P8160 - Less is More: Comparing Logistic and Lasso-Logistic Regression in Breast Cancer Diagnosis

Group 1

Amy Pitts, Hun Lee, Jimmy Kelliher,
Tucker Morgan, Waveley Qiu

2022-04-01

Abstract

With the advancement of research methods and measurement techniques, there might exist a desire among researchers to take advantage of these new tools to collect and use as much data as possible in the construction of predictive models. We sought to evaluate this inclination by constructing two models from a breast cancer imaging dataset to predict diagnosis outcome: a logistic linear model, which contains all predictors of interest and has coefficients selected by a Newton-Raphson algorithm to maximize likelihood, and a logistic LASSO model, which contains an optimal number of predictors as controlled by a penalty term and has coefficients selected by a coordinate descent algorithm. When testing the performance of these two models in predicting diagnosis outcome, we found that the smaller logistic-lasso model out-performed the larger logistic-linear model in both of our measures of model fit (AUC and maximum specificity given unit sensitivity). Though results may vary from study to study and dataset to dataset, our case study presents evidence for the advantage of parsimony in constructing predictive models.

1. Introduction

1.1. Background

Breast cancer has become one of the most common kinds of cancer in the United States, and substantial efforts have been made to aid in early and accurate detection. Along with national public health initiatives encouraging women at certain ages to schedule mammograms, great strides have been made to improve tumor imaging technology used in screening procedures. These improvements allow clinicians more pathways to early diagnosis, as we have access to more data than ever before. However, data is not necessarily in bijective correspondence with *information* [1]. With more data comes more noise, and it becomes more important for statisticians and medical practitioners to separate signal from that noise.

1.2. Objectives

We investigate two key questions: (1) Does having more data always correspond to an advantage in diagnosis prediction? (2) Can we reduce the amount of data we need to collect while maintaining (or increasing) predictive power? Toward that end, we use a breast cancer imaging dataset (detailed below) to fit two different models, both of which aim to predict patient diagnosis outcomes.

Our first model is a logistic regression model with a logit link consisting of a full set of predictors (i.e., the "full model") and model parameters estimated through a Newton-Raphson optimization algorithm. The second model is a logistic LASSO model that (possibly) selects fewer predictors (i.e., the "optimal model"), which is implemented using a path-wise coordinate descent optimization algorithm and tuned via a five-fold cross validation to obtain an optimal penalization term (λ). The algorithms for each method are discussed below in the methods section and corresponding code can be found in the appendix.

2. Methods

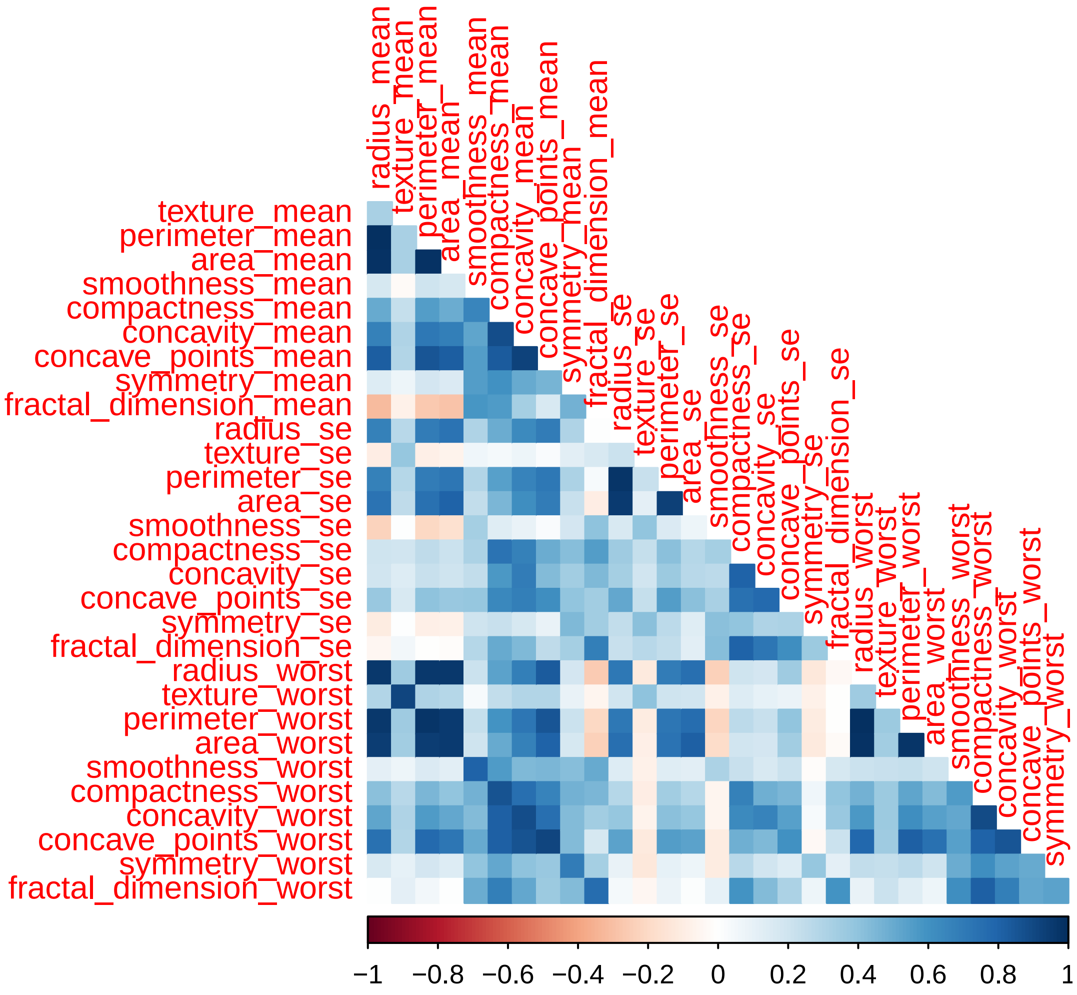
2.1. Data Cleaning and Exploratory Analysis

The dataset of interest contains 569 rows and 32 columns related to breast tissue imaging with each observation representing an individual patient. The outcome of interest is patient diagnosis, taking on values of either malignant or benign. One column contains information about patient ID, which will be excluded from our analysis. The other 30 columns correspond to summary statistics (mean, standard error, and maximum) of tumor radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. This dataset does not contain any missing values.

In exploring the data, we find many of the predictors are highly correlated with one another. In Figure 1, we see a correlation plot where variables with high correlations are denoted by dark shading. High collinearity between predictors is a threat to the convergence of our algorithms, particularly in the context of high-dimensional data. To explore the correlation further, Figure 2 shows the 25 largest correlations in our data. We see that the highest correlation is between `radius_mean` and `perimeter_mean` with a correlation coefficient of 0.998. In our dataset, there are 21 combinations of variables that achieve a correlation exceeding 0.90, which is a cause for concern in this context. We can break these 21 pairings into quasi-equivalence classes[1] for further inspection, where each quasi-equivalence class consists of variables that are all mutually pairwise correlated.

[1] We do not have precise equivalence classes, as transitivity only holds approximately. In particular, for variables x_1, x_2, x_3, $\text{corr}(x_1, x_2) > 0.90$ and $\text{corr}(x_2, x_3) > 0.90$ does not guarantee that $\text{corr}(x_1, x_3) > 0.90$, but x_1 and x_3 will still be highly correlated.

Figure 1: Correlation Heat Plot of all Covariates

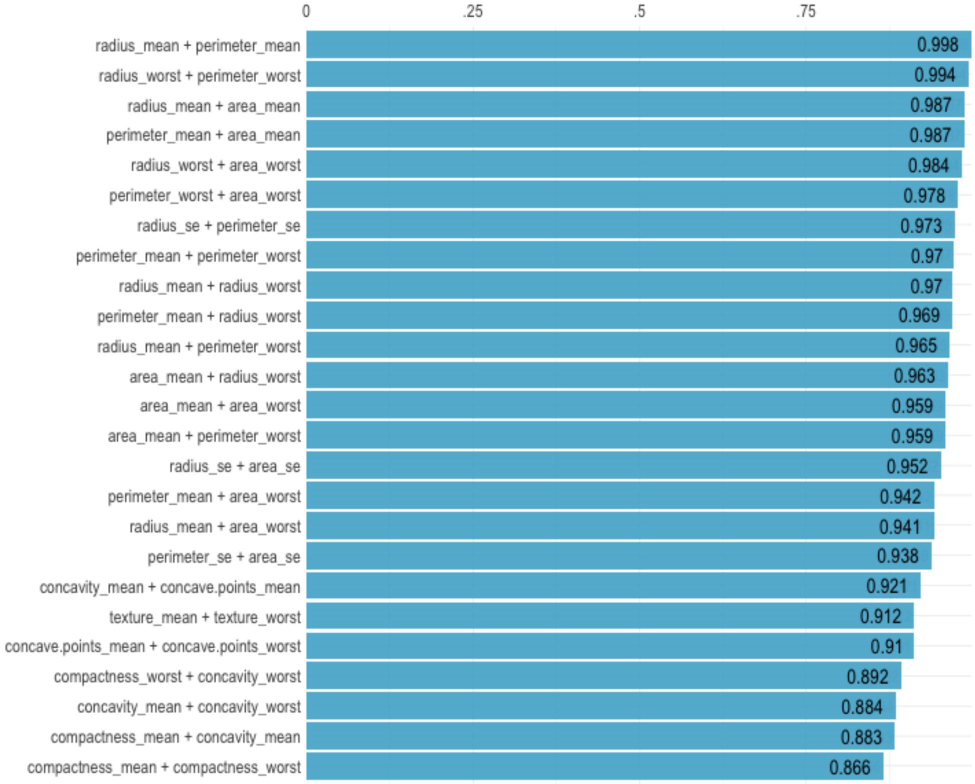


The first grouping of mutually correlated variables is {`area_mean`, `area_worst`, `perimeter_mean`, `perimeter_worst`, `radius_mean`, `radius_worst`}. This grouping represents 15 of the pairs in Figure 2. While we want to reduce the problem of collinearity, we do not want to simply discard all of these variables. Instead, we seek to identify the member of this class that best correlates with all other members. To identify the best representative for this class, we look at the variable with the highest mean correlation. In this case, we select `radius_worst`.

