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

we will write an abstract eventually.

1. Introduction

1.1. Overview

As breast cancer is one of the most common kinds of cancer in the United States, great efforts have been made to aid in early and accurate detection. Improvements in tumor imaging technology used in screening procedures have allowed us access to more data than ever before, ideally to construct better ways to evaluate disease severity. However, data does not always equate to 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

In this paper we investigate two questions: does having more data always correspond to an advantage in diagnosis prediction? Can we reduce the amount of data we need to collect while maintaining (or increasing) predictive power?

To this end, we use a breast cancer imaging dataset (detailed below) to fit two different models with the goal of predicting patient diagnosis outcomes. The first is a generalized linear-logistic regression model with a full set of predictors (i.e., "full model"), calculated using a Newton-Raphson optimization algorithm. The second is a penalized logistic-LASSO model (i.e., "optimal model"), which is capable of reducing the number of selected predictors from our dataset. This is implemented using a path-wise coordinate-descent optimization algorithm, and we utilize 5-fold cross validation to obtain the optimal λ penalization term. 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 data set of interest contains 569 rows and 32 columns related to breast tissue imaging with each entry 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 removed from our dataset. The other 30 columns correspond to summary measures (mean, standard deviation, and maximum) of variables such as radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry, and fractal dimension. This dataset does not contain any missing values.

In a quick exploration of the data, we find many of the predictors are highly correlated with one another. In **Figure 1**, we see a heat map correlation plot where several variables have dark blue coloring representing strong relationships. High collinearity between predictors can cause major issues in regression methods, particularly with 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 value of 0.998 (maximum of 1). In the graph there are 21 combinations of variables that achieve a correlation greater than 0.90, which is a cause for concern in this context. We can break these 21 pairings into equivalence classes for further inspection.

The first grouping that are all mutually correlated is {`area_mean`, `area_worst`, `perimeter_mean`, `perimeter_worst`, `radius_mean`, `radius_worst`}. This grouping represents 15 of the correlation pairs in **Figure 2**. Mathematically, if we consider the equivalence classes of variables that are highly correlated, these six variables would belong to the same equivalence class. To identify the best proxy for this grouping we look at the highest mean correlation which turns out to be `radius_worst`. The next grouping of correlated variables is {`radius_se`, `perimeter_se`, `area_se`}. The best representative will be `radius_se`. Next we can group {`concavity_mean`, `concave_points_worst`, `concave_points_mean`} together, and we find the best proxy variable is `concave_points_mean`. Finally, {`texture_mean`, `texture_worst`} is our last grouping with `texture_mean` being the variable saved. Thus from all the grouping and saving only the best proxy we will be removing 10 variables leaving 20 in our dataset. All the predictors used can be seen in Table 1.

In Table 1 we see that there are 357 benign (B) cases and 212 malignant (M) cases. To implement both the full and optimal model the data set will be split into train and test sets using an 80-20 split. The data is standardized before fitting both of our models to help with comparability. For LASSO regression in particular, it is best practice to center and scale data. The ℓ_1-norm penalization in LASSO regression will unequally penalize coefficient estimates if the covariates are of different magnitudes or scales. Therefore, standardizing our data ensures equal weighting and penalization.

