
lambda_ridge <- rep(0,iter)
```

https://doi.org/10.26219/heal.aueb.6189

```
model_coefs_ridge <- list()
set.seed(1999)
start_time <- Sys.time()
for (i in 1:iter) {
  train <- sample(1:length(df[,1]),0.70*length(df[,1]))
  test <- setdiff(1:length(df[,1]),train)
  task_tune <- TaskRegr$new(id=paste("tune",i),backend = df
     [train,],
                              target = "lpsa")
  resample_tune <- rsmp("cv",folds=10)
  learner_tune <- lrn("regr.glmnet",alpha=0)
  tuner <- tnr("grid_search",resolution=30)
  instance_tune <- TuningInstanceSingleCrit$new(
    task = task_tune,learner = learner_tune,
    resampling = resample_tune,measure = measure,
    terminator = terminator,search_space = param_set,store_
       models = TRUE)
  tuner$optimize(instance_tune)
  learner_tune = lrn("regr.glmnet",alpha=0)
  lambda <- as.numeric(instance_tune$result_x_domain)
  lambda_ridge[i] <- lambda
  learner_tune$param_set$values$lambda <- lambda
  learner_tune$train(task_tune)
  model_coefs_ridge[[i]] <- as.matrix(learner_tune$model$
     beta)
  predictions <- learner_tune$predict_newdata(df[test,])
  ridge_mse[i] <- predictions$score(msr("regr.mse"))
  ridge_mae[i] <- predictions$score(msr("regr.mae"))}
Sys.time()-start_time
ridge_reg <- data.frame("MSE"=ridge_mse,"MAE"=ridge_mae,"
   lambda"=lambda_ridge)
ridge_metrics <- apply(ridge_reg, 2, mean)
```

https://doi.org/10.26219/heal.aueb.6189

```
# pie Chart for lambdas values
hist(ridge_reg$lambda,col="royalblue2",main="Lambda Tuned",
   xlab="")
# boxplot for MSE in Ridge Regression ---
boxplot_ridge_mse <- ggplot(data = ridge_reg, aes(x = "", y
    = MSE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = mean(linear_reg$MSE), linetype =
     "dashed", color = "red") +
  labs(title = "Ridge Regression MSE",x="",subtitle = "
     based on 100 iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.2,0.8))+
  theme_minimal() + theme(plot.title = element_text(face =
     "bold",size = 18),
                           plot.subtitle = element_text(size
                              =12),
                           axis.text.y = element_text(size =
                               16))
# boxplot for MAE in Ridge Regression ---
boxplot_ridge_mae  <- ggplot(data = ridge_reg, aes(x="",y =
    ,MAE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = mean(linear_reg$MAE), linetype =
     "dashed", color = "red") +
  labs(title = "Ridge Regression MAE",x="",subtitle = "
     based on 100 iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.3,0.8))+
  theme_minimal()  +  theme(plot.title = element_text(face
     = "bold",size = 18),
```

https://doi.org/10.26219/heal.aueb.6189

```
                         plot.subtitle = element_text(size
                            =12),
                         axis.text.y = element_text(size =
                            16))
# Interpretation
ridge <- model_coefs_ridge[[82]]
# Number of specific coefficients to be zero out of the
   total 100 iterations #
models_ridge <- as.data.frame(model_coefs_ridge)
for (col in names(models_ridge)) {
  models_ridge[[col]][abs(models_ridge[[col]]) < 0.005] <-
     0}
number_non_zero_coefs_ridge <- apply(models_ridge,1,
   function(x) sum(x!=0))
number_non_zero_coefs_ridge <- as.data.frame(number_non_
   zero_coefs_ridge)
# histogram for number of predictors kept
number_coefs_kept_ridge <- apply(models_ridge,2,function(x)
    sum(x!=0))
df_r <- data.frame(number_coefs_kept_ridge)
colnames(df_r) <- "NumberKept"
rownames(df_r) <- NULL
r_values_vars <- table(df_r$NumberKept)
r_values_vars <- data.frame(
  Predictor = c(1,2,3,4,5,6,7,8),
  Frequency = c(0,0,0,0,0,0,13,87))
plot_vars_kept_ridge<- ggplot(r_values_vars, aes(x=as.
   factor(Predictor),y=Frequency)) +
  geom_bar(stat="identity",fill = "royalblue2",alpha=0.8) +
  labs( title = "Ridge Regression",
    subtitle = "based on 100 iterations - applied threshold
        0.005", x="", y = "") +
```

 https://doi.org/10.26219/heal.aueb.6189