We repeat this procedure for each other quasi-equivalence class. From the class {`radius_se`, `perimeter_se`, `area_se`}, we select `radius_se`. From the class {`concavity_mean`, `concave_points_worst`, `concave_points_mean`}, we select `concave_points_mean`. Finally, for class {`texture_mean`, `texture_worst`}, the choice of representative is arbitrary, se we simply select `texture_mean`. Given our best representatives in each class, we discard the remaining predictors, leaving 20 predictors in our dataset. All surviving predictors can be seen in Table 1.

Figure 2: Ranked Cross-Correlations

25 most relevant



In Table 1, we see that there are 357 benign (B) cases and 212 malignant (M) cases. To evaluate the performance of both models, we partition our data into a training set (80%) and a testing set (20%). The data is standardized before fitting both of our models, which improves run times and simplifies parameter interpretation. For LASSO regression in particular, it is best practice to center and scale data, as the ℓ_1-norm penalization in LASSO regression tends to unequally penalize coefficient estimates if the covariates are of different magnitudes or scales. Therefore, standardizing our data ensures equal weighting and penalization.

Table 1: Patient Characteristics

Variable	**B**, N = 357	**M**, N = 212	**p-value**
texture_mean	17.91 (4.00)	21.60 (3.78)	<0.001
smoothness_mean	0.09 (0.01)	0.10 (0.01)	<0.001
compactness_mean	0.08 (0.03)	0.15 (0.05)	<0.001
concave_points_mean	0.03 (0.02)	0.09 (0.03)	<0.001
symmetry_mean	0.17 (0.02)	0.19 (0.03)	<0.001
fractal_dimension_mean	0.06 (0.01)	0.06 (0.01)	0.5
radius_se	0.28 (0.11)	0.61 (0.35)	<0.001
texture_se	1.22 (0.59)	1.21 (0.48)	0.6
smoothness_se	0.01 (0.00)	0.01 (0.00)	0.2
compactness_se	0.02 (0.02)	0.03 (0.02)	<0.001
concavity_se	0.03 (0.03)	0.04 (0.02)	<0.001
concave_points_se	0.01 (0.01)	0.02 (0.01)	<0.001
symmetry_se	0.02 (0.01)	0.02 (0.01)	0.028
fractal_dimension_se	0.00 (0.00)	0.00 (0.00)	<0.001
radius_worst	13.38 (1.98)	21.13 (4.28)	<0.001
smoothness_worst	0.12 (0.02)	0.14 (0.02)	<0.001
compactness_worst	0.18 (0.09)	0.37 (0.17)	<0.001
concavity_worst	0.17 (0.14)	0.45 (0.18)	<0.001
symmetry_worst	0.27 (0.04)	0.32 (0.07)	<0.001
fractal_dimension_worst	0.08 (0.01)	0.09 (0.02)	<0.001

2.2 Newton-Raphson Algorithm

The Newton-Raphson method is an efficient algorithm for computing the maximum likelihood estimator (MLE) under certain conditions. However, to even consider computing an MLE, we need to choose a reasonable model for our data so that we can construct our likelihood function.

Toward that aim, let $\mathbf{X} \equiv (x_{ij})$ denote the $n \times p$ matrix corresponding to our $n = 569$ observations and our $p = 21$ features (including an intercept term). Because our outcome $\mathbf{Y} \equiv (y_1, \ldots, y_n)^t$ is binary, it is natural to model $y_i \,|\, x_{i1}, \ldots, x_{ip} \sim \text{Bernoulli}(\pi_i)$, where

$$\begin{aligned} \pi_i &\equiv P(y_i = 1 \,|\, x_{i1}, \ldots, x_{ip}) \\ &= \left(1 + \exp\left(-\sum_{j=1}^{p} \beta_j x_{ij}\right)\right)^{-1}, \quad \text{such that} \end{aligned}$$

$$\log\left(\frac{\pi_i}{1 - \pi_i}\right) = \sum_{j=1}^{p} \beta_j x_{ij}$$

for a vector of parameters $\boldsymbol{\beta} \in \mathbb{R}^p$. The likelihood function of $\boldsymbol{\beta}$ given our data is

$$\begin{aligned}\mathcal{L}(\boldsymbol{\beta} \,|\, \mathbf{X}, \mathbf{Y}) &\equiv f(\mathbf{X}, \mathbf{Y} \,|\, \boldsymbol{\beta}) \\ &= \prod_{i=1}^{n} \pi_i^{y_i}(1-\pi_i)^{1-y_i} \\ &= \prod_{i=1}^{n} \left(\frac{\pi_i}{1-\pi_i}\right)^{y_i}(1-\pi_i).\end{aligned}$$

To employ the Newton-Raphson procedure, we need to compute the gradient vector and hessian matrix corresponding to $\mathcal{L}$. However, as computing derivatives of the likelihood function is tedious, we instead consider the log-likelihood function

$$\begin{aligned}l(\boldsymbol{\beta} \,|\, \mathbf{X}, \mathbf{Y}) &\equiv \log \mathcal{L}(\boldsymbol{\beta} \,|\, \mathbf{X}, \mathbf{Y}) \\ &= \log\left(\prod_{i=1}^{n}\left(\frac{\pi_i}{1-\pi_i}\right)^{y_i}(1-\pi_i)\right) \\ &= \sum_{i=1}^{n}\left(y_i \log\left(\frac{\pi_i}{1-\pi_i}\right) - \log\left(\frac{1}{1-\pi_i}\right)\right) \\ &= \sum_{i=1}^{n}\left(y_i \sum_{j=1}^{p}\beta_j x_{ij} - \log\left(1+\exp\left(\sum_{j=1}^{p}\beta_j x_{ij}\right)\right)\right).\end{aligned}$$

Toward computing the derivative of the l, we define

$$\eta_i \equiv \sum_{j=1}^{p}\beta_j x_{ij}$$

for each $i \in \{1, \ldots, n\}$. Observe that for each $k \in \{1, \ldots, p\}$,

$$\frac{\partial \eta_i}{\partial \beta_k} = x_{ik} \quad \text{and} \quad \frac{\partial \pi_i}{\partial \beta_k} = x_{ik}\pi_i(1-\pi_i).$$

Thus, it follows that

$$\begin{aligned}\frac{\partial l}{\partial \beta_k} &= \frac{\partial}{\partial \beta_k}\left(\sum_{i=1}^{n}\left(y_i\eta_i - \log\left(1+e^{\eta_i}\right)\right)\right) \\ &= \sum_{i=1}^{n}\left(y_i\frac{\partial \eta_i}{\partial \beta_k} - \frac{e^{\eta_i}}{1+e^{\eta_i}}\frac{\partial \eta_i}{\partial \beta_k}\right) \\ &= \sum_{i=1}^{n} x_{ik}(y_i - \pi_i), \quad \text{and}\end{aligned}$$

$$\begin{aligned}\frac{\partial l^2}{\partial \beta_k \beta_l} &= \frac{\partial}{\partial \beta_l}\left(\sum_{i=1}^{n} x_{ik}(y_i-\pi_i)\right) \\ &= -\sum_{i=1}^{n} x_{ik}\frac{\partial \pi_i}{\partial \beta_l} \\ &= -\sum_{i=1}^{n} x_{ik}x_{il}\pi_i(1-\pi_i).\end{aligned}$$

for $k, l \in \{1, \dots p\}$. The above expressions completely characterize the gradient vector and hessian matrix corresponding to the log-likelihood l. However, it will be convenient to express these objects compactly via matrix notation. Toward this aim, we define $\boldsymbol{\pi} \equiv (\pi_1, \dots, \pi_n)^t$ to be a vector of probabilities and

$$\mathbf{W} \equiv \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & w_n \end{pmatrix}, \quad \text{where} \quad w_i \equiv \pi_i(1 - \pi_i)$$

for each $i \in \{1, \dots, n\}$. We can think of $\mathbf{W}$ as a pseudo-weight matrix, noting that its entries do not generally sum to unity. Given this notation, the gradient vector and hessian matrix corresponding to l are given by

$$\nabla l(\boldsymbol{\beta} \,|\, \mathbf{X}, \mathbf{Y}) = \mathbf{X}^t(\mathbf{Y} - \boldsymbol{\pi}) \quad \text{and} \quad \nabla^2 l(\boldsymbol{\beta} \,|\, \mathbf{X}, \mathbf{Y}) = -\mathbf{X}^t\mathbf{W}\mathbf{X},$$

respectively. Our Newton-Raphson algorithm is then characterized by the procedure

$$\boldsymbol{\beta}_{t+1} = \boldsymbol{\beta}_t + (\mathbf{X}^t\mathbf{W}\mathbf{X})^{-1}\mathbf{X}^t(\mathbf{Y} - \boldsymbol{\pi}).$$

To assess if this updating procedure is well-behaved, we must investigate the properties of the hessian matrix, thereby motivating the following proposition.

Proposition 1. The hessian matrix of l is negative semi-definite. If $\pi_i \in (0, 1)$ for at least one $i \in \{1, \dots, n\}$, and if $\mathbf{X}$ is of full rank, then the hessian matrix is negative definite.