Figure 1: Correlation Heat Plot of all Covariates

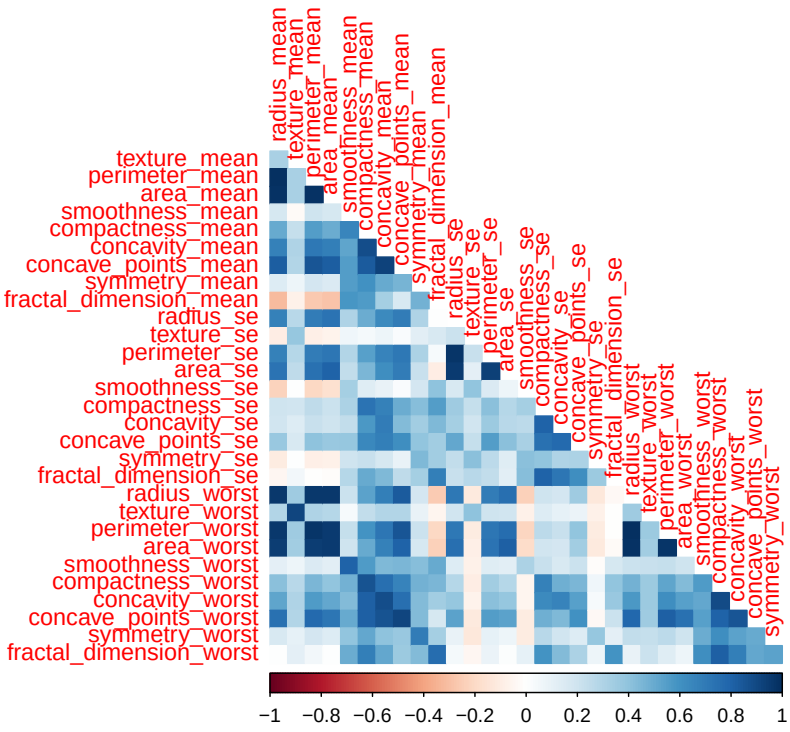


Figure 2: Ranked Cross-Correlations

25 most relevant

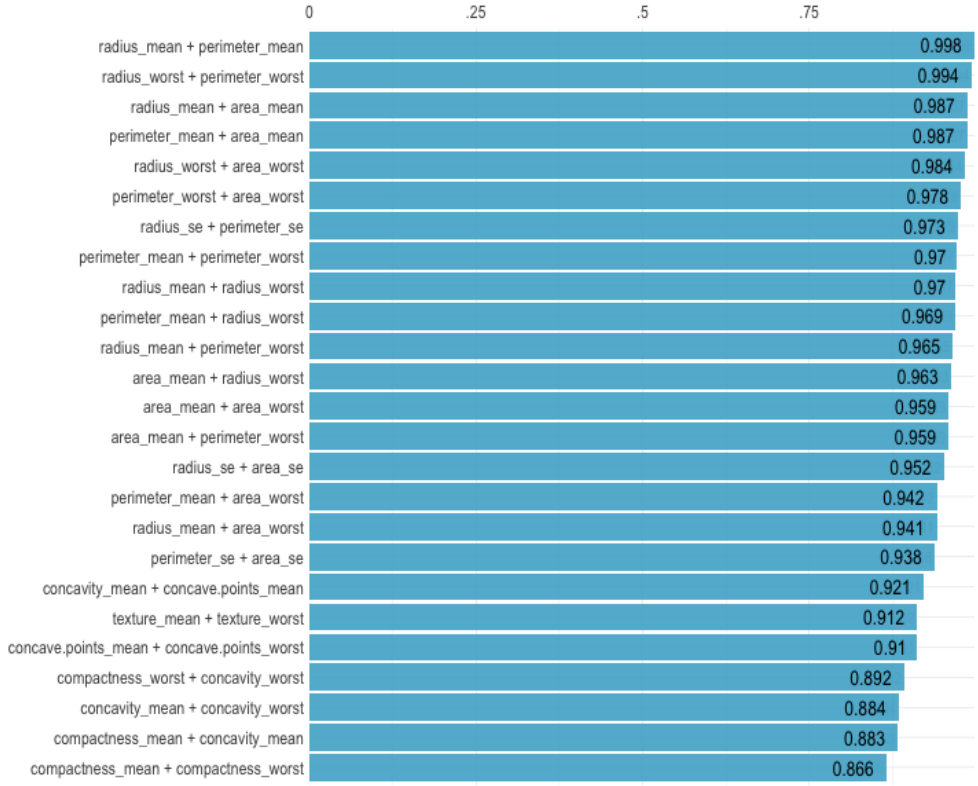


Table 1: Patient Characteristics

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

To implement the Newton-Raphson Algorithm we need to first examine the likelihood function and derive the gradient and Hessian matrix. First, we will look at the likelihood function for our data, which has a single binary response and p numerical explanatory variables. We know that