```
  ylim(c(0,90)) +  theme_minimal() +  theme(
    plot.title = element_text(face = "bold", size = 18),
    plot.subtitle = element_text(size = 16),
    axis.text.y = element_text(size = 16),
    axis.text.x = element_text(size = 16)) +
  guides(fill="none")
# histogram for predictors' variability
smooth_data_hist <- models_ridge[rownames(models_ridge)=="
   smoothness",]
smooth_data_hist <- as.data.frame(smooth_data_hist)
colnames(smooth_data_hist) <- NULL
smooth_data_hist <- as.list(smooth_data_hist)
smooth_data_ridge <- unlist(smooth_data_hist)
hist(smooth_data_ridge,
     col = "gray78", border = "black", breaks=50,
     xlab = "Values", ylab = "Density",
     main = "Smoothness",xlim=c(-0.2,0.2),
     probability =   T)
abline(v=median(smooth_data_ridge),col=3,lty=2)
lines(density(smooth_data_ridge), col=2,lty=3)
# LASSO using mlr3 package --
#algorithm
iter <- 100
param_set = ps(
  lambda = p_dbl(log(1), log(10), trafo = function(x) exp(x
     )-0.999))
lasso_mse <- rep(0,iter)
lasso_mae <- rep(0,iter)
lambda_lasso <- rep(0,iter)
model_coefs_lasso <- list()
set.seed(1999)
start_time <- Sys.time()
```

https://doi.org/10.26219/heal.aueb.6189

```
for (i in 1:iter) {
  train <- sample(1:length(df[,1]),0.70*length(df[,1]))
  test <- setdiff(1:length(df[,1]),train)
  task_tune <- TaskRegr$new(id=paste("tune",i),backend = df
     [train,],
                            target = "lpsa")
  resample_tune <- rsmp("cv",folds=10)
  learner_tune <- lrn("regr.glmnet",alpha=1)
  tuner <- tnr("grid_search",resolution=30)
  instance_tune <- TuningInstanceSingleCrit$new(
    task = task_tune,learner = learner_tune,
    resampling = resample_tune,measure = measure,
    terminator = terminator,search_space = param_set,store_
       models = TRUE)
  tuner$optimize(instance_tune)
  learner_tune = lrn("regr.glmnet",alpha=1)
  lambda <- as.numeric(instance_tune$result_x_domain)
  lambda_lasso[i] <- lambda
  learner_tune$param_set$values$lambda <- lambda
  learner_tune$train(task_tune)
  model_coefs_lasso[[i]] <- as.matrix(learner_tune$model$
     beta)
  predictions <- learner_tune$predict_newdata(df[test,])
  lasso_mse[i] <- predictions$score(msr("regr.mse"))
  lasso_mae[i] <- predictions$score(msr("regr.mae"))}
Sys.time() - start_time
lasso_reg <- data.frame("MSE"=lasso_mse,"MAE"=lasso_mae,"
   lambda"=lambda_lasso)
lasso_metrics <- apply(lasso_reg, 2, mean)
# boxplot for MSE in LASSO ---
boxplot_lasso_mse <- ggplot(data = lasso_reg, aes(x = "", y
     = MSE)) +
```

https://doi.org/10.26219/heal.aueb.6189

```
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = mean(linear_reg$MSE), linetype =
     "dashed", color = "red") +
  labs(title = "LASSO MSE",x="",subtitle = "based on 100
     iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.2,0.8))+
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                         plot.subtitle = element_text(
                            size=12),
                         axis.text.y = element_text(size
                            = 16))
# boxplot for MAE in LASSO
boxplot_lasso_mae  <- ggplot(data = lasso_reg, aes(x="",y =
    ,MAE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = mean(linear_reg$MAE), linetype =
     "dashed", color = "red") +
  labs(title = "LASSO MAE",x="",subtitle = "based on 100
     iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.3,0.8))+
  theme_minimal() + theme(plot.title = element_text(face =
     "bold",size = 18),
                         plot.subtitle = element_text(size
                           =12),
                         axis.text.y = element_text(size =
                            16))
# Interpretation
lasso1 <- model_coefs_lasso[[82]]
round(lasso1,3)
```

https://doi.org/10.26219/heal.aueb.6189

```
# Number of specific coefficients to NOT be zero out of the
    total 100 iterations
models_lasso <- as.data.frame(model_coefs_lasso)
no_non_zero_lasso <- apply(models_lasso,1,function(x) sum(x
   !=0))
no_non_zero_lasso <- as.data.frame(no_non_zero_lasso)
# histogram for number of predictors kept in each iteration
number_coefs_kept_lasso <- apply(models_lasso,2,function(x)
    sum(x!=0))
dataa_lasso <- data.frame(number_coefs_kept_lasso)
colnames(dataa_lasso) <- "NumberKept"
rownames(dataa_lasso) <- NULL
lasso_values_vars <- table(dataa_lasso$NumberKept)
lasso_values_vars <- data.frame(
  Predictor = c(1,2,3,4,5,6,7,8),
  Frequency = c(0,0,7,12,31,3,2,45))
plot_vars_kept_lasso <- ggplot(lasso_values_vars, aes(x=as.
   factor(Predictor),y=Frequency)) +
  geom_bar(stat="identity",fill = "royalblue2",alpha=0.8) +
  labs(title = "LASSO", subtitle = "based on 100 iterations
     ", x="", y = "Frequency") +
  ylim(c(0,90)) + theme_minimal() +  theme(plot.title =
     element_text(face = "bold", size = 18),plot.subtitle =
      element_text(size = 16),
  axis.text.y = element_text(size = 16), axis.text.x =
     element_text(size = 16)) +
  guides(fill="none")
vars_kept <- ggpubr::ggarrange(plot_vars_kept_ridge,plot_
   vars_kept_lasso,ncol=2)
# Elastic Net using mlr3 package --
#algorithm
param_set = ps(
```

https://doi.org/10.26219/heal.aueb.6189