Proof. Because $\mathbf{W}$ is a diagonal matrix with non-negative elements, it is positive semi-definite. Thus, for any $u \in \mathbb{R}^p \setminus \{\mathbf{0}\}$, it follows that $\mathbf{X}u \in \mathbb{R}^p$, and hence

$$\begin{aligned} u^t \nabla^2 l(\boldsymbol{\beta} \,|\, \mathbf{X}, \mathbf{Y}) u &= -u^t \mathbf{X}^t \mathbf{W} \mathbf{X} u \\ &= -(\mathbf{X}u)^t \mathbf{W} (\mathbf{X}u) && \text{(by rules for transpose of a product)} \\ &\leq 0. && \text{(by positive semi-definiteness of } \mathbf{W}\text{)} \end{aligned}$$

That is, the hessian matrix is negative semi-definite. If we further have that $\mathbf{X}$ is of full rank, then $\mathbf{X}u \in \mathbb{R}^p \setminus \{\mathbf{0}\}$. Moreover, if $\pi_i \in (0, 1)$ for at least one $i \in \{1, \dots, n\}$, then $\mathbf{W}$ is positive definite. Together, these facts make the above inequality strict, such that the hessian matrix is negative definite. □

The above result provides us with two key insights: (1) if our data matrix is not of full rank, the hessian matrix might not be negative definite; and (2) if our fitted probabilities are all either zero or one, then our hessian matrix will not be negative definite. By culling highly correlated covariates during the exploratory data analysis phase, we have precluded scenario (1) (and indeed, we find that the Newton-Raphson algorithm does not converge if all 30 covariates are considered). To prevent scenario (2), we seed our initial guess at $\boldsymbol{\beta} = \mathbf{0}$, which corresponds to $\pi_i = \frac{1}{2}$ for all $i \in \{1, \dots, n\}$. As added precautions, we further control for ascent direction and allow step-halving, though this is not critical for such a well-behaved optimization problem.

2.3 Logistic LASSO Algorithm

The logistic LASSO algorithm is characterized by a constrained optimization problem that is not everywhere differentiable. As such, we consider an approximation of the log-likelihood function l and a coordinate descent procedure to make the problem more tractable.

To begin, we consider the function $g : \mathbb{R}^p \to \mathbb{R}$ given by

$$g(\boldsymbol{\beta}) = -\frac{1}{2n} \sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \eta(\boldsymbol{\alpha}),$$

which is *proportional* to the Taylor expansion of l centered around $\boldsymbol{\alpha} \in \mathbb{R}^p$, where

$$\begin{aligned} z_i &\equiv \sum_{j=1}^{p} \alpha_j x_{ij} + \frac{y_i - \pi_i}{w_i}, && \text{(effective response)} \\ w_i &\equiv \pi_i(1 - \pi_i), && \text{(effective weights)} \\ \pi_i &\equiv \left(1 + \exp\left(-\sum_{j=1}^{p} \alpha_j x_{ij}\right)\right)^{-1}, \end{aligned}$$

and some function $\eta : \mathbb{R}^p \to \mathbb{R}$ that does not depend on $\boldsymbol{\beta} \in \mathbb{R}^p$. The details of this derivation can be found in the mathematical appendix (Appendix H), but the result follows largely from the generalized version of Taylor's theorem applied to the gradient vector and hessian matrix computed in the previous section.

With our quadratic approximation g of l in tow, we consider the minimization problem

$$\operatorname*{arg\,min}_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\}.$$

This penalized optimization problem characterizes the updating procedure of our coordinate descent algorithm. However, as alleged earlier, our objective function is not differentiable at the origin. To find our desired minimizer, we consider a series of definitions and lemmas to reduce the complexity of the problem, as inspired by Friedman et al. (2007) [2].

Definition 1. Let $S : \mathbb{R} \times \mathbb{R}_+ \to \mathbb{R}$ be given by

$$S(\beta, \gamma) = \operatorname{sgn}(\beta) \max\{|\beta| - \gamma, 0\}.$$

We call S the *shrinkage function* or the *soft-thresholding operator*. Observe that the map $\beta \mapsto S(\beta, 0)$ is simply the identity function. Thus, we can think of $\gamma \geq 0$ as an additive penalty that shrinks β in magnitude toward zero.

Lemma 1. Let $f : \mathbb{R} \to \mathbb{R}$ be given by

$$f(x) = \frac{1}{2}(x - \beta)^2 + \gamma|x|$$

for $\beta \in \mathbb{R}$ and $\gamma \in \mathbb{R}_+$. It follows that

$$\operatorname*{arg\,min}_{x \in \mathbb{R}} f(x) = S(\beta, \gamma),$$

where S is the soft-thresholding operator.

Proof. Because f is a sum of the convex functions $\frac{1}{2}(x - \beta)^2$ and $\gamma|x|$, it is also a convex function. Thus, the minimizer $x^* \in \mathbb{R}$ of f is unique. While f is not differentiable at $x = 0$, it is the case that for all $x \neq 0$,

$$\frac{df}{dx} = x - \beta + \gamma \operatorname{sgn}(x).$$

Including our boundary case at $x = 0$, we obtain three critical points that serve as candidates for our unique minimum. Namely, $x^* \in \{0, \beta - \gamma, \beta + \gamma\}$, which satisfy

$$\begin{aligned} f(0) &= \frac{1}{2}\beta^2, \\ f(\beta - \gamma) &= \frac{1}{2}\gamma^2 + \gamma|\beta - \gamma|, \quad \text{and} \\ f(\beta + \gamma) &= \frac{1}{2}\gamma^2 + \gamma|\beta + \gamma|. \end{aligned}$$

Now, because $\gamma \geq 0$, it follows that $f(\beta - \gamma) < f(\beta + \gamma)$ if and only if $|\beta - \gamma| < |\beta + \gamma|$ if and only if $\beta > 0$. Moreover, if $\beta > \gamma$, then $|\beta - \gamma| = \beta - \gamma$, and hence

$$\begin{aligned} f(\beta - \gamma) - f(0) &= \frac{1}{2}\gamma^2 + \gamma|\beta - \gamma| - \frac{1}{2}\beta^2 \\ &= \gamma(\beta - \gamma) - \frac{1}{2}(\beta + \gamma)(\beta - \gamma) \\ &= -\frac{1}{2}(\beta - \gamma)^2 \\ &< 0. \end{aligned}$$

That is, $f(\beta - \gamma) < f(0)$ if and only if $\beta > \gamma$. By analogous argument, we find that $f(\beta + \gamma) < f(0)$ if and only if $\beta < -\gamma$. Combining all of these results, we obtain

$$x^* = \begin{cases} \beta - \gamma & \text{if} \quad \beta > \gamma, \\ \beta + \gamma & \text{if} \quad \beta < -\gamma, \\ 0 & \text{otherwise.} \end{cases}$$

Of course, the above expression is precisely $x^* = S(\beta, \gamma)$. □

Lemma 2. Consider the optimization problem

$$\min_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\}$$

for some $k \in \{1, \ldots, p\}$. It follows that the minimizer is given by

$$\hat{\beta}_k = \left(\sum_{i=1}^{n} w_i x_{ik}^2 \right)^{-1} \sum_{i=1}^{n} w_i x_{ik} \left(z_i - \sum_{j \neq k} \beta_j x_{ij} \right).$$

Proof. We first remark that minimizing our objective function is equivalent to maximizing our quadratic approximation g, as the two functions differ by a constant (with respect to β_k) and in signature. For convenience, we refine earlier notation via

$$\eta_i \equiv \sum_{j=1}^{p} \beta_j x_{ij} \quad \text{and} \quad \eta_i^{(k)} \equiv \eta_i - \beta_k x_{ik}$$

for each $i \in \{1, \ldots, n\}$. Note that $\eta_i^{(k)}$ does not depend on β_k. Because $-g$ is convex with respect β_k, it has a unique minimizer $\hat{\beta}_k$ satisfying

$$\begin{aligned} 0 &= -n \frac{\partial g}{\partial \beta_k} \\ &= \frac{\partial}{\partial \beta_k} \left(\frac{1}{2} \sum_{i=1}^{n} w_i \Big(z_i - \eta_i \Big)^2 \right) \\ &= \sum_{i=1}^{n} w_i \Big(\eta_i - z_i \Big) \frac{\partial \eta_i}{\partial \beta_k} \\ &= \sum_{i=1}^{n} w_i x_{ik} \Big(\hat{\beta}_k x_{ik} + \eta_i^{(k)} - z_i \Big) \\ &= \left(\sum_{i=1}^{n} w_i x_{ik}^2 \right) \hat{\beta}_k - \sum_{i=1}^{n} w_i x_{ik} \left(z_i - \eta_i^{(k)} \right). \end{aligned}$$

Rearranging the above expression gives the desired result. □

Lemma 3. With $\hat{\beta}_k$ defined as above,

$$\min_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} = \min_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2} (\beta_k - \hat{\beta}_k)^2 + \lambda_k |\beta_k| \right\}, \quad \text{where}$$

$$\lambda_k \equiv \left(\frac{1}{n} \sum_{i=1}^{n} w_i x_{ik}^2 \right)^{-1} \lambda.$$

Proof. For convenience, we consider the following definition: for any functions $h, k : \mathbb{R}^p \to \mathbb{R}$, we say that $h(\boldsymbol{\beta}) \simeq k(\boldsymbol{\beta})$ if and only if the difference $h(\boldsymbol{\beta}) - k(\boldsymbol{\beta})$ does not depend on β_k. That is, h and k define the same optimization problem with respect to β_k. Observe that

$$\begin{aligned}
\sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 &= \sum_{i=1}^{n} w_i \Big(z_i - \eta_i \Big)^2 \\
&\simeq \sum_{i=1}^{n} w_i \left(\left(\beta_k x_{ik} + \eta_i^{(k)} \right)^2 - 2 z_i \left(\beta_k x_{ik} + \eta_i^{(k)} \right) \right) \\
&\simeq \sum_{i=1}^{n} w_i \left(\beta_k^2 x_{ik}^2 + 2 \beta_k x_{ik} \eta_i^{(k)} - 2 z_i \beta_k x_{ik} \right) \\
&= \left(\sum_{i=1}^{n} w_i x_{ik}^2 \right) \beta_k^2 - 2 \left(\sum_{i=1}^{n} w_i x_{ik} \left(z_i - \eta_i^{(k)} \right) \right) \beta_k \\
&= \left(\sum_{i=1}^{n} w_i x_{ik}^2 \right) \left(\beta_k^2 - 2 \hat{\beta}_k \beta_k \right) \\
&\simeq \left(\sum_{i=1}^{n} w_i x_{ik}^2 \right) \left(\beta_k - \hat{\beta}_k \right)^2 .
\end{aligned}$$