$$\pi_i = P(Y_i = 1|x_{i,1}, \dots x_{i,p}) = \frac{e^{\beta_0+\beta_1 x_{i,1}+\dots\beta_p x_{i,p}}}{1+e^{\beta_0+\beta_1 x_{i,1}+\dots\beta_p x_{i,p}}} = \frac{e^{\beta_0+\sum_{j=1}^{p}\beta_j x_{i,j}}}{1+e^{\beta_0+\sum_{j=1}^{p}\beta_j x_{i,j}}}$$

where $\mathbf{X_i}$ represents the i-th observation of all p of our predictor variables. For our data, the likelihood function is given by

$$L(\mathbf{X}|\beta) = \prod_{i=1}^{n}\left[\pi_i^{y_i}(1-\pi_i)^{1-y_i}\right]$$

where y_i represents out binary outcome for the i-th individual. Finding the log-likelihood we have

$$l(\mathbf{X}|\vec{\beta}) = \sum_{i=1}^{n}\left[y_i\left(\beta_0+\sum_{j=1}^{p}\beta_j x_{i,j}\right) - \log\left(1+\exp\left(\beta_0+\sum_{j=1}^{p}\beta_j x_{i,j}\right)\right)\right].$$

The gradient is the partial derivative of the log-likelihood with respect to each β_j variable. Observe

$$\nabla l(\mathbf{X}|\vec{\beta}) = \left[\sum_{i=1}^{n} y_i - \pi_i \quad \sum_{i=1}^{n} x_{i,1}(y_i-\pi_i) \quad \dots \quad \sum_{i=1}^{n} x_{i,20}(y_i-\pi_i)\right]^T_{(1\times(p+1))}.$$

Finally, using the gradient we can derive our hessian matrix. Note that due to the 20 predictor variables the hessian will be a $p+1$ by $p+1$ matrix.

$$\begin{aligned}\nabla^2 l(\mathbf{X}|\vec{\beta}) &= -\sum_{i=1}^{n}\begin{pmatrix}1\\X\end{pmatrix}\begin{pmatrix}1 & X\end{pmatrix}\pi_i(1-\pi_i)\\ &= -\begin{pmatrix}1 & X\end{pmatrix} diag(\pi_i(1-\pi_i))\begin{pmatrix}1\\X\end{pmatrix}\end{aligned}$$

Where $X = (x_{i,1}, \dots, x_{i,p})$. Note that this matrix will always be negative definite at all parameters making this a well-behaved problem. Plus, the logistic regression objective function is globally concave and hence there exists one global optimum. Using the likelihood, gradient, and hessian, the Newton-Raphson algorithm implemented in R can be seen in **Appendix #**. Two modifications have been made to the algorithm. The first is to control ascent direction by checking that the eigenvalues are negative, representing a negative definite matrix. The second modification is to include half-stepping to increase the speed of the algorithm.

Note that major problems arise when highly correlated variables are included in the model. To be specific, as the Newton-Raphson algorithm proceeds, the absolute values of β_i continue to increase. This causes some of the elements in the probability vector to be very close to 1, leading some of the elements in log(1-p) vector to be negative infinity and hence the next log likelihood to diverge to negative infinity. As a result, the Newton-Raphson algorithm fails to reach the convergence of maximum likelihood estimation. Without excluding the 10 variables the Newton-Raphson method would not converge due to multicollinearity issues.

2.3 Logistic LASSO Algorithm

Lemma 1. Consider the optimization problem

$$\min_{x\in\mathbb{R}}\left\{\frac{1}{2}(x-b)^2 + c|x|\right\}$$

for $b \in \mathbb{R}$ and $c \in \mathbb{R}_{++}$. It follows that the minimizer is given by

$$\hat{x} = S(b, c),$$

where S is the soft-thresholding operator.