```
  lambda = p_dbl(log(1), log(10), trafo = function(x) exp(x
     )-0.999),
  alpha = p_dbl(0, 1))
iter <- 100
elnet_mse <- rep(0, iter)
elnet_mae <- rep(0, iter)
lambda_elnet <- rep(0, iter)
alpha_elnet <- rep(0,iter)
models_elastic <- list()
set.seed(1999)
start_time <- Sys.time()
for(i in 1:iter){
  train_set = sample(1:length(df[,1]), 0.7 * length(df[,1])
     )
  test_set = setdiff(1:length(df[,1]), train_set)
  task_tune <- TaskRegr$new(id=paste("tune",i),backend = df
     [train,],
                            target = "lpsa")
  resample_tune = rsmp("cv", folds = 10)
  learner_tune = lrn("regr.glmnet")
  tuner <- tnr("grid_search", resolution = 10)
  instance_tune <- TuningInstanceSingleCrit$new(
    task = task_tune,learner = learner_tune,
    resampling = resample_tune,measure = measure,
    terminator = terminator,search_space = param_set,store_
       models = TRUE)
  tuner$optimize(instance_tune)
  learner_tune = lrn("regr.glmnet")
  learner_tune$param_set$values$lambda <- as.numeric(
     instance_tune$result_x_domain$lambda)
  learner_tune$param_set$values$alpha <- as.numeric(
     instance_tune$result$alpha)
```

https://doi.org/10.26219/heal.aueb.6189

```
    lambda_elnet[i] <- as.numeric(instance_tune$result_x_
       domain$lambda)
    alpha_elnet[i] <- as.numeric(instance_tune$result$alpha)
    learner_tune$train(task_tune)
    models_elastic[[i]] <- as.matrix(learner_tune$model$beta)
    prediction = learner_tune$predict_newdata(df[test_set,])
    elnet_mse[i] <- prediction$score(msr("regr.mse"))
    elnet_mae[i] <- prediction$score(msr("regr.mae"))}
Sys.time()-start_time
elnet_reg <- data.frame("MSE" = elnet_mse, "MAE" = elnet_
   mae,
                        "Lambda" = lambda_elnet, "Alpha" =
                           alpha_elnet)
elnet_metrics <- apply(elnet_reg, 2, mean)
# hyperparameters tuned results ---
table(elnet_reg$Lambda)
table(elnet_reg$Alpha)
lambda_counts_elnet <- c("0.05" = 81, "0.10" = 19)
total_elnet<- sum(lambda_counts_elnet)
lambda_percent_elnet <- lambda_counts_elnet / total_elnet *
    100
labels <- names(lambda_counts_elnet)
counts <- as.integer(table(elnet_reg$Lambda))
percent <- lambda_percent_elnet
colors <- c("lightsalmon","oldlace" )
par(mfrow=c(1,2))
pie(counts, labels = counts, col = colors, main = "Elastic
   Net Lambda Tuned")
legend("topright", labels, cex = 0.8, fill = colors)
hist(elnet_reg$Alpha,main="Elastic Net Alpha Tuned",breaks
   =10,col="royalblue2",border="black",xlab="")
dev.off()
```

https://doi.org/10.26219/heal.aueb.6189

```
# boxplot for MSE in Elastic Net
boxplot_elnet_mse <- ggplot(data = elnet_reg, aes(x = "", y
    = MSE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = mean(linear_reg$MSE), linetype =
     "dashed", color = "red") +
  labs(title = "Elastic Net MSE",x="",subtitle = "based on
     100 iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.2,0.8)) +
  theme_minimal() + theme(plot.title = element_text(face =
     "bold",size = 18),
                        plot.subtitle = element_text(size
                           =12),
                        axis.text.y = element_text(size =
                           16))
# boxplot for MAE in Elastic Net
boxplot_elnet_mae  <- ggplot(data=elnet_reg, aes(x="",y = ,
    MAE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = mean(linear_reg$MAE), linetype =
     "dashed", color = "red") +
  labs(title = "Elastic Net MAE",x="",subtitle = "based on
     100 iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.3, 0.8))+
  theme_minimal() + theme(plot.title = element_text(face =
     "bold",size = 18),
                        plot.subtitle = element_text(size
                           =12),
                        axis.text.y = element_text(size =
                           16))
```

https://doi.org/10.26219/heal.aueb.6189

