Breast Cancer Diagnosis Prediction Project

Samuel Darteh, Data Science MS

MA 5790

December 2023

Abstract

Breast cancer, undoubtedly, is a significant global health concern; it is today's commonest cancer amongst women, and the second to lung cancer in mortality rate. Early and accurate diagnosis is a critical resolve for effective prevention, treatment, and improved patient outcomes. This study explores the application of predictive modeling techniques to enhance breast cancer diagnosis. A comprehensive dataset comprising clinical imaging and genetic features from breast mass sampling is used to train and evaluate predictive models—classification models. Machine learning algorithms deployed in both linear and non-linear classification models including neural networks, regularized discriminant analysis, and penalized models are utilized to analyze the dataset and predict the likelihood of breast cancer occurrence. The performance of these models, in both training and test, is assessed based on Kappa statistic, accuracy rate, and ROC's AUC (a balance for sensitivity and specificity). Lastly, feature importance analysis is conducted to identify predictors that are most critical to the optimal predictive model's outcomes.

Table of Contents

1.0 Background

Breast cancer is today the commonest cancer amongst women and the second deadliest cancer in the world, next to lung cancer.[1] While this is notably the concern for women, breast cancer equally affects men. Governments and health sponsors all over the world have taken keen interest in the campaign for regular breast checkups and early detection of breast cancer signs.

1.1 Goal of Study

The goal of the study is to create a model to predict the possible diagnosis of breast cancer—whether benign (healthy) or malignant (cancerous). The findings of the study will contribute valuable insights toward the development of efficient and accurate diagnostic tools for breast cancer, and foster advancements in personalized medicine and general patient care.

1.2 Data Source

Data used for the study is the Breast Cancer Wisconsin (Diagnostic) dataset from the University of California (Irvine).[2] Features were computed from a digitized image of a fine needle aspirate (FNA), a medical procedure of breast mass sampling, and relevant features selected. The mean, standard error, and worst/largest values were computed for ten (10) key features and data recorded for all persons sampled.

[1] See url: https://seer.cancer.gov/statfacts/html/common.html#:~:text=Statistics%20at%20a%20Glance&text=Breast%2C%20lung%20and%20bronchus%2C%20prostate,nearly%2050%25%20of%20all%20deaths

[2] See url: http://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic

2.0 Exploring the Data – Structure and Definitions

The breast cancer dataset is comprised of 569 observations and 30 predictors. The data has one categorical response variable, "diagnosis" with the class values: "Benign" or "Malignant". The study is therefore considered as a classification and unsupervised problem. The class distribution is unbalanced with 357 (63%) cases for "Benign" and 212 (37%) cases for "Malignant".

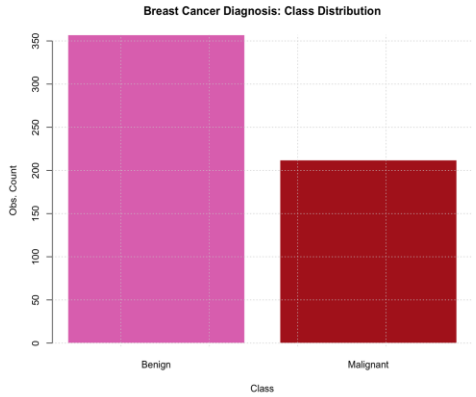


Fig 2.1. Unbalanced distribution of the classes: Benign and Malignant

All 30 predictors are continuous; there are no categorical predictors. Values for *mean, standard error and worst/largest* have been computed for the ten (10) selected features used in the scientific procedure.[3] The table below highlights the 10 selected features with brief description, as well as the diagnosis and id columns.

Predictor	Description
id	Identifier for each sample
diagnosis	Response—Benign (B) or Malignant (M)
radius	**Mean distances from center to points on the perimeter**
texture	**Standard deviation of gray-scale values**
perimeter	**Perimeter of the tumor cells**

[3] Section: "1.0 Background" has details about the data source.

area	**Area of the tumor cells**
smoothness	**Smoothness of the tumor cells**
compactness	**Local variation in radius lengths**
concavity	**Perimeter^2 / area - 1.0**
concave points	**Severity of concave portions of the contour**
symmetry	**Number of concave portions of the contour**
fractal dimension	**Coastline approximation- 1**

The thirty (30) predictors—used in the study—have been outlined below:

radius_mean	texture_mean	perimeter_mean
area_mean	smoothness_mean	compactness_mean
concavity_mean	concave_points_mean	symmetry_mean
fractal_dimension_mean	radius_error	texture_error
perimeter_error	area_error	smoothness_error
compactness_error	concavity_error	concave_points_error
symmetry_error	fractal_dimension_error	radius_worst
texture_worst	perimeter_worst	area_worst
smoothness_worst	compactness_worst	concavity_worst
concave_points_worst	symmetry_worst	fractal_dimension_worst

3.0 Data Preprocessing

The relationship between the predictors were analyzed to guide accurate predictions in the response variable. Preprocessing the data before building models certainly has a significant impact on model performance and analysis. The preprocessing steps looked at possible data anomalies existing in data and handled them using appropriate data transformation techniques. Removal of degenerate (near zero variance) predictors and adding of dummy variables were not applied as dataset had no categorical data.

3.1 Data Anomalies

a. Missing Data, Duplicates and Negative Values

There were no missing values, duplicates, or negative values in the dataset. Since no data was missing, data imputation was not considered.

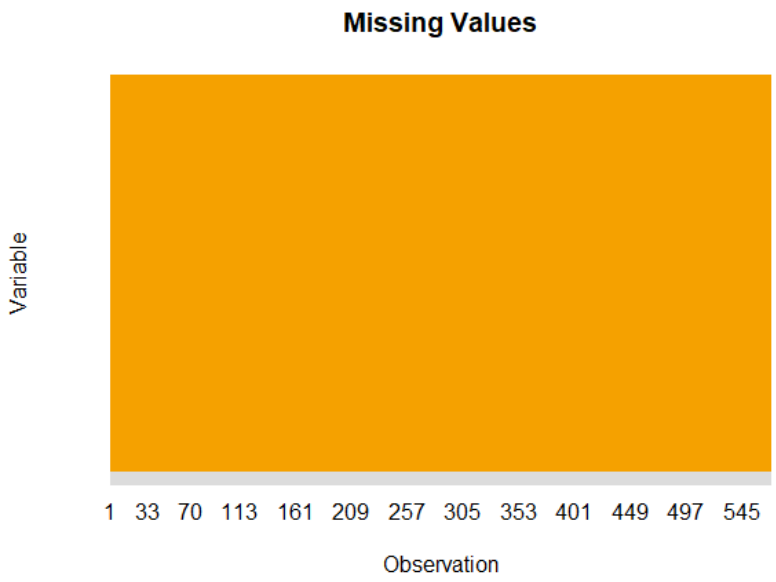
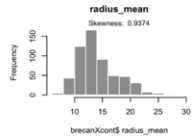
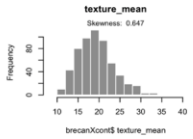


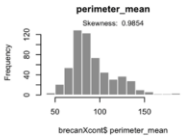
Fig 3.1. Missing values plot

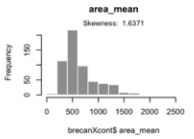
b. Skewness

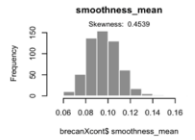
The data showed a strong presence of skewness: 22 of the 30 predictors were found to be heavily skewed to the right. The histogram plot below shows the skewness in the predictors.

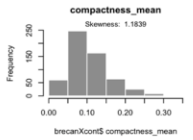

radius_mean
Skewness: 0.9374
Frequency
brecanXcont$ radius_mean

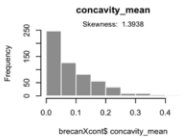

texture_mean
Skewness: 0.647
Frequency
brecanXcont$ texture_mean

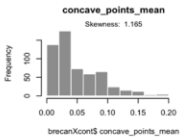

perimeter_mean
Skewness: 0.9854
Frequency
brecanXcont$ perimeter_mean

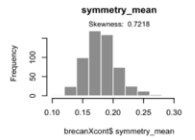

area_mean
Skewness: 1.6371
Frequency
brecanXcont$ area_mean

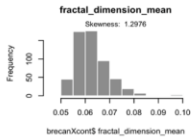

smoothness_mean
Skewness: 0.4539
Frequency
brecanXcont$ smoothness_mean

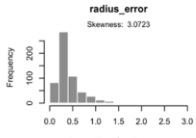

compactness_mean
Skewness: 1.1839
Frequency
brecanXcont$ compactness_mean

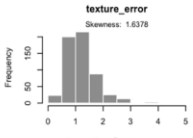

concavity_mean
Skewness: 1.3938
Frequency
brecanXcont$ concavity_mean

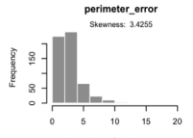

concave_points_mean
Skewness: 1.165
Frequency
brecanXcont$ concave_points_mean

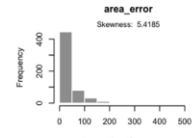

symmetry_mean
Skewness: 0.7218
Frequency
brecanXcont$ symmetry_mean


fractal_dimension_mean
Skewness: 1.2976
Frequency
brecanXcont$ fractal_dimension_mean

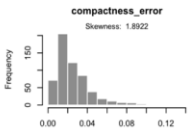

radius_error
Skewness: 3.0723
Frequency
brecanXcont$ radius_error

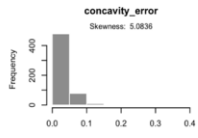

texture_error
Skewness: 1.6378
Frequency
brecanXcont$ texture_error

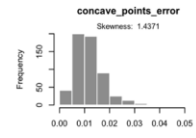

perimeter_error
Skewness: 3.4255
Frequency
brecanXcont$ perimeter_error

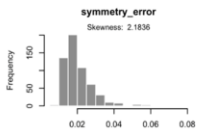

area_error
Skewness: 5.4185
Frequency
brecanXcont$ area_error

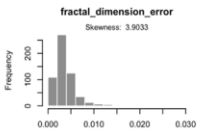

smoothness_error
Skewness: 2.3023
Frequency
brecanXcont$ smoothness_error

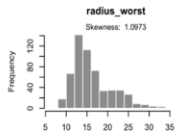

compactness_error
Skewness: 1.8922
Frequency
brecanXcont$ compactness_error


concavity_error
Skewness: 5.0836
Frequency
brecanXcont$ concavity_error

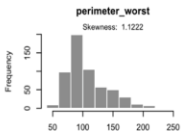

concave_points_error
Skewness: 1.4371
Frequency
brecanXcont$ concave_points_error

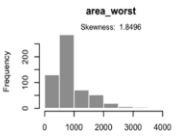

symmetry_error
Skewness: 2.1836
Frequency
brecanXcont$ symmetry_error

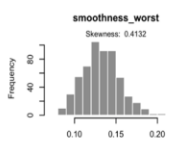

fractal_dimension_error
Skewness: 3.9033
Frequency
brecanXcont$ fractal_dimension_error