Similarly, we further obtain

$$\lambda \sum_{j=1}^{p} |\beta_j| \simeq \lambda |\beta_k|.$$

Thus, our optimization problem can be rewritten as

$$\begin{aligned}
&\min_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \\
&= \min_{\beta_k \in \mathbb{R}} \left\{ \left(\frac{1}{n} \sum_{i=1}^{n} w_i x_{ik}^2 \right) \frac{1}{2} \left(\beta_k - \hat{\beta}_k \right)^2 + \lambda |\beta_k| \right\}.
\end{aligned}$$

Because x_{ik} is a continuous random variable in our data, if not all fitted probabilities are zero or one (a scenario we again mitigate), then $\frac{1}{n} \sum_{i=1}^{n} w_i x_{ik}^2 > 0$ almost surely. As such, dividing through by this term establishes the desired result. □

Proposition 2. With $\hat{\beta}_k$ and λ_k defined as above,

$$\arg\min_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2n} \sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right\} = S \left(\hat{\beta}_k, \lambda_k \right).$$

Proof. By Lemma 3, our minimization problem is equivalent to that of

$$\min_{\beta_k \in \mathbb{R}} \left\{ \frac{1}{2}(\beta_k - \hat{\beta}_k)^2 + \lambda_k |\beta_k| \right\}.$$

By Lemma 1, the solution to this minimization problem is uniquely $S\left(\hat{\beta}_k, \lambda_k\right)$. □

Remarkable! The above result informs us that maximizer of g (and hence, the approximate maximizer of l) subject to our LASSO constraint is simply the un-penalized maximizer $\hat{\beta}_k$ shrunken by weight-adjusted penalty λ_k. Given this, our coordinate descent algorithm proceeds as follows.

- Outer Loop: Decrement over $\lambda \in (\lambda_{\max}, \ldots, \lambda_{\min})$, where $\lambda_{\max} \equiv \frac{1}{n} \max\{|\mathbf{X}^t\mathbf{Y}|\}$.
- Middle Loop: Update $\boldsymbol{\alpha} = \boldsymbol{\beta}$ and Taylor expand g around $\boldsymbol{\alpha}$.
- Inner Loop: Update $\beta_k = S\left(\hat{\beta}_k, \lambda_k\right)$ sequentially for $k \in \{0, 1, \ldots, p, 0, 1, \ldots, p, 0, 1, \ldots\}$ until convergence.

Note: the middle loop terminates when a given Taylor expansion no longer yields updates (within the specified tolerance) to $\boldsymbol{\beta}$ in the inner loop. That is, if we update $\boldsymbol{\alpha}$ (and hence, π_i, w_i, z_i, and g) in the middle loop, and if we then find that no elements of $\boldsymbol{\beta}$ are updated in the following inner loop, we have found our optimal $\boldsymbol{\beta}$ given penalty λ.

2.4 Five-fold Cross Validation

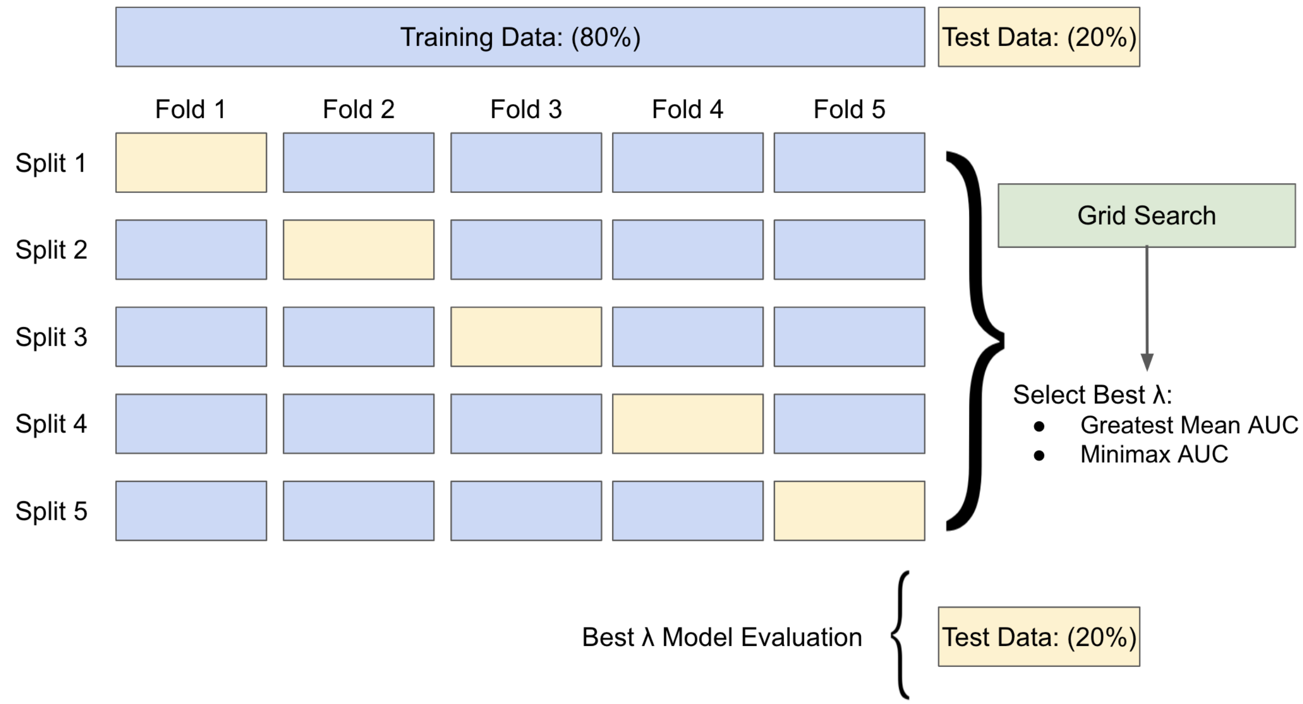
As the performance of the logistic LASSO model depends on the penalty factor (λ), we perform cross validation in order to select the λ that produces a model that optimizes a metric of interest.

First, we define our range of possible λ values, where the largest value produces a model with no predictors selected and the smallest value would produce a model with all predictors selected. We define the maximum value as the smallest penalty for which $\beta_k = 0$ for all $k \in \{1, \ldots, p\}$. This value works out to be the maximum of the inner product between our predictors (X) and our outcome (Y) from the imaging dataset. For λ_{min}, we select the a sufficiently small value that selects the full model. As recommended by Friedman et. al (2009) [3], we started by setting the smallest value in our range to be $\lambda_{min} = \frac{\lambda_{max}}{1000}$. However, we were not able to use this value as as the minimum λ in our range because it did not select the full model. Thus, we used the suggested $\frac{\lambda_{max}}{1000}$ as a starting point and empirically derived our λ_{min} by halving the working value until the full model was selected. Ultimately, we find that 0.0001 $\left(\approx \frac{\lambda_{max}}{4000}\right)$ is sufficiently small to select the full model. We then constructed a sequence between $\log \lambda_{max}$ and $\log \lambda_{min}$ so that the sequence would contain 100 values with equal intervals between values and exponentiated all values in that sequence. In this fashion, we thus created a range of 100 values between λ_{max} and λ_{min} on a logarithmic scale.

After defining this range of λ values, we implemented a five-fold cross validation algorithm to identify the best λ. In Figure 3, the first step of the process is to split the full dataset into train and test data as described above. The test data will not be touched until an optimal λ is selected and final method comparison is performed. Using just the training data the cross validation procedure implements a five-fold process where the test data is split into five "folds" or chunks. During the first iteration, the first fold of data is used as a test or "validation" set while the four other splits are used as the training set. The second iteration will use the second fold as the validation set and the remaining folds as the training set. This process continues for each fold, producing an Area-Under-the- ROC-Curve (AUC) value for each λ in our range of values. For example, in one fold we have 100 λ values all with one corresponding AUC value. Thus, for five folds each λ will have five associated AUC values.

To select our optimal tuning parameter, we use two measures: greatest mean AUC and Minimax AUC from 5-fold cross validation. To be specific, we are using 80% of training set in each split five times to compute mean AUC for each 100 λ values and selects the lambda which gives the greatest mean AUC as well as to compute maximum $1 - AUC$ and selects the lambda which gives the minimum of the maximum $1 - AUC$. The greatest mean AUC is to account for the scenario where we achieve the highest predictive ability to distinguish between classes (separabiltiy) on average and minimax AUC is to account for the scenario where we aim to minimize the possible loss for the worst case scenario. The greatest mean AUC and Minimax AUC optimal models are compared against the full model below.

Figure 3: Cross Validation Procedure



As we implemented the five-fold cross validation process, we grew concerned over the potential of our step size between λ values being too large and that the changes between models selected by consecutive were too dramatic. To increase the precision of our estimate and to prevent the process from removing more than one variable between increasing penalty values, we modified the simple cross validation procedure to include a grid search around perceived maximums when any two models corresponding with consecutive λ values differed by the inclusion/exclusion of more than one predictor after a five-fold cross validation process had been completely run through. When this occurred, we created a new range of possible λ values centered around the λ value ($\hat{\lambda}_{max}$) that had been identified to maximize the mean of the five AUC values produced from the cross validation process.

We will create our new range using the same logarithmic-exponentiation process we used to construct our initial range of λ values. Using the same (un-exponentiated) step size between $\log \lambda$ values in the original sequence of values, we defined the maximum of the new sequence of values to be two of those steps above $\log \hat{\lambda}_{max}$ and the minimum to be two steps below. Finally, we constructed a uniform sequence of the same length as the original sequence (consisting of 100 values) between the two endpoints and exponentiated the result to produce our new truncated range of λ values. From here, we will run the five-fold cross validation again, and repeat the process until consecutive λ values do not remove more than one predictor between models as λ increases.