```
MSEs <- ggpubr::ggarrange(boxplot_ridge_mse,boxplot_lasso_
   mse,boxplot_elnet_mse,nrow =1)
MAEs <- ggpubr::ggarrange(boxplot_ridge_mae,boxplot_lasso_
   mae,boxplot_elnet_mae,nrow=1)
penal_vizuals <-  ggpubr::ggarrange(MSEs,MAEs,nrow=2)
# Interpretation
elnet <- models_elastic[[10]]
# number of specific coefficients NOT to be zero out of the
    total 100 iterations
models_elnet <- as.data.frame(models_elastic)
for (col in names(models_elnet)) {
  models_elnet[[col]][abs(models_elnet[[col]]) < 0.05] <-
     0}
number_non_zero_coefs_elnet <- apply(models_elnet,1,
   function(x) sum(x!=0))
number_non_zero_coefs_elnet <- as.data.frame(number_non_
   zero_coefs_elnet)
number_non_zero_coefs_elnet
# histogram for number of predictors kept in each iteration
number_coefs_kept <- apply(models_elnet,2,function(x) sum(x
   !=0))
data <- data.frame(number_coefs_kept)
colnames(data) <- "NumberKept"
rownames(data) <- NULL
values_vars <- table(data$NumberKept)

# Regression Trees using tidymodels framework ---
# load libraries
library(MLmetrics)
library(parsnip)
library(caret)
library(rpart)
```

https://doi.org/10.26219/heal.aueb.6189

```
library(tidymodels)
library(cli)
library(rsample)
library(dplyr)
library(vip)
# algorithm
iter <- 100
cart_rmse <- rep(0, iter)
cart_mae <- rep(0, iter)
cost <- rep(0,iter)
depth <- rep(0,iter)
scores <- NULL
set.seed(1999)
start_time <- Sys.time()
for(i in 1:iter){
  split_breast= initial_split(data = df, prop = 0.7)
  train_breast = training(split_breast)
  test_breast = testing(split_breast)
  tune_spec <-
    decision_tree(
      cost_complexity = tune(),
      tree_depth = tune(),
    ) %>%
    set_engine("rpart") %>%
    set_mode("regression")
  metrics = metric_set(rmse)
  tree_grid <- grid_regular(cost_complexity(),
                            tree_depth(),
                            levels = 20)
  tree_wf <- workflow() %>%
    add_model(tune_spec) %>%
    add_formula(lpsa ~ .)
```

 https://doi.org/10.26219/heal.aueb.6189

```
    tv_folds = vfold_cv(train_breast)
    tree_res <-
      tree_wf %>%
      tune_grid(
        resamples = tv_folds,
        grid = tree_grid,
        metrics = metrics,)
    h <- tree_res %>%
      collect_metrics()
    best_tree <- tree_res %>%
      select_best("rmse")
    cost[i] <- best_tree$cost_complexity
    depth[i] <- best_tree$tree_depth
    final_wf <-
      tree_wf %>%
      finalize_workflow(best_tree)
    final_fit <-
      final_wf %>%
      last_fit(split_breast, metrics = metric_set(rmse,mae))
    estims = final_fit %>%
      collect_metrics()
    cart_rmse[i] <- estims$.estimate[1]
    cart_mae[i] <- estims$.estimate[2]
    importance_scores <- as.data.frame(final_fit %>%
                                      extract_fit_parsnip
                                         () %>% vi())
    scores <- rbind(scores,importance_scores)}
  end <- Sys.time() - start_time
  cart_mse <- cart_rmse^2
  df_cart <- data.frame("MSE" = cart_mse, "MAE" = cart_mae)
  df_metrics <- apply(df_cart,2,mean)
  # hyperameters tuned
```

https://doi.org/10.26219/heal.aueb.6189

```
table(depth)
table(cost)
# create MSE & MAE boxplots
elastic_mse  <- 0.45
elastic_mae <- 0.52
lin_mse <- 0.60
lin_mae <- 0.58
boxplot_cart_mse <- ggplot(data = df_cart, aes(x = "", y =
   MSE)) +
  geom_boxplot(fill = "royalblue2", alpha = 0.8) +
  geom_hline(yintercept = lin_mse, linetype = "dashed",
     color = "red") +
  geom_hline(yintercept = elastic_mse, linetype = "dotdash"
     , color = "green4") +
  labs(title = "Regression Trees MSE",x="",subtitle = "
     based on 100 iterations") +
  annotate("text", x = 0.5, y = lin_mse, label = "LinearReg
     ", color = "red", vjust = -0.5) +
  annotate("text", x = 0.5, y = elastic_mse, label = "
     ElasticNet", color = "green4", vjust = -0.5) +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.3,1.20)) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                          plot.subtitle = element_text(
                             size=14),
                          axis.text.y = element_text(size
                             =13))
boxplot_cart_mae <- ggplot(data = df_cart, aes(x = "", y =
   MAE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
```

```
  geom_hline(yintercept = lin_mae, linetype = "dashed",
     color = "red") +
  geom_hline(yintercept = elastic_mae,linetype="dotdash",
     color="green4") +
  annotate("text", x = 0.5, y = lin_mae, label = "LinearReg
     ", color = "red", vjust = -0.5) +
  annotate("text", x = 0.5, y = elastic_mae, label = "
     ElasticNet", color = "green4", vjust = -0.5) +
  labs(title="Regression Trees MAE",x="",subtitle = "based
     on 100 iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.50,1))+
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                               plot.subtitle = element_text(
                                  size=14),
                               axis.text.y = element_text(size
                                  =13))
viz_cart <- ggpubr::ggarrange(boxplot_cart_mse,boxplot_cart
   _mae,nrow=1)
# Get the tree visually
final_fit %>%
  extract_fit_engine() %>%
  rpart.plot::rpart.plot(roundint = FALSE)
# get the tree model
final_tree <- extract_workflow(final_fit)
# plot the variance importance scores for the last
   iteration
final_fit %>%
  extract_fit_parsnip() %>%
  vip(num_features=8)
final_fit %>%
```

https://doi.org/10.26219/heal.aueb.6189