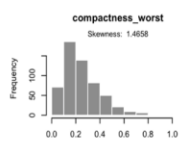

radius_worst
Skewness: 1.0973
Frequency
brecanXcont$ radius_worst

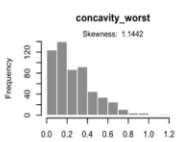

texture_worst
Skewness: 0.4957
Frequency
brecanXcont$ texture_worst

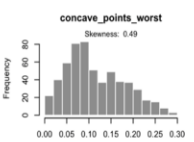

perimeter_worst
Skewness: 1.1222
Frequency
brecanXcont$ perimeter_worst


area_worst
Skewness: 1.8496
Frequency
brecanXcont$ area_worst

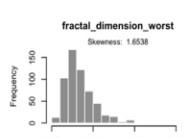

smoothness_worst
Skewness: 0.4132
Frequency
brecanXcont$ smoothness_worst


compactness_worst
Skewness: 1.4658
Frequency
brecanXcont$ compactness_worst


concavity_worst
Skewness: 1.1442
Frequency
brecanXcont$ concavity_worst


concave_points_worst
Skewness: 0.49
Frequency
brecanXcont$ concave_points_worst


symmetry_worst
Skewness: 1.4264
Frequency
brecanXcont$ symmetry_worst


fractal_dimension_worst
Skewness: 1.6538
Frequency
brecanXcont$ fractal_dimension_worst

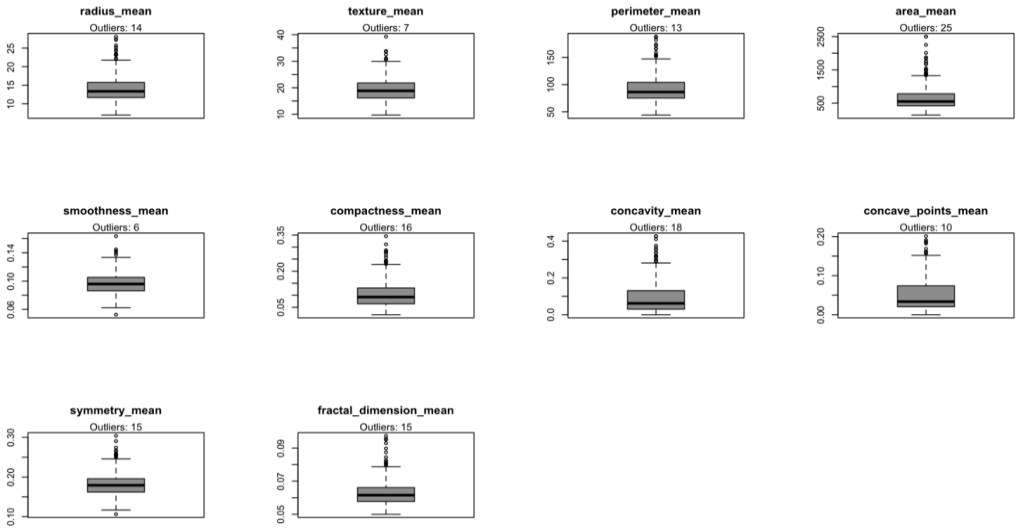
The table below contains a summary of the skewness values and interpretation for the 30 predictors.

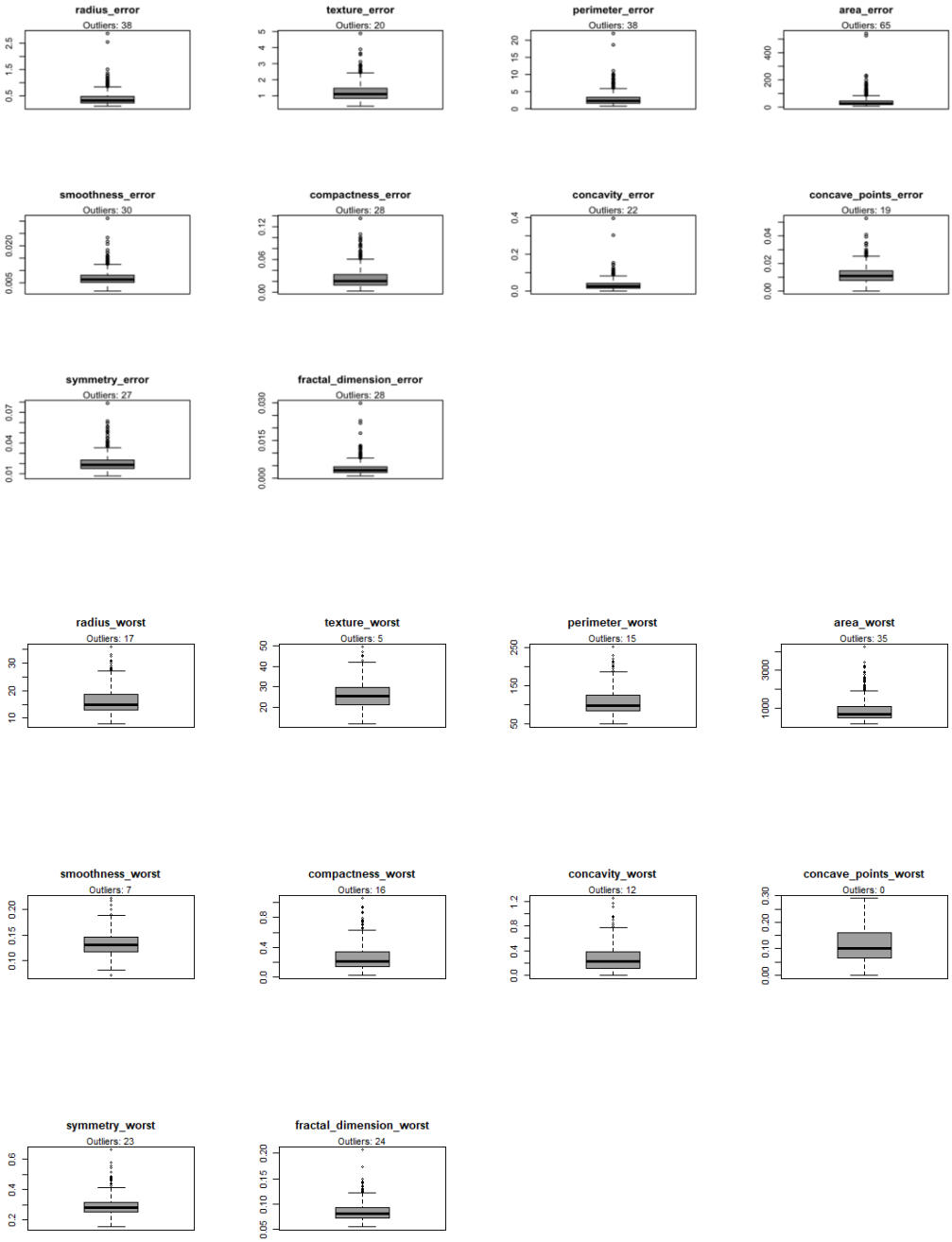
Predictor	Skewness	Interpretation
radius_mean	0.9374168	Moderately right skewed
texture_mean	0.6470241	Moderately right skewed
perimeter_mean	0.9854334	Moderately right skewed
area_mean	1.6370654	Heavily right skewed
smoothness_mean	0.4539207	Approximately symmetric
compactness_mean	1.1838556	Heavily right skewed
concavity_mean	1.3938008	Heavily right skewed
concave_points_mean	1.1650124	Heavily right skewed
symmetry_mean	0.7217877	Moderately right skewed
fractal_dimension_mean	1.2976191	Heavily right skewed
radius_error	3.0723468	Heavily right skewed
texture_error	1.6377733	Heavily right skewed
perimeter_error	3.4254803	Heavily right skewed
area_error	5.4185001	Heavily right skewed
smoothness_error	2.3022616	Heavily right skewed
compactness_error	1.8922032	Heavily right skewed
concavity_error	5.0835502	Heavily right skewed
concave_points_error	1.4370701	Heavily right skewed
symmetry_error	2.1835728	Heavily right skewed
fractal_dimension_error	3.9033041	Heavily right skewed
radius_worst	1.0973059	Heavily right skewed
texture_worst	0.4956970	Approximately symmetric
perimeter_worst	1.1222227	Heavily right skewed

area_worst	1.8495814	Heavily right skewed
smoothness_worst	0.4132383	Approximately symmetric
compactness_worst	1.4657948	Heavily right skewed
concavity_worst	1.1441794	Heavily right skewed
concave_points_worst	0.4900213	Approximately symmetric
symmetry_worst	1.4263764	Heavily right skewed
fractal_dimension_worst	1.6538237	Heavily right skewed

c. Outliers

A total of 608 outliers were found in the data. The “area_error” feature had the most outliers of 65; “concave_points_worst” feature however had no outliers present. The boxplots below give a graphical view of the outliers denoted by “dots” outside the plots’ boxes.



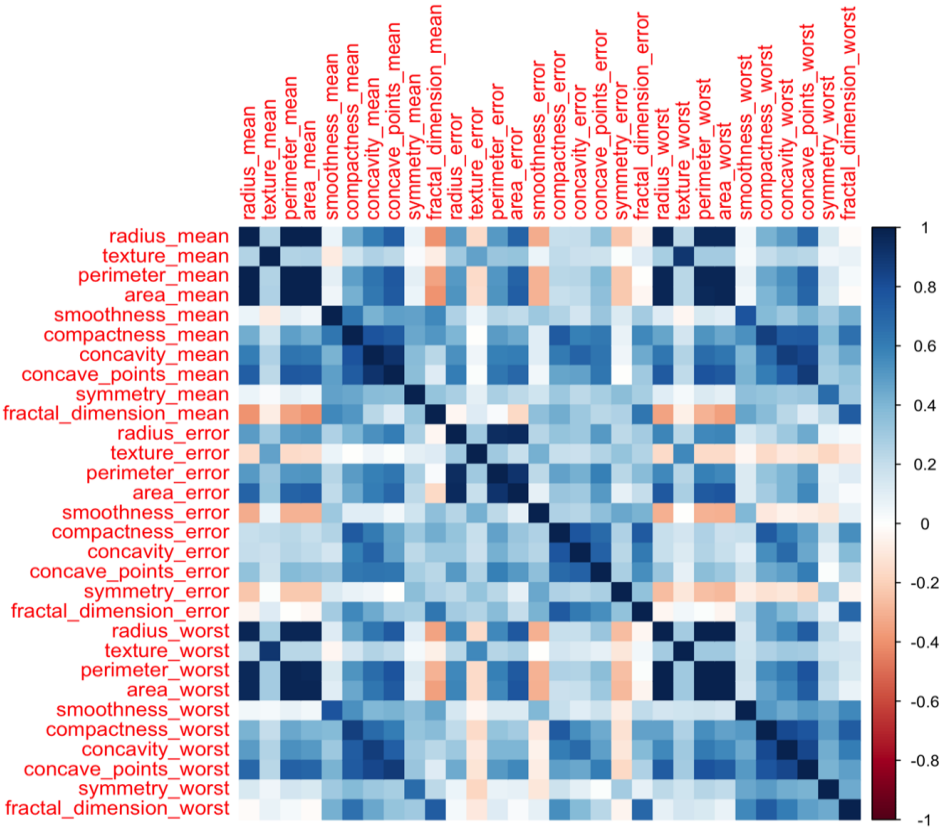


d. Correlation in Data

The predictors were analyzed for the presence of high correlation. For instance, the predictors “radius_mean” and “perimeter_mean”, “area_mean” and “radius_mean” indicated strong correlation with correlation coefficients exceeding 0.98. Appendix 1 has correlation coefficients for the first 10 predictors.

```
> corr_bre_can_dgn[1:10,1:10]
                        radius.mean texture.mean perimeter.mean area.mean smoothness.mean
radius.mean               1.0000000   0.32378189      0.9978553 0.9873572      0.17058119
texture.mean              0.3237819   1.00000000      0.3295331 0.3210857     -0.02338852
perimeter.mean            0.9978553   0.32953306      1.0000000 0.9865068      0.20727816
area.mean                 0.9873572   0.32108570      0.9865068 1.0000000      0.17702838
smoothness.mean           0.1705812  -0.02338852      0.2072782 0.1770284      1.00000000
compactness.mean          0.5061236   0.23670222      0.5569362 0.4985017      0.65912322
concavity.mean            0.6767636   0.30241783      0.7161357 0.6859828      0.52198377
concave_points.mean       0.8225285   0.29346405      0.8509770 0.8232689      0.55369517
symmetry.mean             0.1477412   0.07140098      0.1830272 0.1512931      0.55777479
fractal_dimension.mean   -0.3116308  -0.07643718     -0.2614769 -0.2831098     0.58479200
```

The correlation matrix was plotted to unveil collinearity between bivariate predictors. The heatmap below shows a high correlation relationship between predictors.