2.5 Final Model Evaluation

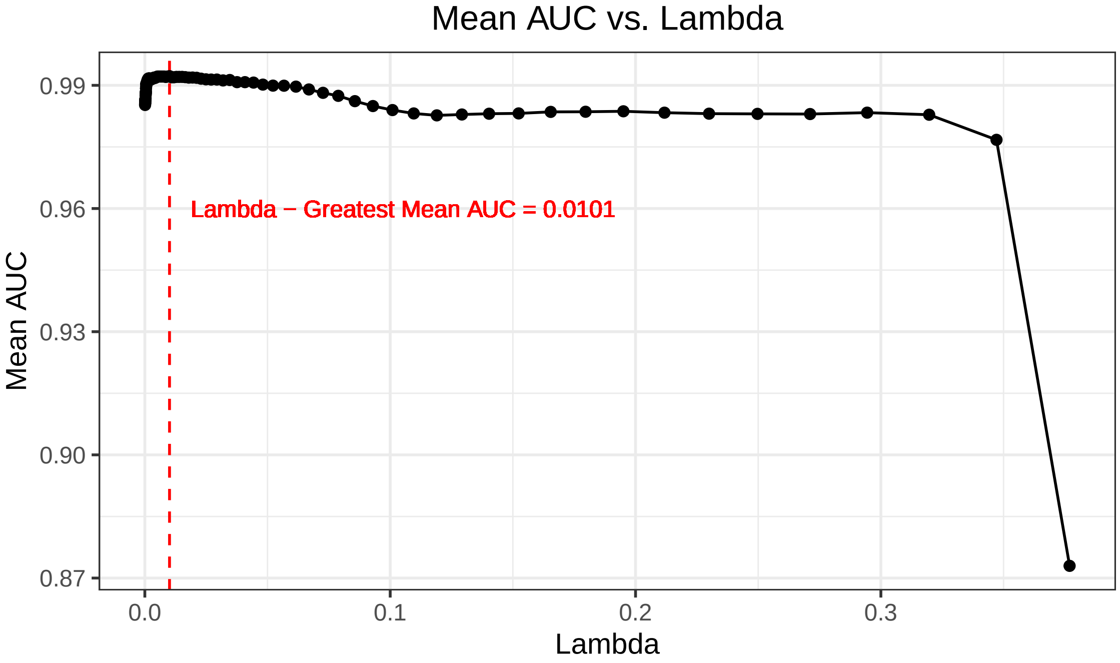
Each optimal λ selected is then used to fit a final model on the full training dataset. To compare these optimal models with the full (Newton-Raphson) model, we use the β_j values specified by each to make classification predictions based on the test dataset, which is held out of analysis prior to this step. In other words, the β_j values in each model are used in conjunction with the test data, X_{ij}, to predict class probabilities for each testing data observation. These predicted probabilities are compared with the observed outcomes in the testing data to produce receiver operating characteristic (ROC) curves, as well as AUC, sensitivity, and specificity values.

3. Results

3.1 Cross-Validation Results

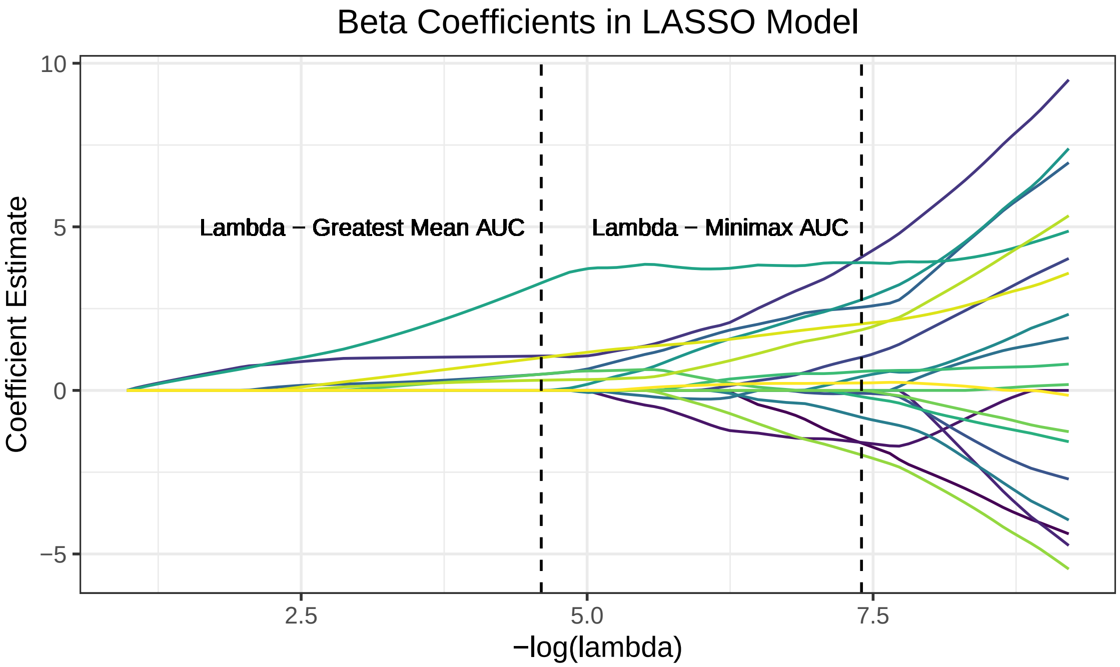
When evaluating our cross validation results, we first confirm that our range of λ values has one maximum AUC value when using the greatest mean AUC optimal lambda method. This can be seen in Figure 4, where the largest mean AUC and corresponding lambda value has been identified.

Figure 4: Cross Validation Results: Selecting Best Lambda



We can also evaluate our cross validation results by looking at the beta estimates for each corresponding λ value (Figure 5). The smallest value of -log(lambda) corresponds with our null model where all estimates of β_j are zero and the λ penalization term is large. The largest value of -log(lambda) corresponds with a full model where no β_j values are zero. We see that the greatest mean AUC selects a larger λ value, greater penalization, and thus a model with fewer predictor coefficients compared to the minimax AUC method.

Figure 5: Cross Validation Results: LASSO Coefficients

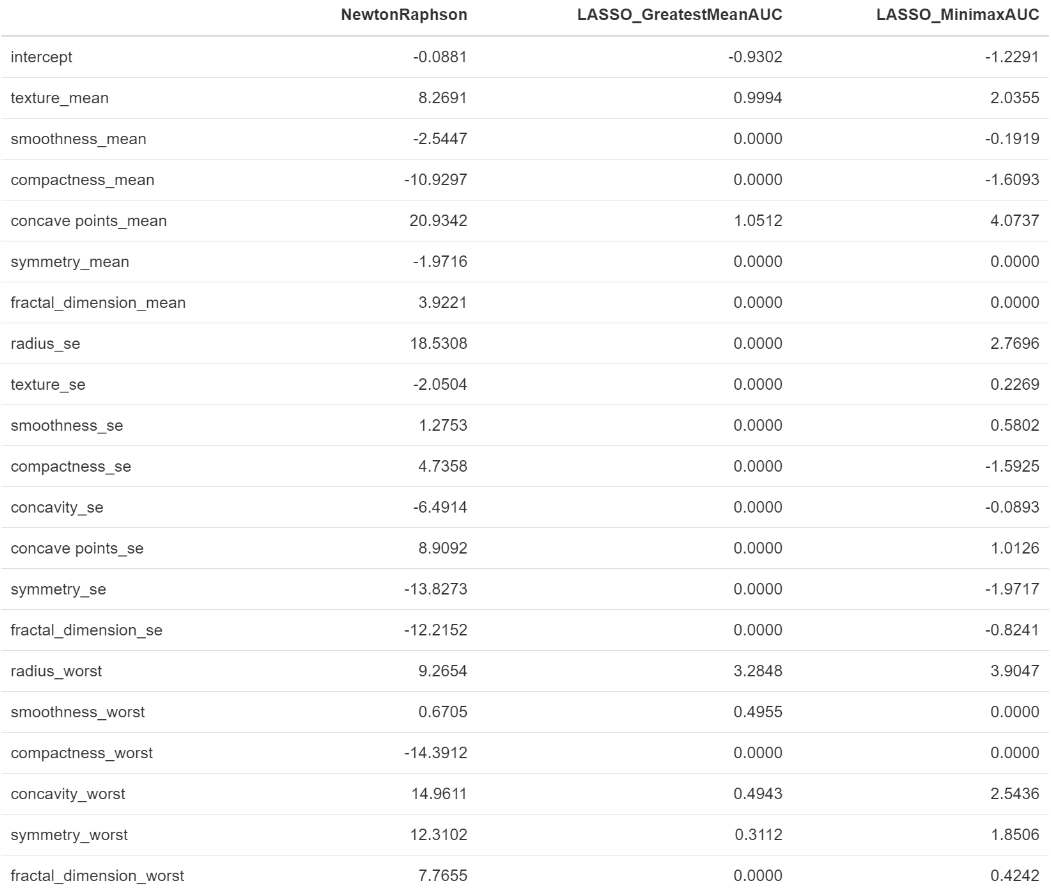


3.2 Model Validation Results

We then compared the β_i estimates in each of our models (Table 2). As our data had been standardized prior to constructing our models, these estimates must be interpreted with caution (especially so for the logistic LASSO model, with its inclusion of a penalty term). We found that the logistic LASSO model constructed using a λ that maximized greatest mean AUC has the most β_i values of 0, indicating that this penalty term selects the fewest predictors. The logistic LASSO model constructed using a λ that optimized on Minimax AUC selects 16 nonzero β_i values. Of importance to note is that between the three models, the smaller models are always nested within the larger models – that is, all the predictors contained in the smallest logistic LASSO model are contained within the larger logistic LASSO model, and all of the predictors contained within the larger logistic LASSO model are contained within the full logistic linear model.