```
  extract_fit_parsnip() %>% vi()
# get the variance importance plots for the total
   iterations ---
lcavol_index <- which(scores$Variable=="lcavol")
lcavol_data <- scores[lcavol_index,]
lcp_index <- which(scores$Variable=="lcp")
lcp_data <- scores[lcp_index,]
svi_index <- which(scores$Variable=="svi")
svi_data <- scores[svi_index,]
pgg45_index <- which(scores$Variable=="pgg45")
pgg45_data <- scores[pgg45_index,]
lbph_index <- which(scores$Variable=="lbph")
lbph_data <- scores[lbph_index,]
lweight_index <- which(scores$Variable=="lweight")
lweight_data <- scores[lweight_index,]
gleason_index <- which(scores$Variable=="gleason")
gleason_data <- scores[gleason_index,]
age_index <- which(scores$Variable=="age")
age_data <- scores[age_index,]
lcavol_ <- mean(lcavol_data[,-1])
lcp_ <- mean(lcp_data[,-1])
svi_ <- mean(svi_data[,-1])
pgg45_ <- mean(pgg45_data[,-1])
lbph_ <- mean(lbph_data[,-1])
lweight_ <- mean(lweight_data[,-1])
gleason_ <- mean(gleason_data[,-1])
age_ <- mean(age_data[,-1])
signific <- rbind(lcavol_, lcp_,  lbph_, svi_,
                  pgg45_, lweight_, gleason_, age_)
signific <- as.data.frame(signific)
rownames(signific) <- NULL
colnames(signific) <- "importance"
```

https://doi.org/10.26219/heal.aueb.6189

```
signific$variable <- c("cancer volume","capsular
    penetration","benign hyperplasia amount","seminal
    vesicle invasion",
                        "Gleason scores 4 or 5","prostate
                           weight","Gleason","age")
library("forcats")
importanceplot <- signific %>%
  mutate(variable = fct_reorder(variable, importance),
         importance = as.numeric(importance)) %>%
  ggplot(aes(x = variable, y = importance, fill = variable)
     ) +
  geom_bar(stat = "identity", alpha = 0.8, width = 0.4) +
  scale_fill_manual(values = c("#4169E1", "#4169E1", "#4169
     E1", "#4169E1", "#4169E1", "#4169E1", "#4169E1", "
     #4169E1")) +
  coord_flip() + xlab("") + theme_minimal() +
  labs(title = "Importance Scores", subtitle = "based on
     Breiman's method in 100 iterations") +
  theme( plot.title = element_text(face = "bold", size =
     18),
    plot.subtitle = element_text(size = 16),
    axis.text.y = element_text(size = 16),
    axis.text.x = element_text(size = 16)) +
  xlab("") + guides(fill = "none")

# Gradient Boosted Regression Trees using mlr3 and gbm
    packages ---
# algorithm
iter <- 100
boost_mse <- rep(0,iter)
boost_mae <- rep(0,iter)
boost_trees <- rep(0,iter)
```

https://doi.org/10.26219/heal.aueb.6189

```
boost_shr <- rep(0,iter)
scores <- list()
set.seed(1999)
start_time <- Sys.time()
for (i in 1:iter) {
  measure <- msr("regr.mse")
  learner = mlr3extralearners::lrn("regr.gbm",distribution=
     "gaussian")
  resample = rsmp("cv",folds=10)
  train_set = sample(1:length(df[,1]), 0.7 * length(df[,1])
     )
  test_set = setdiff(1:length(df[,1]), train_set)
  terminator = trm("none")
  param_set = ps(
    n.trees = p_int(lower=10,upper=100),
    shrinkage=p_dbl(lower=0.1,upper=0.3))
  task = TaskRegr$new(id=paste("boosted regression trees
     tune",i),
                    backend=df[train_set,],
                    target="lpsa")
  tuner = mlr3tuning::tnr("grid_search",resolution=15)
  instance_tune <- TuningInstanceSingleCrit$new(task=task,
     learner = learner,  resampling = resample,  measure=
        measure,
     terminator = terminator, search_space = param_set,
        store_models = TRUE)
  tuner$optimize(instance_tune)
  learner_tune = lrn("regr.gbm",distribution="gaussian")
  learner_tune$param_set$values$n.trees <- instance_tune$
     result_x_domain$n.trees
  learner_tune$param_set$values$shrinkage <- instance_tune$
     result_x_domain$shrinkage
```

https://doi.org/10.26219/heal.aueb.6189