3.2 Data Transformations

a. “Center and Scale”, “BoxCox” Transformation

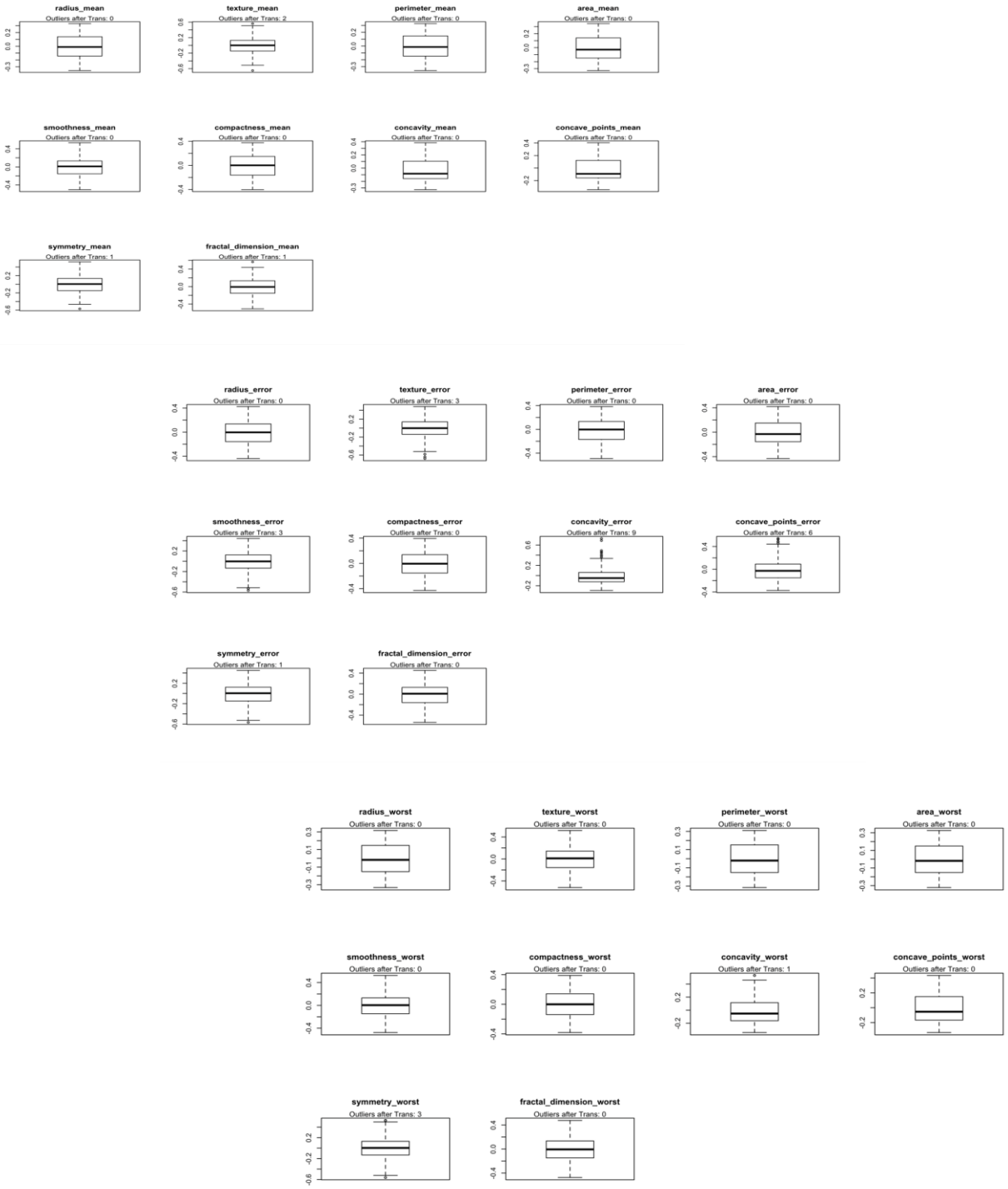
After applying “center and scale” and “BoxCox” transformations, the histogram below showed most of the predictors getting approximately symmetric. Centering and scaling added to the numerical stability and symmetrical alignment of the data.

BoxCox transformation was performed for 24 of the 30 predictors. All 30 predictors were centered and scaled.

```
> preprocTrans
Created from 569 samples and 30 variables

Pre-processing:
  - Box-Cox transformation (24)
  - centered (30)
  - ignored (0)
  - scaled (30)
  - spatial sign transformation (30)

Lambda estimates for Box-Cox transformation:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.0000 -0.5000 -0.3000 -0.3583  0.0000  0.3000

> brecanXcontTrans <- predict(preprocTrans, brecanXcont)
> dim(brecanXcontTrans)
[1] 569  30
```

b. “spatialSign” Transformation

Total outliers reduced from 608 to 30 after applying “spatialSign” transformation as illustrated in the boxplots below.



c. Removing Highly Correlated Data

Three different thresholds were tested to see how many predictors will be removed and analyzed the impact on the data. The table below shows cutoff values of 0.8 applied, resulting in 12 highly correlated predictors removed from the dataset; 18 predictors were retained to build the models.

Cut-off	Total	Predictors to remove
0.75	16	[1] "concavity_mean" "concave_points_worst" "concave_points_mean" [4] "compactness_mean" "perimeter_worst" "concavity_worst" [7] "radius_worst" "area_worst" "compactness_worst" [10] "perimeter_mean" "area_mean" "area_error" [13] "perimeter_error" "compactness_error" "smoothness_mean" [16] "texture_mean"
0.80	**12**	[1] "concavity_mean" "concave_points_worst" "compactness_mean" [4] "perimeter_worst" "concavity_worst" "radius_worst" [7] "area_worst" "perimeter_mean" "area_mean" [10] "area_error" "perimeter_error" "texture_mean" 
0.85	11	[1] "concavity_mean" "concave_points_worst" "compactness_mean" [4] "perimeter_worst" "radius_worst" "area_worst" [7] "perimeter_mean" "area_mean" "area_error" [10] "perimeter_error" "texture_mean"

"Control-Experimenting" with PCA

Principal Component Analysis (PCA) was also tested to find out how many predictors will be retained, and which predictors (PCs) will be considered relevant for the model—results in Appendix 1. PCA needed 11 components to capture 95% of the variance.

Handling the collinearity by directly removing high correlated predictors with a 0.8 threshold was considered the better alternative to deal with the collinearity within the dataset.

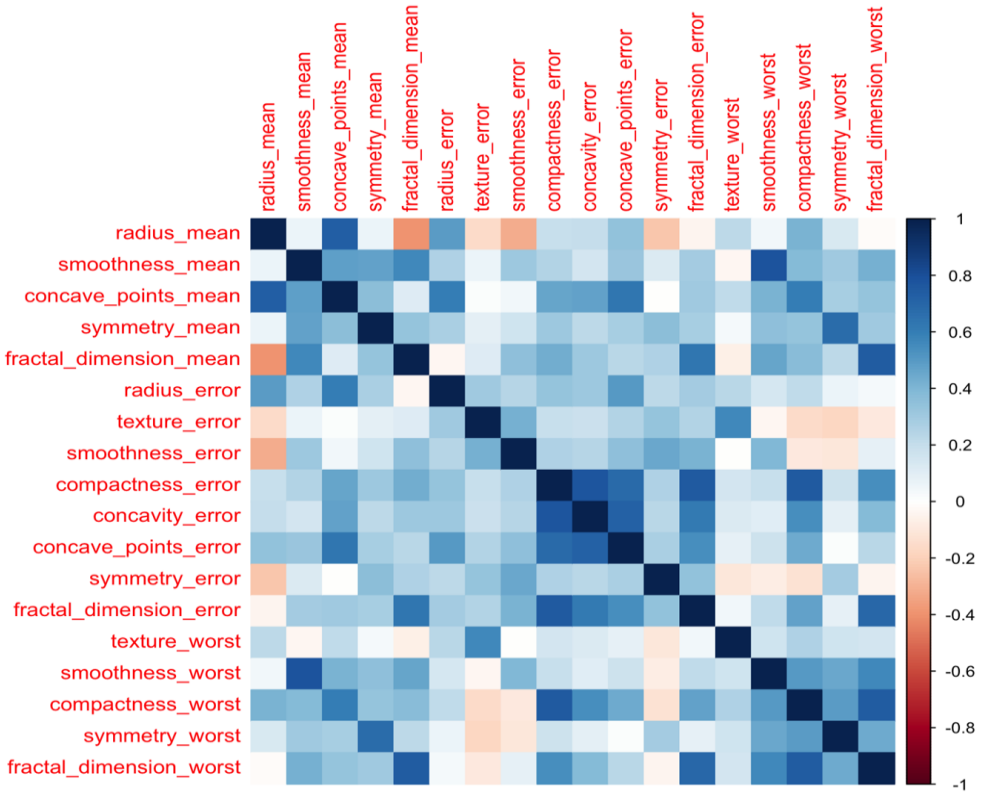


Fig 3.2. Correlation matrix after removing 12 highly correlated predictors.

3.3 Spending the Data

The training set was assigned 80% while 20% was reserved for test. Stratified random sampling was considered most appropriate to handle the “mean”, “standard error”, and “worst” strata within the dataset. The resampling technique used was k-Fold (10-Fold) Cross-Validation and repeating 5 times.

3.4 Data Partition

After the dataset (of 569 observations) was partitioned, 456 was earmarked to train (fit, tune) the model; and 113 was used to test the model. A total of 18 predictors were retained after data transformation.

5.0 Building Models

In fitting the appropriate model for the breast cancer diagnosis data, five (5) linear and eight (8) non-linear classification models were considered. Models were built on training dataset with the 456 partitioned samples.