We then compared the β_i estimates in each of our models (Table 2). As our data had been standardized prior to constructing our models, these estimates must be interpreted with caution (especially so for the logistic LASSO model, with its inclusion of a penalty term). We found that the logistic LASSO model constructed using a λ that maximized greatest mean AUC has the most β_i values of 0, indicating that this penalty term selects the fewest . Table 3 also confirms that greatest mean AUC produces a model with 6 nonzero beta values with lambda value 0.0101, not including the intercept term. Minimax optimization selects 16 nonzero beta values with lambda value 0.0006. While each model has test AUC values above 0.97, the model with the largest test AUC is the greatest mean optimal model.

Table 2: Beta Coefficients Comparing Full and Optimal Models

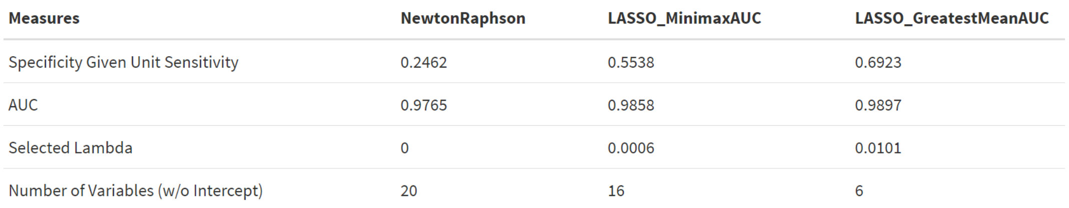


	NewtonRaphson	LASSO_GreatestMeanAUC	LASSO_MinimaxAUC
intercept	-0.0881	-0.9302	-1.2291
texture_mean	8.2691	0.9994	2.0355
smoothness_mean	-2.5447	0.0000	-0.1919
compactness_mean	-10.9297	0.0000	-1.6093
concave points_mean	20.9342	1.0512	4.0737
symmetry_mean	-1.9716	0.0000	0.0000
fractal_dimension_mean	3.9221	0.0000	0.0000
radius_se	18.5308	0.0000	2.7696
texture_se	-2.0504	0.0000	0.2269
smoothness_se	1.2753	0.0000	0.5802
compactness_se	4.7358	0.0000	-1.5925
concavity_se	-6.4914	0.0000	-0.0893
concave points_se	8.9092	0.0000	1.0126
symmetry_se	-13.8273	0.0000	-1.9717
fractal_dimension_se	-12.2152	0.0000	-0.8241
radius_worst	9.2654	3.2848	3.9047
smoothness_worst	0.6705	0.4955	0.0000
compactness_worst	-14.3912	0.0000	0.0000
concavity_worst	14.9611	0.4943	2.5436
symmetry_worst	12.3102	0.3112	1.8506
fractal_dimension_worst	7.7655	0.0000	0.4242

We then compared the two measures of model fit for these three models. When looking at the prediction AUC produced by each of these models on the test set we had withheld, we found that the smaller logistic LASSO model produced an AUC of 0.9897, the larger logistic LASSO model produced an AUC of 0.9858, and the full logistic linear model produced an AUC of 0.9765. Thus, we found that the logistic LASSO models both out-performed the logistic linear model, with the smallest logistic LASSO model producing the largest AUC overall.

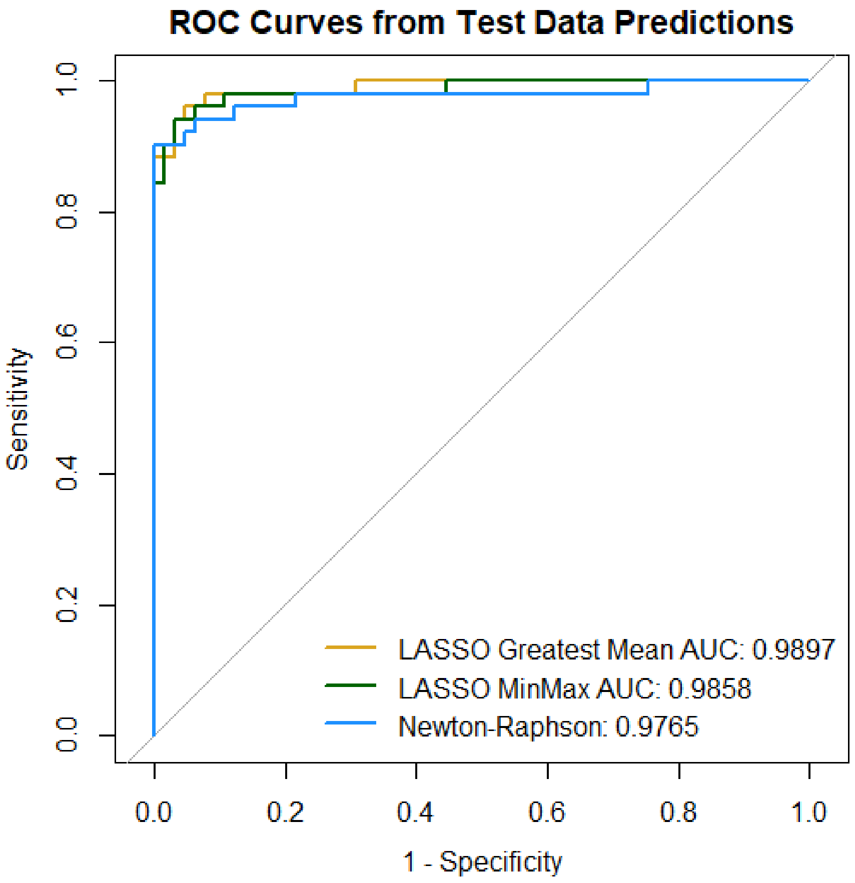
While the AUC value is a good indicator of overall classification performance, we are still confronted with the trade-off between sensitivity and specificity (Figure 6). Imagining is typically the first line of defense for diagnosing cancers, so optimizing sensitivity is usually of great interest. Thus, ideally, for clinical diagnosis purposes, it would be good to recommend a model that sacrifices the least amount of specificity while still achieving a sensitivity of 1 (Table 3). We found that the greatest mean AUC lasso model has the highest specificity of 69.23% when holding sensitivity at 1. Though the larger logistic LASSO model had its penalty term selected using the “minimum-worst-AUC” criteria in our cross validation process, it still achieved a much larger specificity than the full model, 55.38% when controlling for sensitivity. Namely, if we choose to a lasso model, it is still expected to manage to achieve 55.38% specificity while maintaining a sensitivity of 1 in the worst case scenario. It is to be observed that full model has the lowest sensitivity, 24.62%, and hence needs to sacrifice specificity more than 75% to achieve 1 sensitivity.

Table 3: Model Summary Comparing Full and Optimal Models



Measures	NewtonRaphson	LASSO_MinimaxAUC	LASSO_GreatestMeanAUC
Specificity Given Unit Sensitivity	0.2462	0.5538	0.6923
AUC	0.9765	0.9858	0.9897
Selected Lambda	0	0.0006	0.0101
Number of Variables (w/o Intercept)	20	16	6

Figure 6: Comparing Full and Optimal Models Performance



4. Discussion

4.1. Summary of Findings

Comparing the Newton-Raphson full model against two optimal models (greatest mean AUC, Minimax AUC) the model that out-preformed the others is the greatest mean AUC optimal model. This greatest mean AUC optimal model had the fewest predictors, highest AUC value, and highest specificity when maximizing sensitivity. Of course the ideal performance is to accurately classify every patient, AUC = 1; although very close, our test performance does not reach AUC = 1. Therefore, we need to balance sensitivity and specificity, to determine the lesser of two errors: false positives vs false negatives, when setting decision boundaries.

It is clear from our results that having more data does not always correspond to an advantage in diagnosis prediction. We found better prediction with many fewer predictors, six, in our optimal model compared to a full 20 in the model incorporating the most data. Given these results, it may benefit clinicians and practitioners to focus on certain indicators or attributes of breast imaging data in an effort to separate signal from noise.

Lastly, we find that that 6 variables in particular - `texture_mean`, `concave_points_mean`, `radius_worst`, `smoothness_worst`, `concavity_worst`, and `symmetry_worst` - play important roles in predicting whether breast imaging data is malignant.

4.2. Limitations

While the initial reduction of variables to limit high correlations was beneficial, we may have not selected the best representative of the correlation group. We did not try different representatives of the equivalence classes and so this might be a limitation that could prompt further study. Additionally, though we primarily used AUC as a measure of model performance in our study, future studies may find the measure of the accuracy rate of prediction interesting to use, if the information of the cutoff probability for classification is given from medical professionals.

We also see AUC values very close to a perfect 1.0 in each of the models considered. This could be due to very clean and unambiguous data, which limits our knowledge of how these models might actually perform in a "real-world" setting. In the future, it would be beneficial to examine different data sets in order to better understand how these models perform on breast imaging data.

4.3 Future Work

Two avenues of future work were discussed above, including the consideration of different model evaluation criteria based on clinician recommendation and implementation on more ambiguous data sets. Another interesting improvement could be the implementation of Monte Carlo Cross Validation. In this project, we performed five-fold cross validation once, however this procedure can be repeated multiple times with varied folds of the training data to obtain more stable estimates of the optimal λ value. Additionally, models may perform better on larger datasets. Here, we only had 569 observations, but more observations could help us learn more about the relationships between the imaging data and diagnosis outcome.

4.4. Group Contributions

Our group worked in conjunction on many aspects of the project. Amy, Waveley, and Hun worked on developing and implementing the Newton-Raphson optimization algorithm. Jimmy worked on developing and implementing the LASSO coordinate-descent algorithm. Tucker and Waveley worked on the cross validation procedure and plotting results from these output. All group members worked on the project presentation and report in varying capacities.

References

[1] Duncan, J. R. (2017, September 1). Information overload: When less is more in medical imaging. De Gruyter. https://www.degruyter.com/document/doi/10.1515/dx-2017-0008/html?lang=en

[2] Jerome Friedman. Trevor Hastie. Holger Höfling. Robert Tibshirani. "Pathwise coordinate optimization." Ann. Appl. Stat. 1 (2) 302 - 332, December 2007. https://doi.org/10.1214/07-AOAS131.