```
  boost_trees[i] <- instance_tune$result_x_domain$n.trees
  boost_shr[i] <- instance_tune$result_x_domain$shrinkage
  learner_tune$train(task)
  scores[[i]] <- learner_tune$importance()
  prediction <- learner_tune$predict_newdata(df[test_set,])
  boost_mse[i] <- prediction$score(msr("regr.mse"))
  boost_mae[i] <- prediction$score(msr("regr.mae"))}
Sys.time()-start_time
# results
boost_df <- data.frame("MSE"=boost_mse,"MAE"=boost_mae,
                       "Shrinkage"=boost_shr,"Boosted trees
                          "=boost_trees)
boost_metrics <- round(apply(boost_df, 2, mean),2)
# tuned results
count_boost_trees <- as.data.frame(table(boost_trees))
count_boost_lr <- as.data.frame(round(boost_shr,3))
colnames(count_boost_lr) <- "rate"
plot1 <- ggplot(count_boost_trees, aes(x=as.factor(boost_
   trees),y=Freq)) +
  geom_bar(stat="identity",fill = "royalblue2",alpha=0.8) +
  labs(
    title = "Number of boosted trees",
    subtitle = "based on 100 iterations",
    x="",
    y = "Frequency") +
  ylim(c(0,20)) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                           plot.subtitle = element_text(
                              size=14),
                           axis.text.y = element_text(size
                              =13),
```

https://doi.org/10.26219/heal.aueb.6189

```
                              axis.text.x = element_text(size
                                 = 13))
plot2 <- ggplot(data = count_boost_lr, aes(x = "", y = rate
   )) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  labs(title = "Learning rate",x="",subtitle = "based on
     100 iterations") +
  scale_x_discrete(labels = NULL) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                              plot.subtitle = element_text(
                                 size=14),
                              axis.text.y = element_text(size
                                 =13))
tuned <- ggpubr::ggarrange(plot1,plot2,nrow=1)
# Boxplots for MSE and MAE ---
boxplot_boost_mse <- ggplot(data = boost_df , aes(x = "", y
    = MSE)) +
  geom_boxplot(fill = "royalblue2", alpha = 0.8) +
  geom_hline(yintercept = lin_mse, linetype = "dashed",
     color = "red") +
  geom_hline(yintercept = ridge_mse, linetype = "dotdash",
     color = "green4") +
  labs(title = "Boosting MSE",x="",subtitle = "based on 100
      iterations") +
  annotate("text", x = 0.5, y = lin_mse, label = "LinearReg
     ", color = "red", vjust = -0.5) +
  annotate("text", x = 0.5, y = ridge_mse, label = "
     RidgeReg", color = "green4", vjust = -0.5) +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.20,1.20)) +
```

https://doi.org/10.26219/heal.aueb.6189

```
  theme_minimal() +   theme(plot.title = element_text(face
     = "bold",size = 18),
                            plot.subtitle = element_text(
                               size=14),
                            axis.text.y = element_text(size
                               =13))
boxplot_boost_mae <- ggplot(data = boost_df, aes(x = "", y
   = MAE)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = lin_mae, linetype = "dashed",
     color = "red") +
  geom_hline(yintercept = ridge_mae,linetype="dotdash",
     color="green4") +
  annotate("text", x = 0.5, y = lin_mae, label = "LinearReg
     ", color = "red", vjust = -0.5) +
  annotate("text", x = 0.5, y = ridge_mae, label = "
     RidgeReg", color = "green4", vjust = -0.5) +
  labs(title="Boosting MAE",x="",subtitle = "based on 100
     iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.20,1.20)) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                            plot.subtitle = element_text(
                               size=14),
                            axis.text.y = element_text(size
                               =13))
viz_boost <- ggpubr::ggarrange(boxplot_boost_mse,boxplot_
   boost_mae,nrow=1)
# VIMP
library("dplyr")
library("forcats")
```

https://doi.org/10.26219/heal.aueb.6189

```
imp_scores <- round(as.matrix(unlist(scores)),4)
age_mat <- imp_scores[rownames(imp_scores)=="age"]
gleason_mat <- imp_scores[rownames(imp_scores)=="gleason"]
lcavol_mat <- imp_scores[rownames(imp_scores)=="lcavol"]
lbph_mat <- imp_scores[rownames(imp_scores)=="lbph"]
lcp_mat <- imp_scores[rownames(imp_scores)=="lcp"]
lweight_mat <- imp_scores[rownames(imp_scores)=="lweight"]
pgg45_mat <- imp_scores[rownames(imp_scores)=="pgg45"]
svi_mat <- imp_scores[rownames(imp_scores)=="svi"]
imp_matrix <- data.frame("variable"=c("age","Gleason","
   cancer volume","benign hyperplasia amount",
                                   "capsular penetration
                                      ","prostate weight
                                      ","Gleason scores
                                      4 or 5 ","seminal
                                      vesicle invasion")
                                      ,
                        "scores"=c(mean(age_mat),mean(
                           gleason_mat),mean(lcavol_mat),
                           mean(lbph_mat),mean(lcp_mat),
                                   mean(lweight_mat),mean(
                                      pgg45_mat),mean(svi_
                                      mat)))
bar <- imp_matrix %>%
  mutate(variable = fct_reorder(variable, scores),
         importance = as.numeric(scores)) %>%
  ggplot(aes(x = variable, y = scores,fill=variable)) +
  geom_bar(stat = "identity",alpha = 0.8, width = 0.4) +
  scale_fill_manual(values = c("#4169E1","#4169E1","#4169E1
     ","#4169E1","#4169E1","#4169E1","#4169E1","#4169E1"))
     +  coord_flip() + xlab("") + theme_minimal() + labs(
     title = "Importance Scores", subtitle = "based on
```

 https://doi.org/10.26219/heal.aueb.6189