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

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

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

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

Proposition. By Lemma 1 and Lemma 3,

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

More info from the pres: NEed to be editted to paragraph form

For vector $\boldsymbol{\alpha} \in \mathbb{R}^{p+1}$, define $g : \mathbb{R}^{p+1} \to \mathbb{R}$ to be

$$g(\boldsymbol{\beta}) \equiv -\frac{1}{2n} \sum_{i=1}^{n} w_i (z_i - \mathbf{X}_i^t \boldsymbol{\beta})^2 + O(\boldsymbol{\alpha}),$$

the Taylor expansion of our log-likelihood centered around $\boldsymbol{\alpha}$, where

$$\begin{aligned} z_i &\equiv \mathbf{X}_i^t \boldsymbol{\alpha} + \frac{y_i - \pi_i}{w_i}, && \text{(effective response)} \\ w_i &\equiv \pi_i (1 - \pi_i), \text{ and} && \text{(effective weights)} \\ \pi_i &\equiv \frac{e^{\mathbf{X}_i^t \boldsymbol{\alpha}}}{1 + e^{\mathbf{X}_i^t \boldsymbol{\alpha}}} \end{aligned}$$

for $i \in \{1, \ldots, n\}$.

It follows that for any $\lambda \in \mathbb{R}_+$,

$$\arg\min_{\beta_k \in \mathbb{R}} \left\{ g(\boldsymbol{\beta}) + \lambda \sum_{j=1}^{p} |\beta_j| \right\} = S \left(\hat{\beta}_k, \lambda_k \right), \text{ where}$$

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

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

and S is the soft-thresholding (or *shrinkage*) function. This is analogous to a penalized, weighted Gaussian regression.

Our coordinate descent algorithm proceeds as follows.

- Outer Loop: Decrement over $\lambda \in (\lambda_{\max}, \ldots, \lambda_{\min})$ where $\lambda_{\max} = \frac{max|X^T y|}{n}$
- 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.

To be specific, an outer loop is created for every new value of λ and then we use coordinate descent to solve the penalized weighted least-squares problem. Namely, z_i, w_i, and π_i are updated at the current parameters for every outer loop to update the quadratic approximation $g(\beta)$ and then keep updating parameters until it satisfies the threshold set beforehand.

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.

2.4 Five-fold Cross Validation

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

First, we need to 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. While we could make λ_{max} infinitely large, we will define the maximum value as 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. Likewise, while we could make λ_{min} infinitely small (but greater than 0), we will select the largest value that produces the full model, which is identical to the model produced by the Newton Raphson algorithm we have previously defined. As recommended by Friedman et. al [2], 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 a few times – ultimately, we saw that a value of 0.0001 $\left(\approx \frac{\lambda_{max}}{4000}\right)$ would be a suffice as a sufficiently small penalty factor that would 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} a log 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.