[3] Friedman J, Hastie T, Tibshirani R. Regularization Paths for Generalized Linear Models via Coordinate Descent. J Stat Softw. 2010;33(1):1-22. PMID: 20808728; PMCID: PMC2929880.

Appendices

Appendix A: Comparison to GLMNET Estimations

Coefficients	glmnet_full	NewtonRaphson	glmnet_GreatestMeanAUC	LASSO_GreatestMeanAUC	glmnet_MinimaxAUC	LASSO_MinimaxAUC
intercept	-1.7917	-0.0881	-0.9303	-0.9302	-1.2294	-1.2291
texture_mean	8.2499	8.2691	0.9999	0.9994	2.0360	2.0355
smoothness_mean	-2.5990	-2.5447	0.0000	0.0000	-0.1926	-0.1919
compactness_mean	-10.9923	-10.9297	0.0000	0.0000	-1.6107	-1.6093
concave points_mean	20.6086	20.9342	1.0513	1.0512	4.0760	4.0737
symmetry_mean	-1.9932	-1.9716	0.0000	0.0000	0.0000	0.0000
fractal_dimension_mean	3.9817	3.9221	0.0000	0.0000	0.0000	0.0000
radius_se	17.8593	18.5308	0.0000	0.0000	2.7709	2.7696
texture_se	-2.0888	-2.0504	0.0000	0.0000	0.2269	0.2269
smoothness_se	1.2916	1.2753	0.0000	0.0000	0.5804	0.5802
compactness_se	4.6236	4.7358	0.0000	0.0000	-1.5929	-1.5925
concavity_se	-6.7292	-6.4914	0.0000	0.0000	-0.0893	-0.0893
concave points_se	9.0910	8.9092	0.0000	0.0000	1.0135	1.0126
symmetry_se	-13.0681	-13.8273	0.0000	0.0000	-1.9728	-1.9717
fractal_dimension_se	-12.3642	-12.2152	0.0000	0.0000	-0.8251	-0.8241
radius_worst	9.1506	9.2654	3.2863	3.2848	3.9048	3.9047
smoothness_worst	0.6767	0.6705	0.4958	0.4955	0.0000	0.0000
compactness_worst	-14.4338	-14.3912	0.0000	0.0000	0.0000	0.0000
concavity_worst	15.0936	14.9611	0.4946	0.4943	2.5440	2.5436
symmetry_worst	11.5523	12.3102	0.3113	0.3112	1.8515	1.8506
fractal_dimension_worst	7.7952	7.7655	0.0000	0.0000	0.4251	0.4242

Appendix B: Logistic Likelihood

```r
loglike_func <- function(dat, betavec){

  dat = bc_trn

  # x matrix
  dat_temp <-
    dat %>%
    mutate(intercept = 1) %>%
    select(-diagnosis) %>%
    relocate(intercept)

  dat_x <-
    dat_temp %>%
    as.matrix() %>%
    unname()

  # pi vector
  u <- dat_x %*% betavec
  pi <- exp(u) / (1 + exp(u))

  # loglikelihood
  loglik <- sum(dat[,1]*u - log(1 + exp(u)))

  #gradient
  grad <- t(dat_x) %*% (dat[,1] - pi)

  # Hessian
  W <- diag(nrow(pi))
  diag(W) <- pi*(1 - pi)
  hess <- -(t(dat_x) %*% W %*% (dat_x))

  return(list(loglik = loglik, grad = grad, hess = hess))

}
```

Appendix C: Newton Raphson Implementation

```r
NewtonRaphson <- function(dat,  start, tol = 1e-8, maxiter = 200){

  i <- 0
  cur <- start
  stuff <- loglike_func(dat, cur)
  res <- c(i = 0, "loglik" = stuff$loglik,  "step" = 1, cur)

  prevloglik <- -Inf # to make sure it iterates

  while (i < maxiter && abs(stuff$loglik - prevloglik) > tol) {
    step <- 1
    i <- i + 1
    prevloglik <- stuff$loglik

    # check negative definite
    eigen_vals <- eigen(stuff$hess)

    if (max(eigen_vals$values) <= 0 ) { # check neg def, if not change
      hess <- stuff$hess
    } else { # if it is pos def then need to adjust
      hess <- stuff$hess - (max(eigen_vals$values) + 0.1)*diag(nrow(stuff$hess))
    }

    prev  <- cur
    cur   <- prev - step*(solve(stuff$hess) %*% stuff$grad)
    stuff <- loglike_func(dat, cur) # log-lik, gradient, Hessian

    # step halving
    while (stuff$loglik < prevloglik) {
      stuff <- loglike_func(dat, prev)
      step  <- step / 2 # this is where half steping happens
      cur   <- prev - step*(solve(stuff$hess) %*% stuff$grad)
      stuff <- loglike_func(dat, cur)
    }
    # add current values to results matrix
    res <- rbind(res, c(i, stuff$loglik, step, cur))
  }

  colnames(res) <- c("i", "loglik",  "step", "intercept", names(dat[,-1]))
  return(res)
}
```

Appendix D: Logistic Lasso Implementation

```r
# logistic function
logistic <- function(x) 1 / (1 + exp(-x))

# shrinkage function
S <- function(beta, gamma) {
  if(abs(beta) <= gamma) {
    0
  } else if(beta > 0) {
    beta - gamma
  } else {
    beta + gamma
  }
}

# probability adjustment function
p_adj <- function(p, epsilon) {
  if (p < epsilon) {
    0
  } else if(p > 1 - epsilon) {
    1
  } else {
    p
  }
}

# weight adjustment function
w_adj <- function(p, epsilon) {
  if ((p < epsilon) | (p > 1 - epsilon)) {
    epsilon
  } else {
    p * (1 - p)
  }
}

# executes logistic lasso regression via coordinate descent
logistic_lasso <- function(
  # a numeric design matrix or data frame with named columns
  inputs
  # a vector of outputs; we must have length(output) == nrow(inputs)
  , output
  # a vector of descending penalization factors, ideally on a logarithmic scale
  , lambda_vec
  # standardize inputs using scale
  , standardize   = TRUE
  # a buffer to prevent divergence when fitted probabilities approach 0 or 1
  , epsilon       = 10^-8
  # maximum number of updates to quadratic approximation of likelihood
  , outer_maxiter =  100
  # maximum number of cycles for coordinate descent given quadratic approximation
  , inner_maxiter = 1000
  # tolerance for convergence of coordinate descent
  , tolerance     = 10^-12
) {

  # standardize data unless otherwise specified
  if(standardize) {

    # format data
    X <- as.matrix(cbind(rep(1, nrow(inputs)), scale(inputs)))
    y <- output

  } else {

    # format data
    X <- as.matrix(cbind(rep(1, nrow(inputs)), inputs))
    y <- output

  }

  # initialize coefficients at origin
  beta    <- rep(0, ncol(X))
  beta_df <- NULL

  # begin lambda decrement
  for(lambda in lambda_vec) {

    outer_term <- 0
    outer_iter <- 1

    # update quadratic approximation, execute coordinate descent until convergence, repeat
    while(outer_term < 1) {

      # update quadratic approximation; i.e., taylor expand around current estimates
      p <- map_dbl(logistic(X %*% beta), p_adj, epsilon)
      w <- map_dbl(p, w_adj, epsilon)
      z <- X %*% beta + (y - p) / w

      inner_term <- 0
      inner_iter <- 1

      # given current quadratic approximation, execute coordinate descent
      while(inner_term < 1) {

        beta_old <- beta

        # execute a complete cycle of coordinate descent
        for(k in 1:ncol(X)) {

          # un-penalized coefficient update
          b_k_temp <- sum(w * (z - X[ , -k] %*% beta[-k]) * X[ , k]) / sum(w * X[ , k]^2)
          # shrinkage update
          b_k      <- S(b_k_temp, (k > 1) * lambda / mean(w * X[ , k]^2))
          # update coefficient vector
          beta[k] <- b_k
```

```r
        }

        inner_iter <- inner_iter + 1

        if(inner_iter == inner_maxiter | max(abs(beta - beta_old)) < tolerance) {

          inner_term <- 1

        }

      }

      outer_iter <- outer_iter + 1

      if(outer_iter == outer_maxiter | inner_iter == 2) {

        outer_term <- 1

      }

    }

    beta_df <- rbind(beta_df, t(beta))

  }

  # format data frame of coefficient estimates
  colnames(beta_df) <- c("intercept", names(inputs))
  beta_df <- as_tibble(beta_df)

  # extract number of variables selected for each lambda
  selected_vec <- apply(beta_df, 1, function(x) sum(x != 0) - 1)

  # output results
  list(lambda = lambda_vec, beta = beta_df, selected = selected_vec)
}
```

Appendix E: Defining Lambda Range

```
# lambda initialization function
lambda_init <- function(start, stop, step, func = identity){
  lambda_vec <- func(seq(start, stop, step))
  return(lambda_vec)
}


lambda_max <- max(t(scale(as.matrix(bc_trn[,-1]))) %*% bc_trn[,1]) / nrow(bc_trn[,-1])
lambda_list <- list(log(lambda_max), log(0.0001), -(log(lambda_max) - log(0.0001))/100, exp)

  lam_start <- lambda_list[[1]]
  lam_stop <- lambda_list[[2]]
  lam_step <- lambda_list[[3]]
  lam_func <- lambda_list[[4]]

init_lambda_vec <- lambda_init(lam_start, lam_stop, lam_step, lam_func)
```