```
      Breiman's method in 100 iterations") +
    theme( plot.title = element_text(face = "bold", size =
      18),
     plot.subtitle = element_text(size = 16), axis.text.y =
       element_text(size = 16),
     axis.text.x = element_text(size = 16)) + guides(fill="
       none")

# Random Forests using RandomForestSRC package ---
# algorithm
iter <- 100
num_iters <- 100
rf_mse <- rep(0, iter)
rf_mae <- rep(0, iter)
vimp_matrix <- NULL
tuned_results <- NULL
set.seed(1999)
start_time <- Sys.time()
for(i in 1:iter){
  train_set = sample(1:length(df[,1]), 0.7 * length(df[,1])
     )
  test_set = setdiff(1:length(df[,1]), train_set)
  config_ll <- rep(0, num_iters)
  num.trees_val <- sample(50:500, size = num_iters, replace
      = T)
  mtry_val <- sample(2:7, size = num_iters, replace = T)
  min.node.size_val <- sample(1:10, size = num_iters,
     replace = T)
  nodedepth_val <- sample(2:15, size = num_iters, replace =
      T)
  config_df <- data.frame("num.trees_val" = num.trees_val,
     "mtry_val" = mtry_val,
```

https://doi.org/10.26219/heal.aueb.6189

```
                            "min.node.size_val" = min.node.
                               size_val,
                            "nodedepth_val" = nodedepth_val,
                            "config_ll" = config_ll)
train2 <- createFolds(df[train_set, ]$lpsa, k = 5,
                      list = TRUE, returnTrain = T)
valid <- lapply(train2, function(x)
  setdiff(1:length(df[train_set,]$lpsa), x))
for(j in 1:num_iters){
  ll <- rep(0, 5)
  for(k in 1:5){
    modRFSRC <- rfsrc(lpsa ~ ., data = df[train2[[k]],],
                      ntree=config_df$num.trees_val[j],
                      nodesize = config_df$min.node.size_
                         val[j],
                      mtry = config_df$mtry_val[j],
                         splitrule = "mse",
                      nodedepth = config_df$nodedepth_val
                         [j])
    predictions <- predict(modRFSRC, df[valid[[k]],])
    p <- predictions$predicted
    true <- df[valid[[k]],]$lpsa
    ll[k] <- MSE(p, true) }
  config_df$config_ll[j] <- mean(ll)}
b <- which.min(config_df$config_ll)
tuned_values <- config_df[b,]
tuned_results <- rbind(tuned_results,tuned_values)
final_rf <- rfsrc(lpsa ~ ., data = df[train_set,],
                  ntree=config_df$num.trees_val[b],
                  nodesize = config_df$min.node.size_val[
                     b],
                  mtry = config_df$mtry_val[b],
```

https://doi.org/10.26219/heal.aueb.6189

```
                    splitrule = "mse",
                    nodedepth = config_df$nodedepth_val[b]
                    ,importance="permute")
  test_imp <- final_rf$importance
  vimp_matrix <- rbind(vimp_matrix,test_imp)
  final_preds <- predict(final_rf, df[test_set,])
  p <- final_preds$predicted
  true <- df[test_set,]$lpsa
  rf_mse[i] <- MLmetrics::MSE(p,true)
  rf_mae[i] <- MLmetrics::MAE(p,true)}
Sys.time()-start_time
df_rf <- data.frame("MSE" = rf_mse,
                    "MAE" = rf_mae)
# results
rf_metrics <- apply(df_rf, 2, mean)
# tuned results
count_mtry <- as.data.frame(table(tuned_results$mtry_val))
count_min_node_size <- as.data.frame(table(tuned_results$
   min.node.size_val))
count_node_depth <- as.data.frame(table(tuned_results$
   nodedepth_val))
count_total_trees <- as.data.frame(tuned_results$num.trees_
   val)
colnames(count_total_trees) <- "count"
plot1 <- ggplot(count_mtry, aes(x=as.factor(Var1),y=Freq))
   +
  geom_bar(stat="identity",fill = "royalblue2",alpha=0.8) +
  labs(
    title = "Number of predictors in each split",
    subtitle = "based on 100 iterations",
    x="",
    y = "Frequency") +
```

https://doi.org/10.26219/heal.aueb.6189