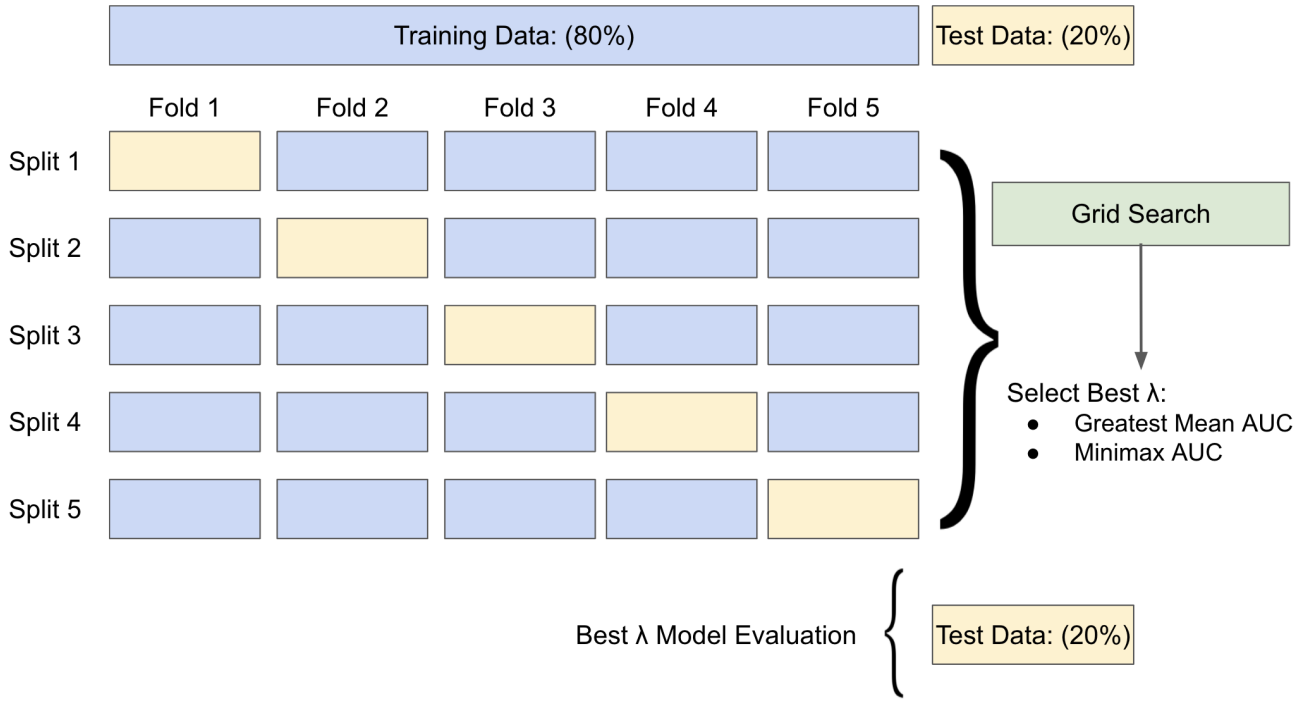
To select our optimal tuning parameter, we calculated two AUC statistics for each λ in our five-fold cross validation: mean AUC and Minimax AUC. The mean AUC is a simple average of the five AUC value produced

from the cross validation process for each λ in our range. The Minimax AUC selects the minimum of the maximum $1 - AUC$ values over all 100 λ values to account for the scenario where we aim to minimize the possible loss for the worst case scenario. Then the model with minimum-worst AUC is selected as our optimal model. The greatest mean AUC and Minimax AUC optimal models are compared against the full model below.

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 logorithmic-exponentiation process we used to construct our initial range of λ values. Using the same (unexponentiated) 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.

Figure 3: Cross Validation Procedure



2.5 Final Model Evaulation

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.

This section might not be needed. I wanted to highlight Charly's question from the pres...

3. Results

3.1 Cross-Validation Results

Evaluating the cross validation results, we can 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** depicting the largest AUC and corresponding lambda values by the vertical dotted line. After the vertical dotted line the AUC values decrease thus indicating one maximum values exists in our range.

We can also evaluate our cross validation results by looking at the beta estimates for each corresponding λ value. This is seen in **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. The vertical lines depict the optimal lambda value chosen for each of our selection methods. 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 4: Cross Validation Results: Selecting Best Lambda