Linear Classification Models	**Non-linear Classification Models**
Logistic Regression	Quadratic Discriminant Analysis – QDA
Linear Discriminant Analysis – LDA	Regularized Discriminant Analysis – RDA
PLS Discriminant Analysis – PLSDA	Mixture Discriminant Analysis – MDA
Penalized models	Flexible Discriminant Analysis – FDA
Nearest Shrunken Centroids – NSC	Neural Networks – NNet
	Support Vector Machines – SVM
	K-Nearest Neighbors – KNN
	Naive Bayes model — NB

5.1 Logistic Regression Model

No model tuning plot available — no tuning parameter used.

```
Generalized Linear Model

456 samples
 18 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 410, 411, 410, 410, 411, 411, ...
Resampling results:

  Accuracy   Kappa
  0.9592754  0.9131085
```

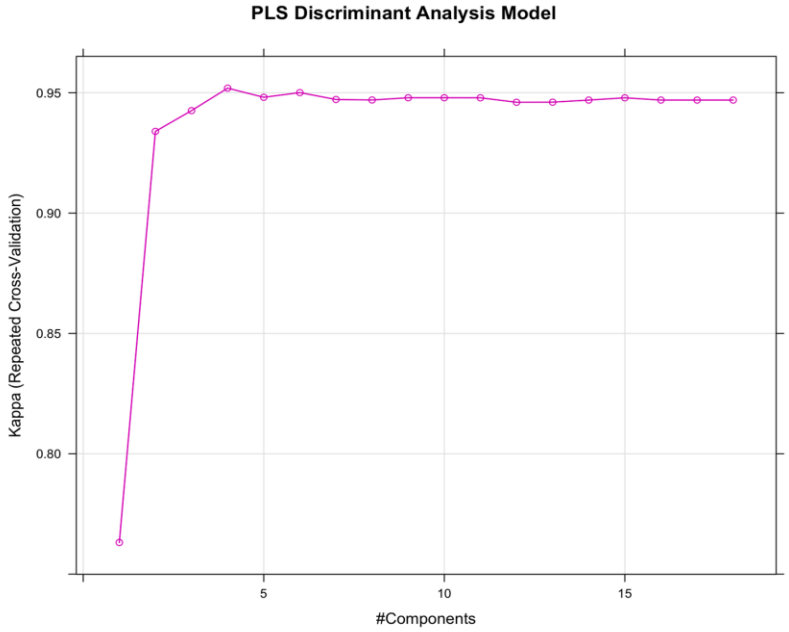
5.2 Linear Discriminant Analysis (LDA) Model

No model tuning plot available — no tuning parameter used.

```
Linear Discriminant Analysis

456 samples
 18 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 411, 410, 410, 410, 410, 411, ...
Resampling results:

  Accuracy   Kappa
  0.9758841  0.9478061
```

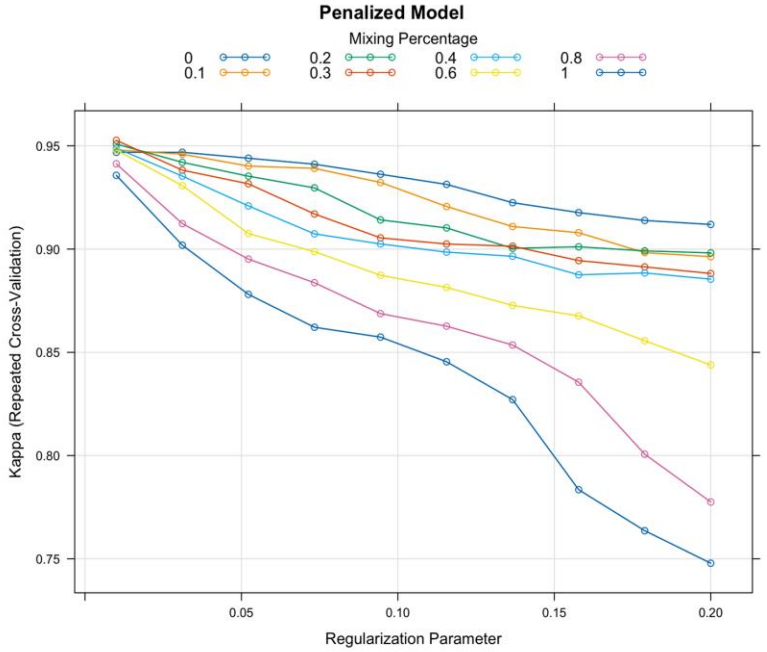
5.3 Partial Least Square Discriminant Analysis (PLSDA) Model

Model tuning plot below. Optimal model at “ncomp” = 4



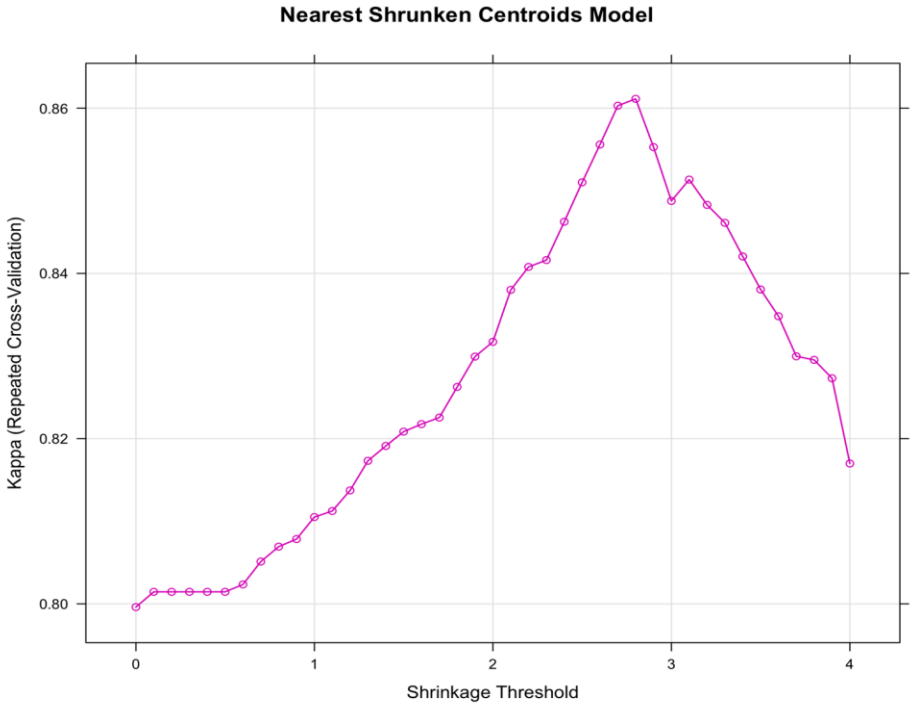
5.4 Penalized Model

Model tuning plot below. Optimal model selected at: lambda = 0.3. gamma=0.01



5.5 Nearest Shrunken Centroids (NSC) Model

Model tuning plot below. Optimal model at shrinking threshold = 2.8



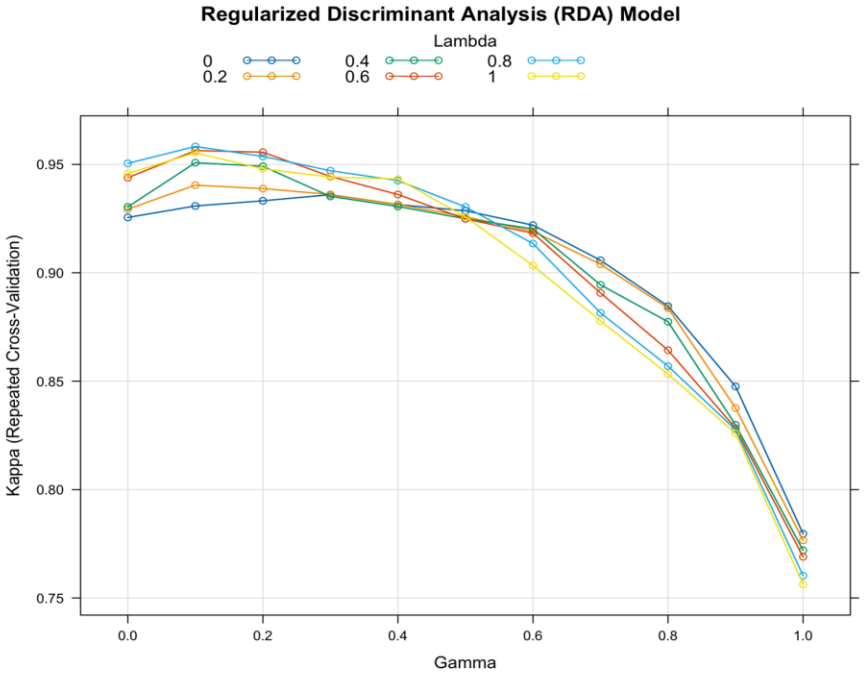
5.6 Quadratic Discriminant Analysis (QDA) Model

No model tuning plot available — no tuning parameter used.

```
Quadratic Discriminant Analysis

456 samples
 18 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 411, 411, 410, 410, 410, 410, ...
Resampling results:

  Accuracy  Kappa
  0.964     0.9214361
```

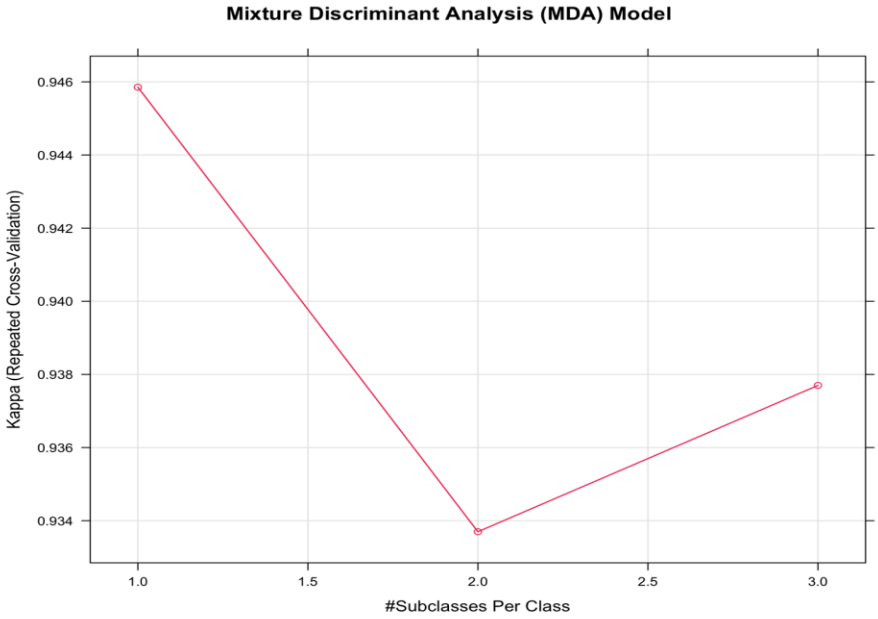
5.7 Regularized Discriminant Analysis (RDA) Model

Model tuning plot below. Optimal model selected at: gamma=0.1. lambda = 0.8.