```
  ylim(c(0,60)) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                          plot.subtitle = element_text(
                             size=14))
plot2 <- ggplot(count_min_node_size, aes(x=as.factor(Var1),
   y=Freq)) +
  geom_bar(stat="identity",fill = "royalblue2",alpha=0.8) +
  labs(
    title = "Minimum node size ",
    subtitle = "based on 100 iterations",
    x="",
    y = "Frequency") +
  ylim(c(0,20)) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                          plot.subtitle = element_text(
                             size=14))
plot3 <- ggplot(count_node_depth, aes(x=as.factor(Var1),y=
   Freq)) +
  geom_bar(stat="identity",fill = "royalblue2",alpha=0.8) +
  labs(
    title = "Node depth ",
    subtitle = "based on 100 iterations",
    x="",
    y = "Frequency") +
  ylim(c(0,20)) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                          plot.subtitle = element_text(
                             size=14))
```

https://doi.org/10.26219/heal.aueb.6189

```
plot4 <- ggplot(data = count_total_trees, aes(x = "", y =
    count)) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  labs(title = "Total number of trees",x="",subtitle = "
     based on 100 iterations") +
  scale_x_discrete(labels = NULL) +
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                            plot.subtitle = element_text(
                               size=14))
tuned_plots <- ggpubr::ggarrange(plot1,plot2,plot3,plot4,
    ncol=2,nrow=2)
# boxplots for MSE and MAE
boxplot_rf_mse <- ggplot(data = df_rf , aes(x = "", y = MSE
    )) +
  geom_boxplot(fill = "royalblue2", alpha = 0.8) +
  geom_hline(yintercept = lin_mse, linetype = "dashed",
     color = "red") +
  geom_hline(yintercept = ridge_mse, linetype = "dotdash",
     color = "green4") +
  labs(title = "Random Forest MSE",x="",subtitle = "based
     on 100 iterations") +
  annotate("text", x = 0.5, y = lin_mse, label = "LinearReg
     ", color = "red", vjust = -0.5) +
  annotate("text", x = 0.5, y = ridge_mse, label = "
     RidgeReg", color = "green4", vjust = -0.5) +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.30,0.90)) +  # Set y-axis
      limits
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
```

https://doi.org/10.26219/heal.aueb.6189

```
                              plot.subtitle = element_text(
                                 size=12))
boxplot_rf_mae <- ggplot(data = df_rf, aes(x = "", y = MAE)
   ) +
  geom_boxplot(fill="royalblue2",alpha=0.8) +
  geom_hline(yintercept = lin_mae, linetype = "dashed",
     color = "red") +
  geom_hline(yintercept = ridge_mae,linetype="dotdash",
     color="green4") +
  annotate("text", x = 0.5, y = lin_mae, label = "LinearReg
     ", color = "red", vjust = -0.5) +
  annotate("text", x = 0.5, y = ridge_mae, label = "
     RidgeReg", color = "green4", vjust = -0.5) +
  labs(title="Random Forest MAE",x="",subtitle = "based on
     100 iterations") +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(limits = c(0.4,0.9))+
  theme_minimal() +  theme(plot.title = element_text(face =
      "bold",size = 18),
                              plot.subtitle = element_text(
                                 size=12))
viz_rf <- ggpubr::ggarrange(boxplot_rf_mse,boxplot_rf_mae,
   nrow=1)

# importance scores and plots
rownames(vimp_matrix) <- NULL
vimp_matrix <- as.data.frame(vimp_matrix)
lcavol_ <- mean(vimp_matrix$lcavol)
age_ <- mean(vimp_matrix$age)
gleason_ <- mean(vimp_matrix$gleason)
lbph_ <- mean(vimp_matrix$lbph)
lcp_ <- mean(vimp_matrix$lcp)
```

 https://doi.org/10.26219/heal.aueb.6189

```
lweight_ <- mean(vimp_matrix$lweight)
pgg45_ <- mean(vimp_matrix$pgg45)
svi_ <- mean(vimp_matrix$svi)
signific <- rbind(age_,
                  gleason_,
                  lbph_,
                  lcavol_,
                  lcp_,
                  lweight_,
                  pgg45_,
                  svi_)
signific <- as.data.frame(signific)
rownames(signific) <- c("age","gleason","lbph","lcavol","
   lcp","lweight","pgg45","svi")
colnames(signific) <- "score"
# create an importance bar plot
sign <- signific
sign <- as.data.frame(sign)
colnames(sign) <- "importance"
sign$variable <- c("age","Gleason","benign hyperplasia
   amount","cancer volume","capsular penetration",
                   "prostate weight","Gleason scores 4 or 5
                      ","seminal vesicle invasion")
bar <- sign %>%
  mutate(variable = fct_reorder(variable, importance),
         importance = as.numeric(importance)) %>%
  ggplot(aes(x = variable, y = importance, fill = variable)
     ) +
  geom_bar(stat = "identity", alpha = 0.8, width = 0.4) +
  scale_fill_manual(values = c("#FF6666", "#4169E1", "#4169
     E1", "#4169E1", "#4169E1", "#4169E1", "#4169E1", "
     #4169E1")) +  coord_flip() + theme_minimal() +
```

https://doi.org/10.26219/heal.aueb.6189

```
labs(title = "Importance scores", subtitle = "based on
   permutation method") +
theme( plot.title = element_text(face = "bold", size =
   18),
  plot.subtitle = element_text(size = 16), axis.text.y =
     element_text(size = 16),
  axis.text.x = element_text(size = 16)) + xlab("") +
     guides(fill = "none")
```