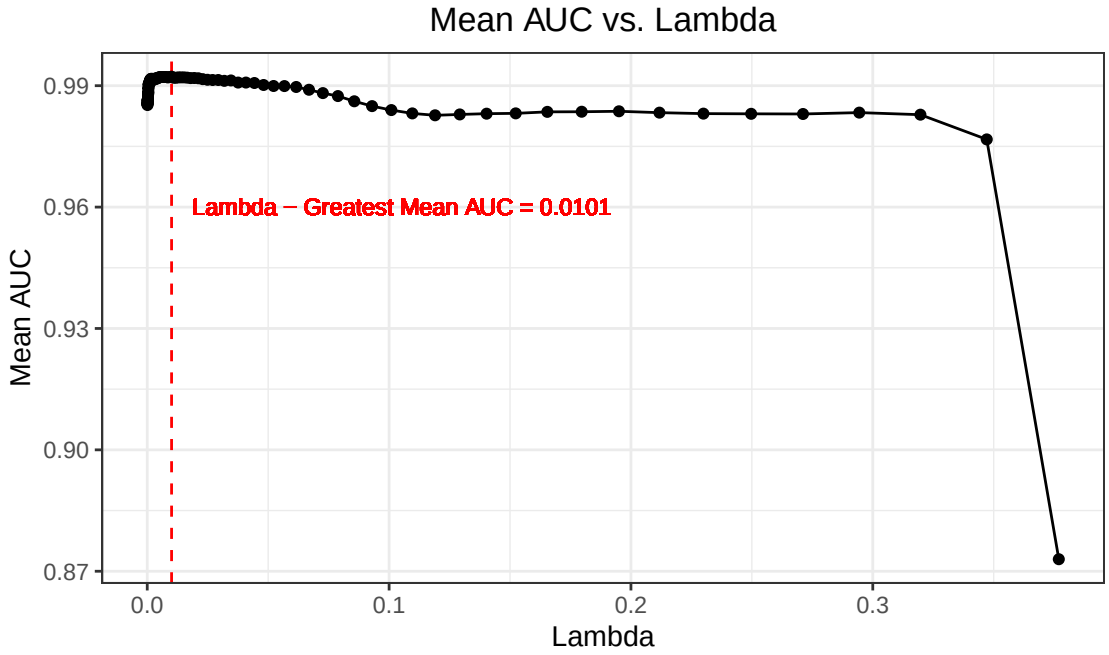
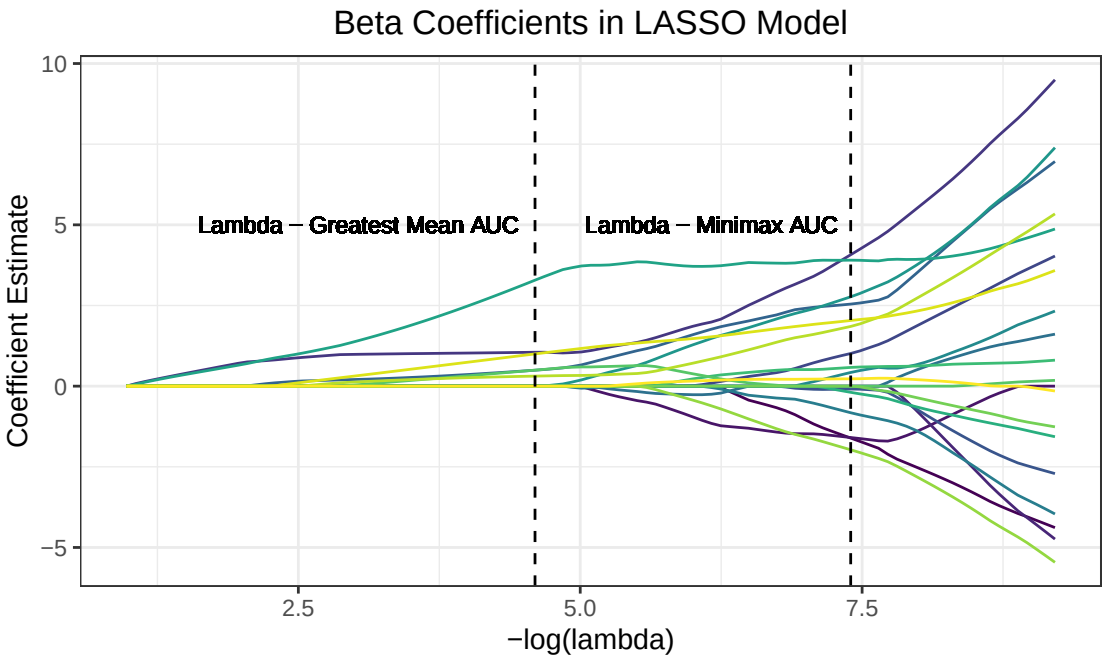


Figure 5: Cross Validation Results: LASSO Coefficients



3.2 Model Validation Results

To compare the beta estimates in our full model and two optimal models, we can look at **Table 2**. Please note that we standardized our data so care needs to be taken before interpreting these values in the context of the project. Also note that care needs to be taken when interpreting the beta values for the optimal model because of standardization and the lambda penalty term. In **Table 2** we see that the greatest mean AUC has the most beta values equal to zero. **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.