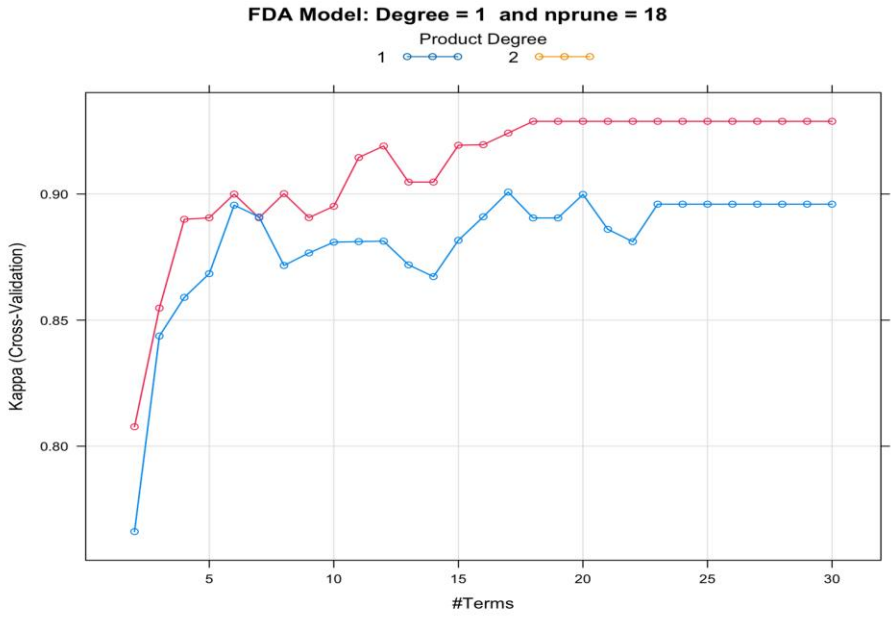
5.8 Mixture Discriminant Analysis (MDA) Model

Model tuning plot below. Optimal model at “subclasses” = 1



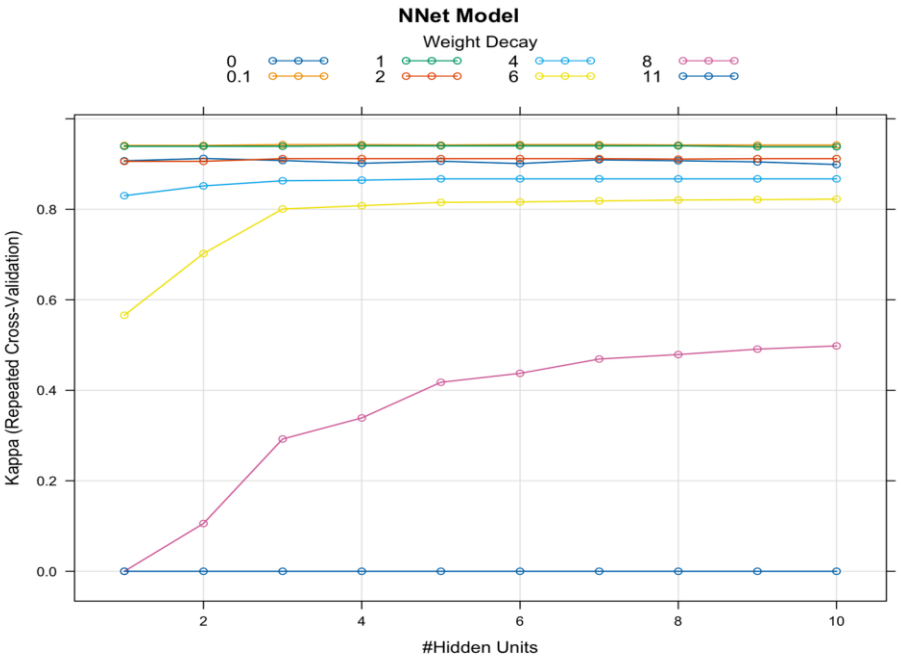
5.9 Flexible Discriminant Analysis (FDA) Model

Model tuning plot below. Optimal model: 18 terms (nprune) retained at degree = 2



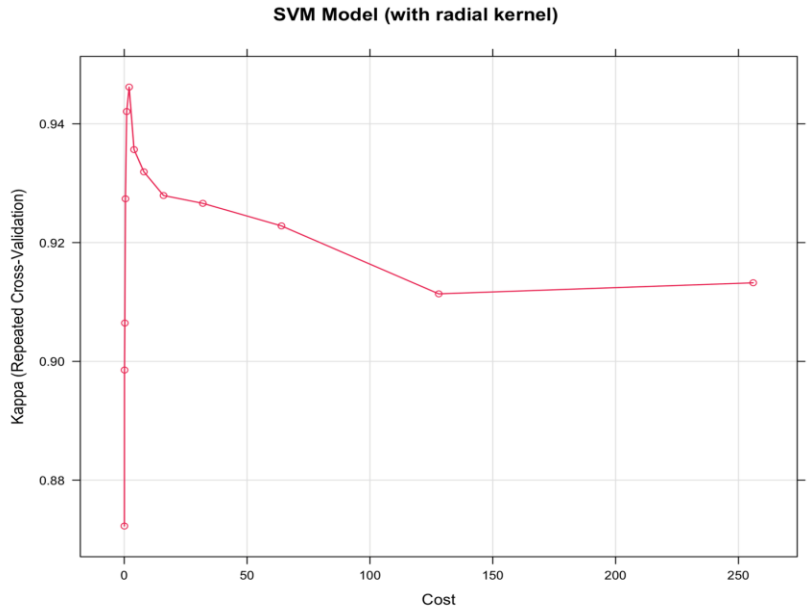
5.10 Neural Networks (NNet) Model

Model tuning plot below. Optimal model at size = 3 and decay = 0.1



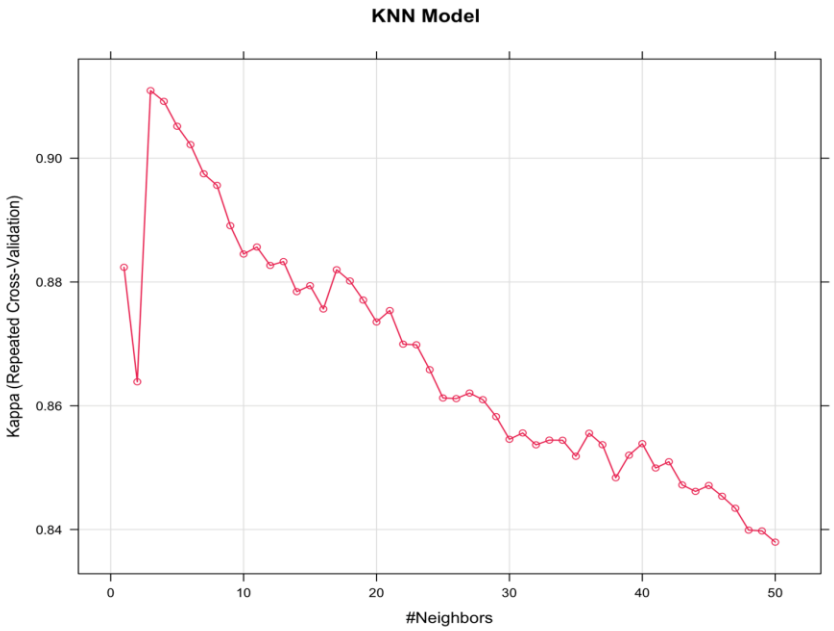
5.11 Support Vector Machines (SVM) Model

Model tuning plot below. Optimal model at Cost = 2. sigma was held constant.



5.12 K-Nearest Neighbors (KNN) Model

Model tuning plot below. Optimal model selected at: k = 3



5.13 Naive Bayes (NB) Model

No model tuning plot. No tuning parameter used for the model. The “fL” (Laplace correction) value was used to “smoothen” (control) the occurrences of zero probability estimates.

```
Naive Bayes

456 samples
 18 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 410, 410, 410, 411, 410, 410, ...
Resampling results:

  Accuracy   Kappa
  0.9337391  0.8564409

Tuning parameter 'fL' was held constant at a value of 2
Tuning parameter
 'usekernel' was held constant at a value of TRUE
Tuning parameter 'adjust' was
 held constant at a value of TRUE
```

6.0 Choosing Best Models: Top Two (2) Models

The Kappa statistic was used to train (or fit) all 13 classification models. Kappa was considered first followed by the accuracy rate. High Kappa values indicate strong concordance—agreement between predictors and the response variable—that an event (someone getting diagnosed as benign or malignant) occurs per chance.

Analyzing results from the Summary Statistics Table (on next page), Regularized Discriminant Analysis (RDA) registered the highest Kappa value of 95.83% and accuracy rate of 98.08% on the training data. This was followed by the Penalized model with 95.27% and accuracy rate of 97.80%. All models except Logistic Regression and Nearest Shrunken Centroids decreased in the evaluation metrics on the test data.

Predicting on the test data, the two best models RDA and Penalized coincidentally had the same values for Kappa (92.57%) and Accuracy rate (96.46%); as well as Sensitivity (94.37%) and Specificity (1.0). The specificity value explains the false positive value of 0 in the confusion matrix. The ROC AUC value for the penalized model (99.77%) was marginally better than the RDA (99.66%). The RDA model however tends to underfit the model (from 98.08% to 96.46%) a little bit more compared to the Penalized model (from 97.80% to 96.46%). Appendix 2 has detailed model performance on both training and test data for the top two (2) models—RDA and Penalized models.

The Penalized model is therefore selected as the best model.