Appendix F: Cross Validation Implementation

```
cv_jt <- function(k = 5, training, func, lam_start_stop_func, lambda_list){

  #### initializing lambda vector ####

  lam_start <- lambda_list[[1]]
  lam_stop <- lambda_list[[2]]
  lam_step <- lambda_list[[3]]
  lam_func <- lambda_list[[4]]

  lam_list <- tibble(
    lam_count = 0,
    lam_start = 0,
    lam_stop = 0,
    lam_step = 0
  )

  lam_count <- 0
  del_too_many_var <- 1
  out_res <- list()
  res <- list()

  while (del_too_many_var > 0) {

    lam_count <- lam_count + 1

    # saving lambda vector parameters
    cur_lam_list <- tibble(lam_count, lam_start, lam_stop, lam_step)

    lam_list <- bind_rows(lam_list, cur_lam_list)

    new_lambda_vec <- lambda_init(lam_start, lam_stop, lam_step, lam_func)

    lasso_list <- list()
    auc_list <- list()

    for (i in 1:k) {

      # this will identify the training set as not i
      trn_set =
        training %>%
        filter(fold_id != i) %>%
        select(-fold_id)

      # and this assigns i to be the test set
      tst_set =
        training %>%
        filter(fold_id == i) %>%
        select(-fold_id) %>%
        rename(y = diagnosis)

      # making matrices
      X_trn <- trn_set[,-1]
      Y_trn <- trn_set$diagnosis

      # lasso_list
      lasso_list <- func(inputs = X_trn, output = Y_trn, lambda_vec = new_lambda_vec)

      lasso_lambda <- lasso_list[[1]]
      lasso_beta <- lasso_list[[2]]
      lasso_selected <- tibble(selected_num = lasso_list[[3]])

      lasso_lam_bet <- cbind(lasso_lambda, lasso_beta) %>% as.matrix()

      trn_roc <- auc_calc_lasso(lasso_lam_bet, tst_set)
      auc_list <- bind_rows(auc_list, trn_roc)

    }

    auc_res <-
      auc_list %>%
      group_by(lambda) %>%
      summarise(mean_auc = mean(auc_vals))

    res[[lam_count]] <- bind_cols(auc_res, lasso_selected)

    res[[lam_count]] <- res[[lam_count]] %>%
      mutate(num_dropped_vars = selected_num - lag(selected_num, 1))

    del_too_many_var <- sum(na.omit(res$num_dropped_vars) > 1)

   if (del_too_many_var > 0) {
     #### performing grid search ####
    max_auc_lam <- res %>% filter(mean_auc == max(mean_auc)) %>% pull(lambda) %>% mean()
    lam_start <- lam_start_stop_func(max_auc_lam) + 2*abs(lam_step)
    lam_stop <- lam_start_stop_func(max_auc_lam) - 2*abs(lam_step)
    lam_step <- sign(lam_step)*(lam_start - lam_stop)/length(new_lambda_vec)
}
  }

  # creating dataframe to show lambda values and corresponding mean AUC
  out_res[[1]] <- res
  out_res[[2]] <- lam_list
  out_res[[3]] <- auc_list

  return(out_res)
}
```

Appendix G: Grid Search

```
# checking to see if more than one variable is selected between lambdas at any point
del_too_many_var <- sum(na.omit(res$num_dropped_vars) > 1)

# if more than one variable is selected/dropped, adjust range and step size
if (del_too_many_var > 0) {
    max_auc_lam <- res %>% filter(mean_auc == max(mean_auc)) %>% pull(lambda) %>% mean()
    lam_start <- lam_start_stop_func(max_auc_lam) + 2*abs(lam_step)
    lam_stop <- lam_start_stop_func(max_auc_lam) - 2*abs(lam_step)
    lam_step <- sign(lam_step)*(lam_start - lam_stop)/length(new_lambda_vec)
}
```

Appendix H: Mathematical Appendix

In this appendix entry, we show that the log-likelihood function f can be approximated via the quadratic function $g : \mathbb{R}^p \to \mathbb{R}$ given by

$$g(\boldsymbol{\beta}) = -\frac{1}{2}\sum_{i=1}^{n} w_i \left(z_i - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \eta(\boldsymbol{\alpha}).$$

Fix vectors $\boldsymbol{\beta} \in \mathbb{R}^p$. For each observation $i \in \{1, \ldots, n\}$, we first define

$$\xi_i \equiv \sum_{j=1}^{p} x_{ij}(\beta_j - \alpha_j), \quad \text{such that}$$

$$\begin{aligned}
\frac{y_i - \pi_i}{w_i} - \xi_i &= \frac{y_i - \pi_i}{w_i} - \sum_{j=1}^{p} x_{ij}(\beta_j - \alpha_j) \\
&= \left(\frac{y_i - \pi_i}{w_i} + \sum_{j=1}^{p} x_{ij}\alpha_j \right) - \sum_{j=1}^{p} x_{ij}\beta_j \\
&= z_i - \sum_{j=1}^{p} x_{ij}\beta_j .
\end{aligned}$$

We proceed with a series of computations. Observe that

$$\begin{aligned}
\Big(\mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\alpha})\Big)^t (\mathbf{Y} - \boldsymbol{\pi}) &= \left(\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_1 - \alpha_1 \\ \vdots \\ \beta_p - \alpha_p \end{pmatrix} \right)^t \begin{pmatrix} y_1 - \pi_1 \\ \vdots \\ y_n - \pi_n \end{pmatrix} \\
&= \begin{pmatrix} \sum_{j=1}^{p} x_{1j}(\beta_j - \alpha_j) \\ \vdots \\ \sum_{j=1}^{p} x_{nj}(\beta_j - \alpha_j) \end{pmatrix}^t \begin{pmatrix} y_1 - \pi_1 \\ \vdots \\ y_n - \pi_n \end{pmatrix} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{p} x_{ij}(\beta_j - \alpha_j)(y_i - \pi_i) \\
&= \sum_{i=1}^{n} (y_i - \pi_i)\xi_i .
\end{aligned}$$

Because W is a diagonal matrix with non-negative elements, $\sqrt{\mathbf{W}}$ is well defined. As such, we can compute

$$\begin{aligned}
\sqrt{\mathbf{W}}\mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\alpha}) &= \begin{pmatrix} \sqrt{w_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{w_n} \end{pmatrix} \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_1 - \alpha_1 \\ \vdots \\ \beta_p - \alpha_p \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{w_1}x_{11} & \cdots & \sqrt{w_1}x_{1p} \\ \vdots & \ddots & \vdots \\ \sqrt{w_n}x_{n1} & \cdots & \sqrt{w_n}x_{np} \end{pmatrix} \begin{pmatrix} \beta_1 - \alpha_1 \\ \vdots \\ \beta_p - \alpha_p \end{pmatrix} \\
&= \begin{pmatrix} \sum_{j=1}^{p} \sqrt{w_1}x_{1j}(\beta_j - \alpha_j) \\ \vdots \\ \sum_{j=1}^{p} \sqrt{w_n}x_{1n}(\beta_j - \alpha_j) \end{pmatrix} .
\end{aligned}$$

Given the above computation, we further obtain

$$\begin{aligned}
\left(\sqrt{\mathbf{W}}\mathbf{X}(\boldsymbol{\beta}-\boldsymbol{\alpha})\right)^t\left(\sqrt{\mathbf{W}}\mathbf{X}(\boldsymbol{\beta}-\boldsymbol{\alpha})\right) &= \sum_{i=1}^n\left(\sum_{j=1}^p\sqrt{w_i}x_{ij}(\beta_j-\alpha_j)\right)^2\\
&= \sum_{i=1}^n w_i\left(\sum_{j=1}^p x_{ij}(\beta_j-\alpha_j)\right)^2\\
&= \sum_{i=1}^n w_i\xi_i^2.
\end{aligned}$$

Finally, we consider the second-order Taylor expansion of f centered at $\boldsymbol{\alpha}$. In particular, given all of the above computations, and by completing the square, we obtain

$$\begin{aligned}
f(\boldsymbol{\beta}) &\approx f(\boldsymbol{\alpha})+(\boldsymbol{\beta}-\boldsymbol{\alpha})^t\nabla f(\boldsymbol{\alpha})+\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\alpha})^t\nabla^2 f(\boldsymbol{\alpha})(\boldsymbol{\beta}-\boldsymbol{\alpha})\\
&= f(\boldsymbol{\alpha})+(\boldsymbol{\beta}-\boldsymbol{\alpha})^t\Big(\mathbf{X}^t(\mathbf{Y}-\boldsymbol{\pi})\Big)+\frac{1}{2}(\boldsymbol{\beta}-\boldsymbol{\alpha})^t\Big(-\mathbf{X}^t\mathbf{W}\mathbf{X}\Big)(\boldsymbol{\beta}-\boldsymbol{\alpha})\\
&= f(\boldsymbol{\alpha})+\Big(\mathbf{X}(\boldsymbol{\beta}-\boldsymbol{\alpha})\Big)^t(\mathbf{Y}-\boldsymbol{\pi})-\frac{1}{2}\Big(\sqrt{\mathbf{W}}\mathbf{X}(\boldsymbol{\beta}-\boldsymbol{\alpha})\Big)^t\Big(\sqrt{\mathbf{W}}\mathbf{X}(\boldsymbol{\beta}-\boldsymbol{\alpha})\Big)\\
&= f(\boldsymbol{\alpha})+\sum_{i=1}^n(y_i-\pi_i)\xi_i-\frac{1}{2}\sum_{i=1}^n w_i\xi_i^2\\
&= f(\boldsymbol{\alpha})-\frac{1}{2}\sum_{i=1}^n w_i\left(\xi_i^2-2\left(\frac{y_i-\pi_i}{w_i}\right)\xi_i\right)\\
&= \eta(\boldsymbol{\alpha})-\frac{1}{2}\sum_{i=1}^n w_i\left(\xi_i-\frac{y_i-\pi_i}{w_i}\right)^2\\
&= \eta(\boldsymbol{\alpha})-\frac{1}{2}\sum_{i=1}^n w_i\left(z_i-\sum_{j=1}^p x_{ij}\beta_j\right)^2,
\end{aligned}$$

where $\eta:\mathbb{R}^p\to\mathbb{R}$ is some function of $\boldsymbol{\alpha}$ that does not depend on $\boldsymbol{\beta}$.