While the AUC value is a good indicator of overall classification performance, we are still confronted with the trade-off between sensitivity and specificity. Imagining is typically the first line of defense for diagnosing cancers, so optimizing sensitivity is usually of great interest. In **Table 3** we have displayed the largest specificity value achieved while realizing sensitivity to equal 1 (all positive individuals correctly identified as positive). Namely, we are looking for a model that sacrifices the least value of specificity while still achieving 1 sensitivity. We can see in **Table 3**, the greatest mean AUC lasso model has the highest specificity, 69.23%. Though minimax AUC Lasso model is the minimum of the worst AUC model, it still achieves much larger specificity than the full model, 55.38%. Namely, if we choose to a lasso model, it is still expected to manage to achieve 55.38% specificity while maintaining 1 sensitivity for the worst case scenario. It is to be observed that full model has the lowest sensitivity, 24.62%, and hence needs to sacrificice specificity more than 75% to achieve 1 sensitivity. This sensitivity and specificity trade-off can also be seen in our ROC curves displayed in **Figure 6**.

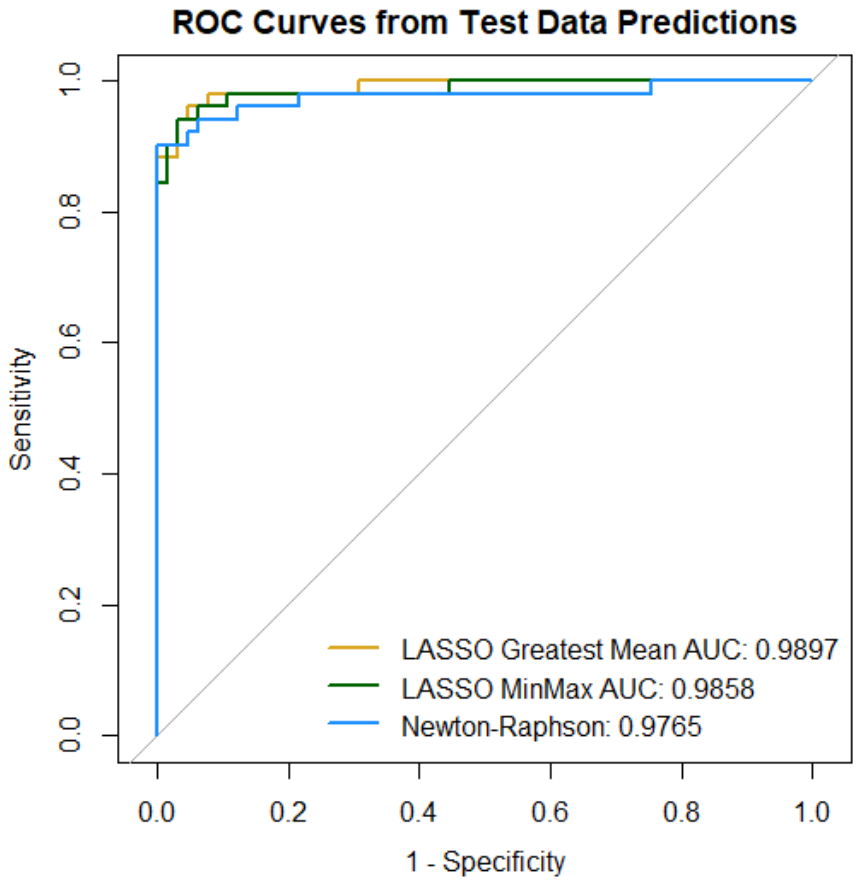
Table 2: Beta Coefficients Comparing Full and Optimal Models

	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

Table 3: Model Summary Comparing Full and Optimal Models

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

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.

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 for further study. Besides AUC as a metric of model performance, the accuracy rate of prediction can be also utilized as a means of measuring model performance 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] 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.

Coding Appendices

Appendix A: Logistic Likelihood

```
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 B: Newton Raphson Implementation

```
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 C: Logistic Lasso Implementation

```
# 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
```

```
        }

        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 D: 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 E: 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)
}
```