6.1 Summary Statistics Table

	Model	Best Tuning Parameter	Training		Testing		
			AR	Kappa	AR	Kappa	ROC AUC
LINEAR MODELS	LR	None	0.9593	0.9131	0.9646 ↑	0.9249 ↑	0.9990**
	LDA	None	0.9759	0.9478	0.9558 ↓	0.9075 ↓	0.9970
	PLSDA	ncomp = 4	0.9776	0.9519	0.9558 ↓	0.9066 ↓	0.9963
	Penalized	**alpha = 0.3, lambda = 0.01**	**0.9780***	**0.9527***	**0.9646 ↓**	**0.9257 ↓**	**0.9977**
	NSC	threshold = 2.8	0.9364	0.8611	0.9381 ↑	0.8680 ↑	0.9805
NON-LINEAR MODELS	QDA	None	0.9640	0.9214	0.9558 ↓	0.9048 ↓	0.9930
	RDA	**Gamma = 0.1, lambda = 0.8**	**0.9808****	**0.9583****	**0.9646 ↓**	**0.9257 ↓**	**0.9966**
	MDA	subclasses = 1	0.9750	0.9459	0.9558 ↓	0.9075 ↓	0.9970
	NNet	size = 3, decay = 0.1	0.9732	0.9429	0.9646 ↓	0.9249 ↓	0.9987*
	FDA	degree = 1, nprune = 18	0.9671	0.9288	0.9381 ↓	0.8693 ↓	0.9903
	SVM	cost, C = 2	0.9749	0.9462	0.9558 ↓	0.9057 ↓	0.9966
	KNN	k = 3	0.9583	0.9109	0.9381 ↓	0.8693 ↓	0.9792
	NB	None	0.9337	0.8564	0.9204 ↓	0.8303 ↓	0.9826

6.2 Confusion Matrix for Best Model (Penalized Model)

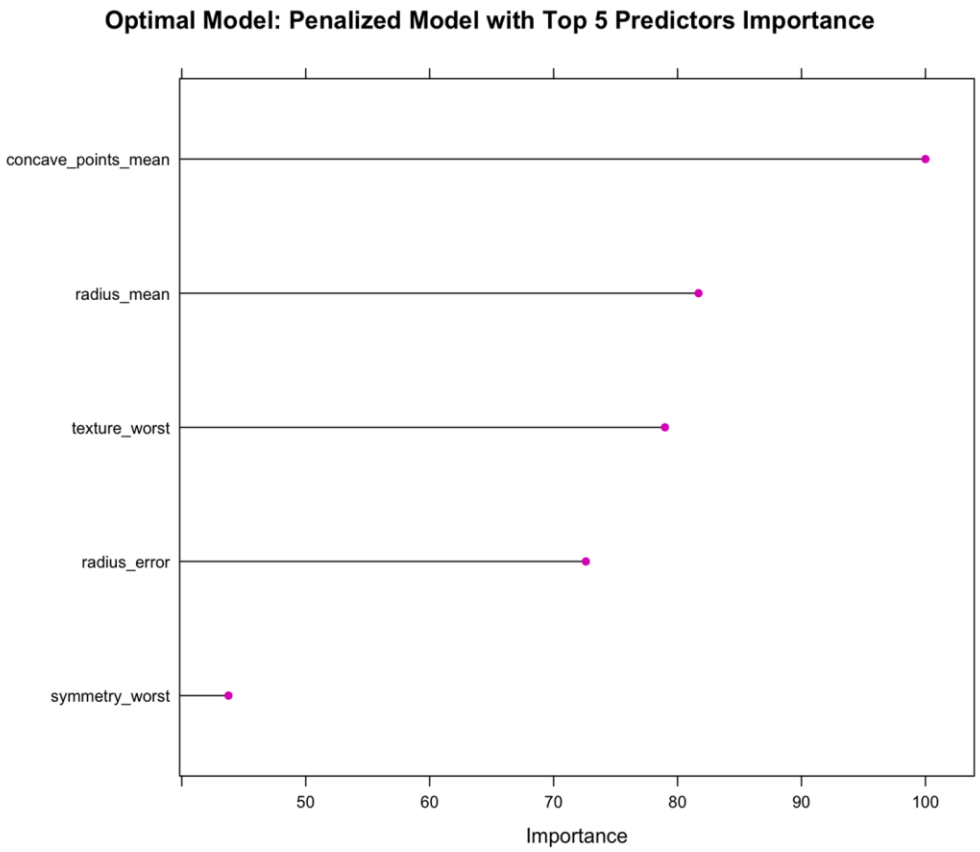
The confusion matrix after model was tested on new, unseen data.

```
> glmnConfMatrix
Confusion Matrix and Statistics

          Reference
Prediction  B  M
         B 67  0
         M  4 42

               Accuracy : 0.9646
                 95% CI : (0.9118, 0.9903)
    No Information Rate : 0.6283
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9257

 Mcnemar's Test P-Value : 0.1336

            Sensitivity : 0.9437
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9130
             Prevalence : 0.6283
         Detection Rate : 0.5929
   Detection Prevalence : 0.5929
      Balanced Accuracy : 0.9718

       'Positive' Class : B
```

6.3 Important Predictors using the Penalized Model

The top 5 predictors using the best model in order of importance are as follows:

#	Predictor	Overall
1	concave_points_mean	100.000
2	radius_mean	81.691
3	texture_worst	78.984
4	radius_error	72.602
5	symmetry_worst	43.768



7.0 Summary and Recommendations

In conclusion, all thirteen (13) classification models were trained on the data. All models were tested on new, unseen data, however limiting the analysis on the test data to the two best models from training—namely RDA and Penalized models. Generally, the models performed creditably well (with least metric at 83%) considering the summary statistics table information above.

The Penalized model is therefore the recommended model for the study, and possibly productionized for onward business case usage.

8.0 References

Applied Predictive Modeling, Max Kuhn and Kjell Johnson

url: https://link.springer.com/book/10.1007/978-1-4614-6849-3

Breast Cancer Wisconsin (Diagnostic)

url: http://archive.ics.uci.edu/dataset/17/breast**+cancer+wisconsin+diagnostic**

National Cancer Institute

url: https://seer.cancer.gov/statfacts/html/common.html#:~:text=Statistics%20at%20a%20Glance&text=Breast%2C%20lung%20and%20bronchus%2C%20prostate,nearly%2050%25%20of%20all%20deaths

Kaggle

url: https://www.kaggle.com/

StackOverflow

url: https://stackoverflow.com/

Appendix 1: Data Exploration and Preprocessing Outputs

Explored dataset.

Class column ($diagnosis) in “chr” datatype was encoded as a factor to have two (2) levels— “B” for Benign and “M” for Malignant.

```
'data.frame':   569 obs. of  32 variables:
 $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 84501001 ...
 $ diagnosis              : Factor w/ 2 levels "B","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ radius_mean            : num  18 20.6 19.7 11.4 20.3 ...
 $ texture_mean           : num  10.4 17.8 21.2 20.4 14.3 ...
 $ perimeter_mean         : num  122.8 132.9 130 77.6 135.1 ...
 $ area_mean              : num  1001 1326 1203 386 1297 ...
 $ smoothness_mean        : num  0.1184 0.0847 0.1096 0.1425 0.1003 ...
 $ compactness_mean       : num  0.2776 0.0786 0.1599 0.2839 0.1328 ...
 $ concavity_mean         : num  0.3001 0.0869 0.1974 0.2414 0.198 ...
 $ concave_points_mean    : num  0.1471 0.0702 0.1279 0.1052 0.1043 ...
 $ symmetry_mean          : num  0.242 0.181 0.207 0.26 0.181 ...
 $ fractal_dimension_mean : num  0.0787 0.0567 0.06 0.0974 0.0588 ...
 $ radius_error           : num  1.095 0.543 0.746 0.496 0.757 ...
 $ texture_error          : num  0.905 0.734 0.787 1.156 0.781 ...
 $ perimeter_error        : num  8.59 3.4 4.58 3.44 5.44 ...
 $ area_error             : num  153.4 74.1 94 27.2 94.4 ...
 $ smoothness_error       : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
 $ compactness_error      : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
 $ concavity_error        : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
 $ concave_points_error   : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
 $ symmetry_error         : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
 $ fractal_dimension_error: num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
 $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
 $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
 $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
 $ area_worst             : num  2019 1956 1709 568 1575 ...
 $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
 $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
 $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
 $ concave_points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
 $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
 $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

Removing high correlated data

Threshold: 0.75

```
> corThresh <- .75
> tooHigh <- findCorrelation(cor(brecanXcontTrans), corThresh) #pred to remove
> length(tooHigh)
[1] 16
> (dim(brecanXcontTrans)[2]-length(findCorrelation(cor(brecanXcontTrans), corThresh)))
#pred to retain
[1] 14
> corrPred <- names(brecanXcontTrans)[tooHigh]
> names(brecanXcontTrans)[tooHigh]  # too high predictors...
 [1] "concavity_mean"       "concave_points_worst" "concave_points_mean"
 [4] "compactness_mean"     "perimeter_worst"      "concavity_worst"
 [7] "radius_worst"         "area_worst"           "compactness_worst"
[10] "perimeter_mean"       "area_mean"            "area_error"
[13] "perimeter_error"      "compactness_error"    "smoothness_mean"
[16] "texture_mean"
```

Threshold: 0.8

```
> corThresh <- .8
> tooHigh <- findCorrelation(cor(brecanXcontTrans), corThresh) #pred to remove
> length(tooHigh)
[1] 12
> (dim(brecanXcontTrans)[2]-length(findCorrelation(cor(brecanXcontTrans), corThresh)))
#pred to retain
[1] 18
> corrPred <- names(brecanXcontTrans)[tooHigh]
> names(brecanXcontTrans)[tooHigh]  # too high predictors...
 [1] "concavity_mean"       "concave_points_worst" "compactness_mean"
 [4] "perimeter_worst"      "concavity_worst"      "radius_worst"
 [7] "area_worst"           "perimeter_mean"       "area_mean"
[10] "area_error"           "perimeter_error"      "texture_mean"
```

Threshold: 0.85

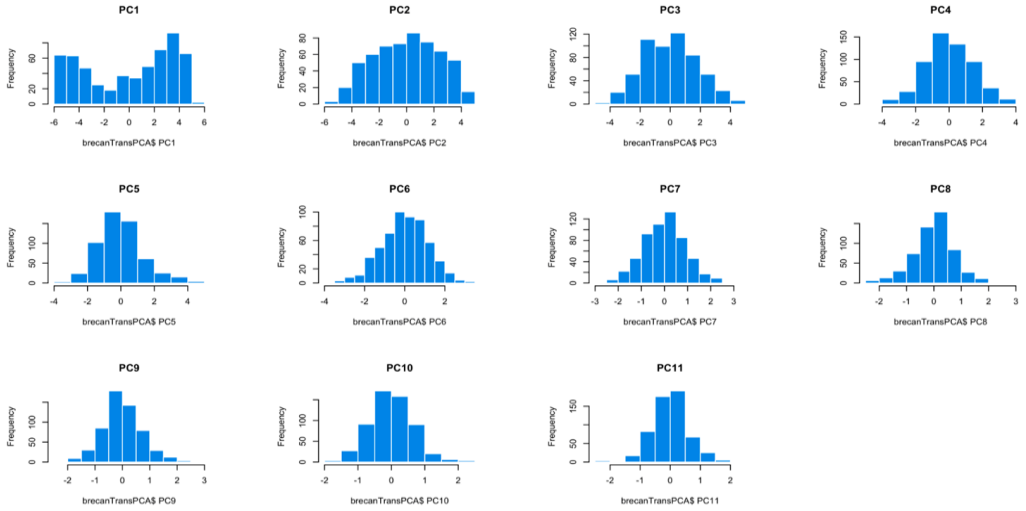
```
> corThresh <- .85
> tooHigh <- findCorrelation(cor(brecanXcontTrans), corThresh) #pred to remove
> length(tooHigh)
[1] 11
> (dim(brecanXcontTrans)[2]-length(findCorrelation(cor(brecanXcontTrans), corThresh)))
#pred to retain
[1] 19
> corrPred <- names(brecanXcontTrans)[tooHigh]
> names(brecanXcontTrans)[tooHigh]  # too high predictors...
 [1] "concavity_mean"       "concave_points_worst" "compactness_mean"
 [4] "perimeter_worst"      "radius_worst"         "area_worst"
 [7] "perimeter_mean"       "area_mean"            "area_error"
[10] "perimeter_error"      "texture_mean"
```

Correlation coefficients for the first 10 predictors:

```
> corr_bre_can_dgn[1:10,1:10]
                        radius.mean texture.mean perimeter.mean area.mean smoothness.mean
radius.mean               1.0000000   0.32378189      0.9978553 0.9873572      0.17058119
texture.mean              0.3237819   1.00000000      0.3295331 0.3210857     -0.02338852
perimeter.mean            0.9978553   0.32953306      1.0000000 0.9865068      0.20727816
area.mean                 0.9873572   0.32108570      0.9865068 1.0000000      0.17702838
smoothness.mean           0.1705812  -0.02338852      0.2072782 0.1770284      1.00000000
compactness.mean          0.5061236   0.23670222      0.5569362 0.4985017      0.65912322
concavity.mean            0.6767636   0.30241783      0.7161357 0.6859828      0.52198377
concave_points.mean       0.8225285   0.29346405      0.8509770 0.8232689      0.55369517
symmetry.mean             0.1477412   0.07140098      0.1830272 0.1512931      0.55777479
fractal_dimension.mean   -0.3116308  -0.07643718     -0.2614769 -0.2831098     0.58479200

compactness.mean concavity.mean concave_points.mean symmetry.mean fractal_dimension.mean
       0.5061236      0.6767636           0.8225285     0.14774124            -0.31163083
       0.2367022      0.3024178           0.2934641     0.07140098            -0.07643718
       0.5569362      0.7161357           0.8509770     0.18302721            -0.26147691
       0.4985017      0.6859828           0.8232689     0.15129308            -0.28310981
       0.6591232      0.5219838           0.5536952     0.55777479             0.58479200
       1.0000000      0.8831207           0.8311350     0.60264105             0.56536866
       0.8831207      1.0000000           0.9213910     0.50066662             0.33678336
       0.8311350      0.9213910           1.0000000     0.46249739             0.16691738
       0.6026410      0.5006666           0.4624974     1.00000000             0.47992133
       0.5653687      0.3367834           0.1669174     0.47992133             1.00000000
```

What if PCA was used?

This feature reduction by PCA will use only 11 out of the 30 predictors — which may have a high impact on the sensitivity in the dataset.



11 predictors retained by PCA – feature reduction:

```
> str(brecanXcontTransPCA)
'data.frame':   569 obs. of  11 variables:
 $ PC1 : num  -4.97 -3.5 -5.7 -3.29 -4.48 ...
 $ PC2 : num  -1.329 3.744 0.323 -4.154 1.046 ...
 $ PC3 : num  0.9757 0.2756 0.0662 0.9391 -0.8222 ...
 $ PC4 : num  -1.9478 -1.056 -0.9536 -0.0457 -2.9266 ...
 $ PC5 : num  -0.341 0.081 -0.391 -1.128 0.932 ...
 $ PC6 : num  -0.585 0.178 -0.512 -0.828 1.07 ...
 $ PC7 : num  -0.431 -1.175 0.191 0.177 0.401 ...
 $ PC8 : num  0.094 0.3953 0.0207 0.2335 0.6746 ...
 $ PC9 : num  -0.3648 0.5775 0.3268 -0.0506 0.2527 ...
 $ PC10: num  -0.8545 -0.2927 0.0522 -0.3513 -0.4241 ...
 $ PC11: num  0.0345 -1.4876 0.0182 0.5869 -0.8296 ...
>
```

After data was partitioned:

```
> dim(brecanX)
[1] 569  18
> # partioned dataset into 80% training and 20% testing...
> dim(trainRows)
[1] 456   1
> dim(brecanX_train)
[1] 456  18
> length(brecanY_train)
[1] 456
> dim(brecanX_test)
[1] 113  18
> length(brecanY_test)
[1] 113
```

Appendix 2: Building Models Outputs

Penalized Models

Fitting the model—preprocessing on training data

Accuracy Rate = 0.9780483 (97.80%). Kappa = 0.9526598 (95.27%)

```
glmnet

456 samples
 18 predictor
  2 classes: 'B', 'M'

Pre-processing: centered (18), scaled (18)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 411, 410, 411, 410, 410, 411, ...
Resampling results across tuning parameters:

  alpha  lambda      Accuracy   Kappa
  0.0    0.01000000  0.9754010  0.9468763
  0.0    0.03111111  0.9754010  0.9468763
  0.0    0.05222222  0.9740966  0.9439955
  0.2    0.17888889  0.9538937  0.8991271
  0.2    0.20000000  0.9534493  0.8981480
  0.3    0.01000000  0.9780483  0.9526598
  0.3    0.03111111  0.9714686  0.9382106
  0.3    0.05222222  0.9683865  0.9316178
  1.0    0.15777778  0.9025990  0.7834607
  1.0    0.17888889  0.8947053  0.7636039
  1.0    0.20000000  0.8890531  0.7478615

Kappa was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.3 and lambda = 0.01.
```

Prediction on test data

Confusion Matrix: model performance on test data below.

Accuracy Rate = 0.9646018 (96.46%). Kappa = 0.9256579 (92.57%). AUC = 0.9977 (99.77%)

```
> glmnConfMatrix
Confusion Matrix and Statistics

          Reference
Prediction  B  M
         B 67  0
         M  4 42

               Accuracy : 0.9646
                 95% CI : (0.9118, 0.9903)
    No Information Rate : 0.6283
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9257

 Mcnemar's Test P-Value : 0.1336

            Sensitivity : 0.9437
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9130
             Prevalence : 0.6283
         Detection Rate : 0.5929
   Detection Prevalence : 0.5929
      Balanced Accuracy : 0.9718

       'Positive' Class : B
```

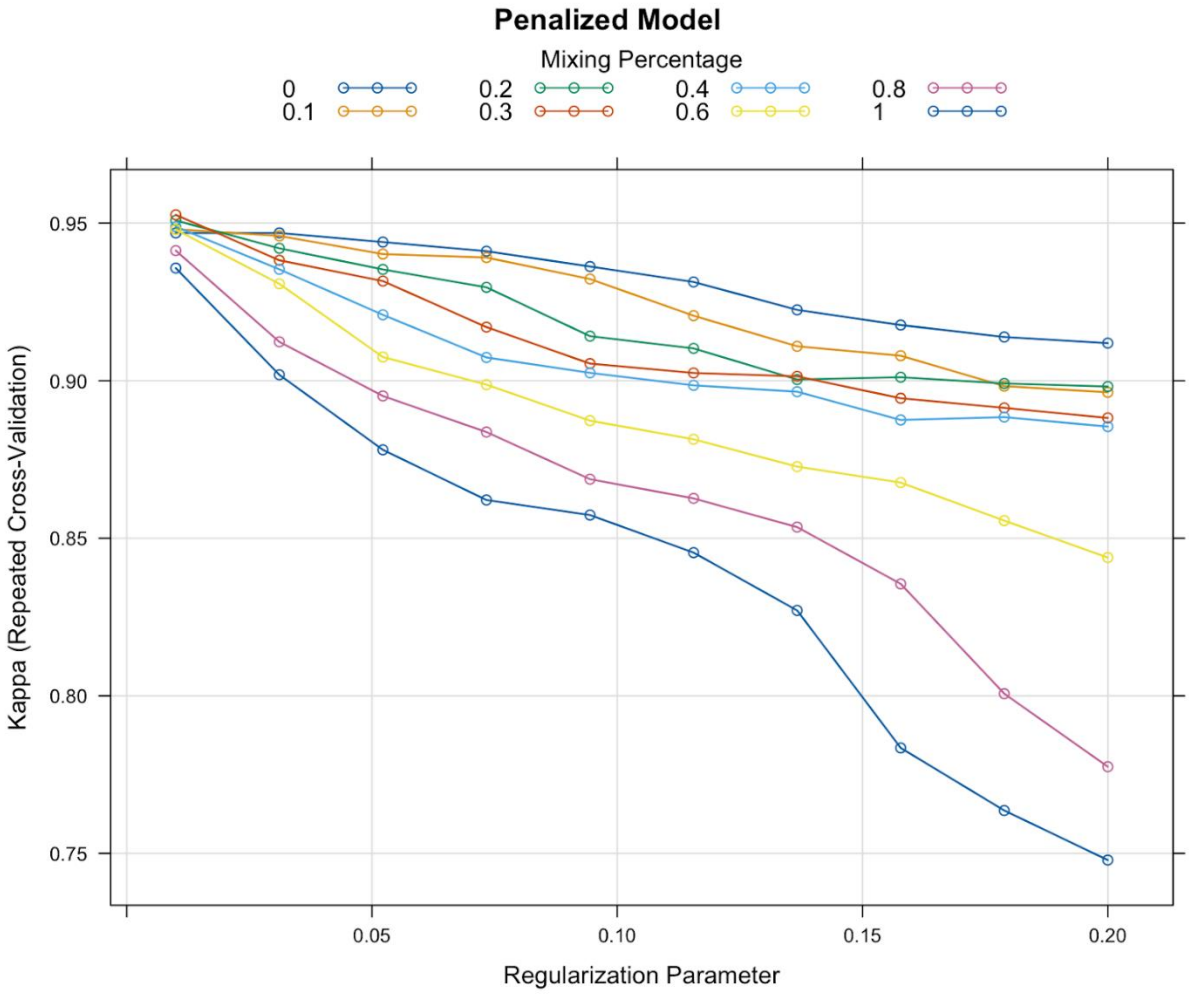
Predicted data:

```
> glmnPredTest
  [1] M M M M M M M B B B B M B M M B M M B B B M M M B M B M B B B B M M B B M B B
 [41] B M M B M M M B B B B M M M B B B M M M M B M B M B B B B B B B B M M M B B B B
 [81] B M M B B M B B M B B B B B B B B B B B B B M B M B B M M B B B B
Levels: B M
```

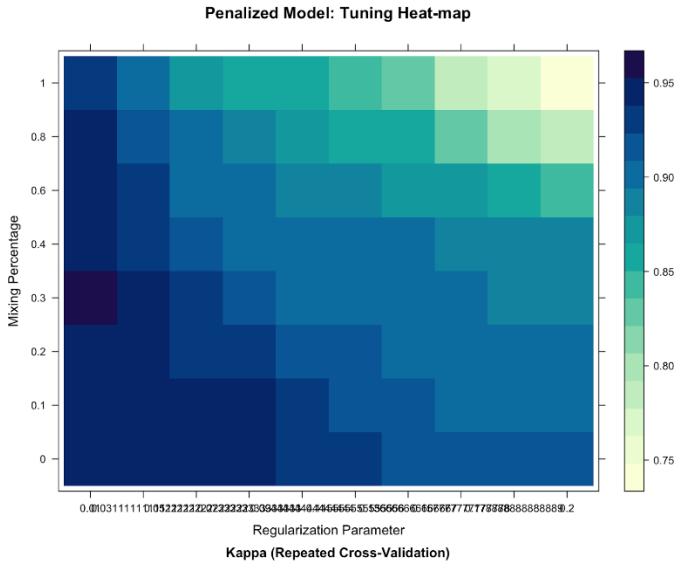
Area under curve (ROC) — AUC: 0.9977 (99.77%)

```
> # find AUC
> glmnAUC <- auc(glmnROC)
> print(glmnAUC)
Area under the curve: 0.9977
```

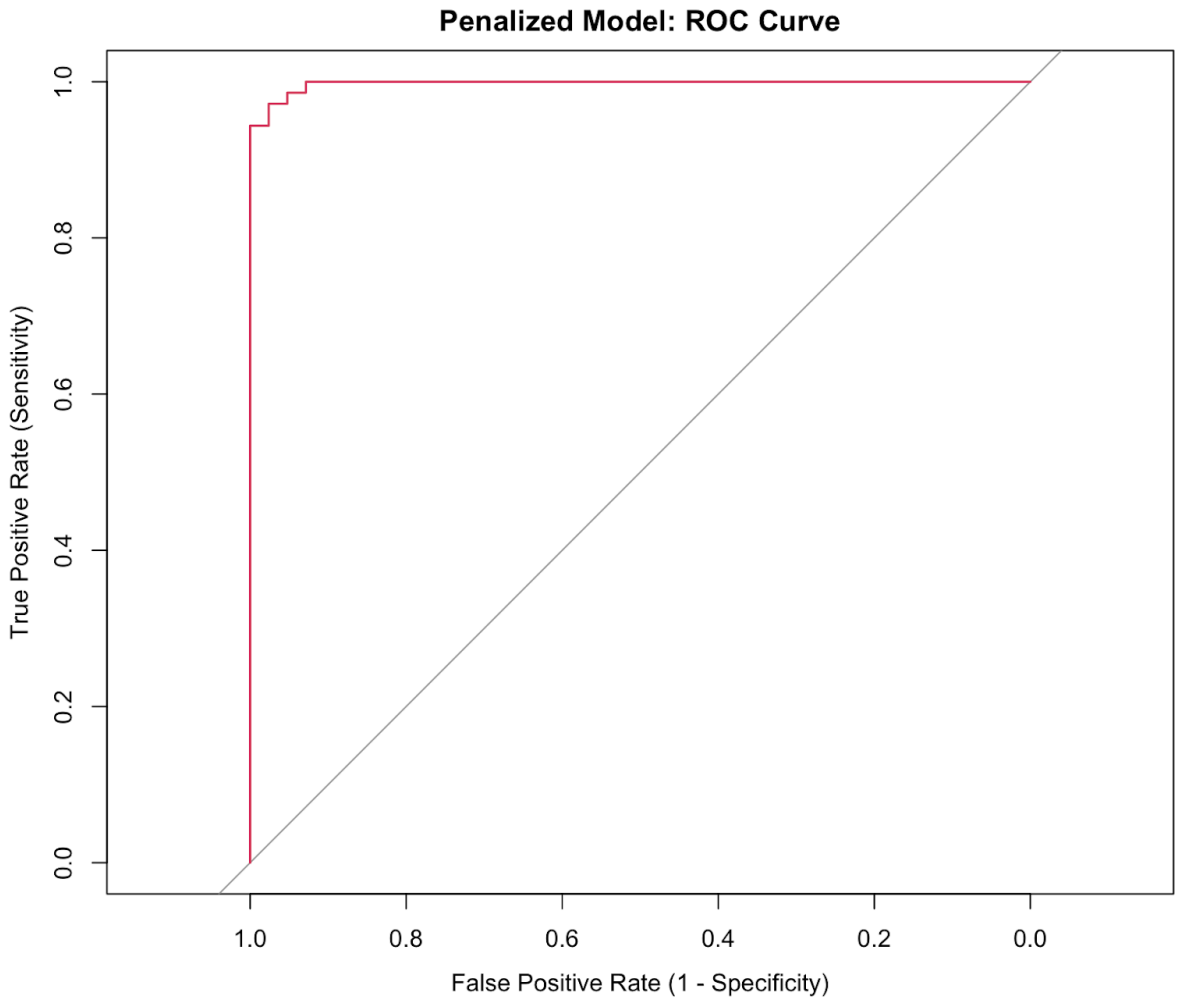
Model Tuning Plot



Tuning Parameters Variation—Heatmap



The ROC Curve



Regularized Discriminant Analysis (RDA) Model

Fitting the model—preprocessing on training data

Accuracy Rate = 0.9807536 (98.08%). Kappa = 0.9582638 (95.83%).

```
Regularized Discriminant Analysis

456 samples
 18 predictor
  2 classes: 'B', 'M'

Pre-processing: centered (18), scaled (18)
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 410, 410, 410, 411, 410, 411, ...
Resampling results across tuning parameters:

  lambda  gamma  Accuracy   Kappa
  0.0     0.0    0.9657971  0.9255173
  0.0     0.1    0.9680386  0.9308278
  0.0     0.2    0.9689179  0.9332048
  0.0     0.3    0.9702415  0.9360114
  0.6     0.9    0.9197198  0.8283169
  0.6     1.0    0.8916135  0.7690090
  0.8     0.0    0.9772464  0.9504841
  0.8     0.1    0.9807536  0.9582638
  0.8     0.2    0.9785507  0.9536842
  0.8     0.3    0.9754879  0.9470860
  1.0     0.9    0.9184058  0.8259653
  1.0     1.0    0.8850435  0.7562424

Kappa was used to select the optimal model using the largest value.
The final values used for the model were gamma = 0.1 and lambda = 0.8.
```

Prediction on test data

Confusion Matrix: model performance on test data below.

Accuracy Rate = 0.9646018 (96.46%). Kappa = 0.9256579 (92.57%). AUC = 0.9966 (99.66%)

```
> rdaConfMatrix
Confusion Matrix and Statistics

          Reference
Prediction  B  M
         B 67  0
         M  4 42

               Accuracy : 0.9646
                 95% CI : (0.9118, 0.9903)
    No Information Rate : 0.6283
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9257

 Mcnemar's Test P-Value : 0.1336

            Sensitivity : 0.9437
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9130
             Prevalence : 0.6283
         Detection Rate : 0.5929
   Detection Prevalence : 0.5929
      Balanced Accuracy : 0.9718

       'Positive' Class : B
```

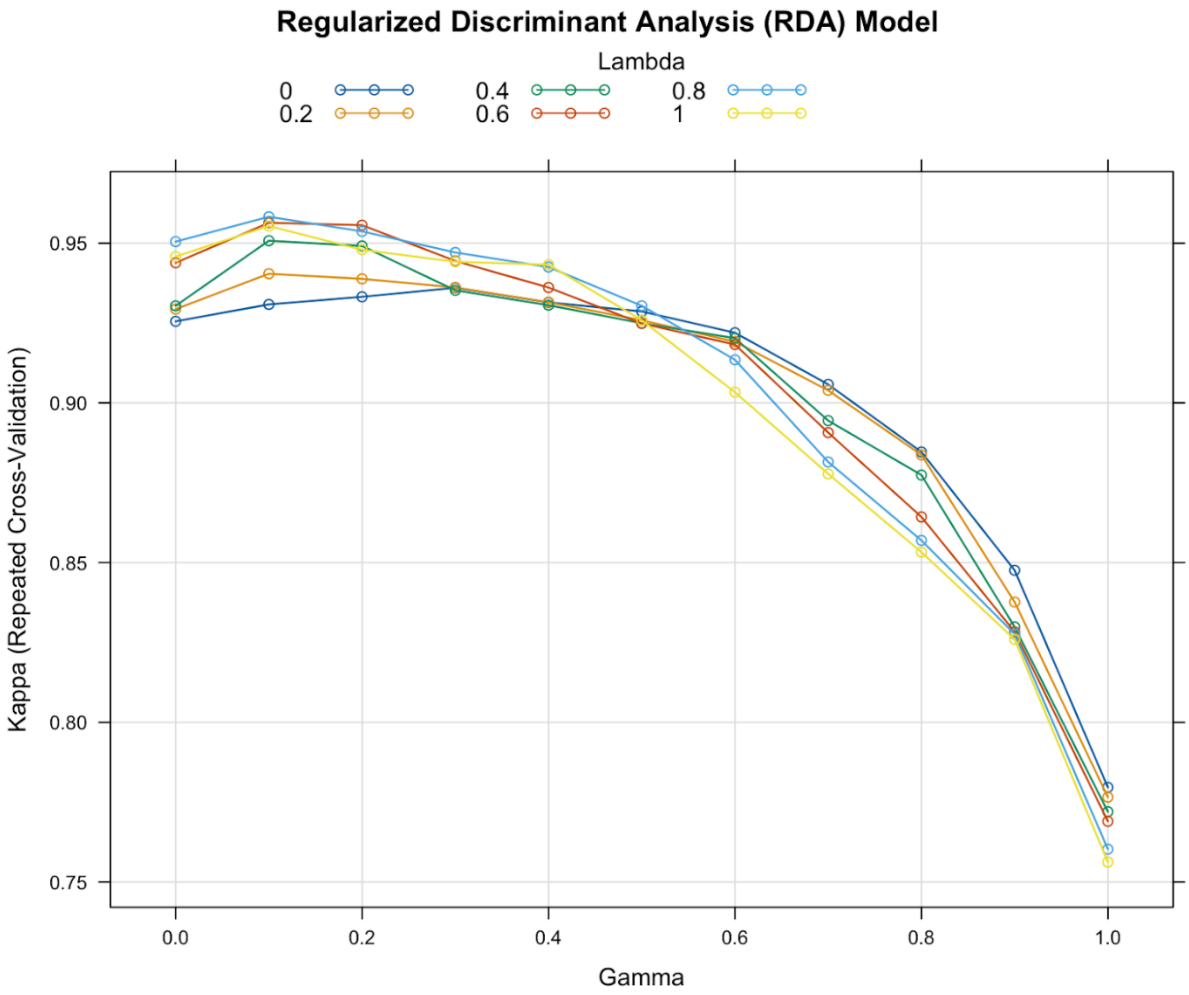
Predicted data

```
> rdaPredTest
  [1] M M M M M M M B B B B M B M M B M M B B B M M M B M B M B B B B M M B B M B B
 [41] B M M B M M M B B B B M M M B B B M M M M B M B M B B B B B B B B M M M B B B B
 [81] B M M B B M B B M B B B B B B B B B B B B M B M B B M M B B B B
Levels: B M
```

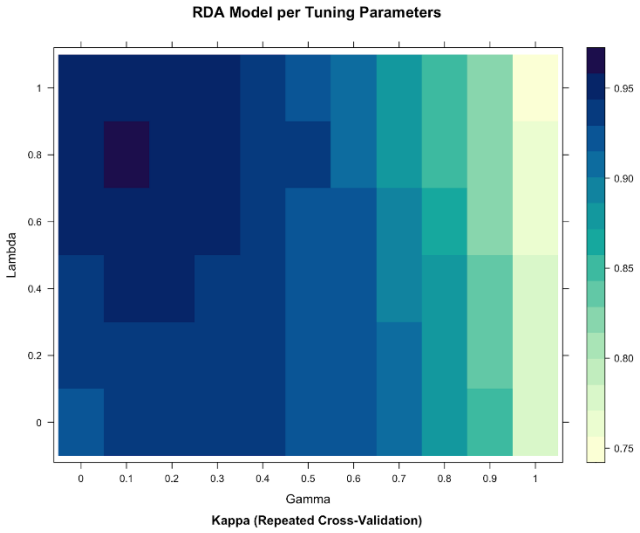
Area under curve (ROC) — AUC: 0.9966 (99.66%)

```
> # find AUC
> rdaAUC <- auc(rdaROC)
> print(rdaAUC)
Area under the curve: 0.9966
```

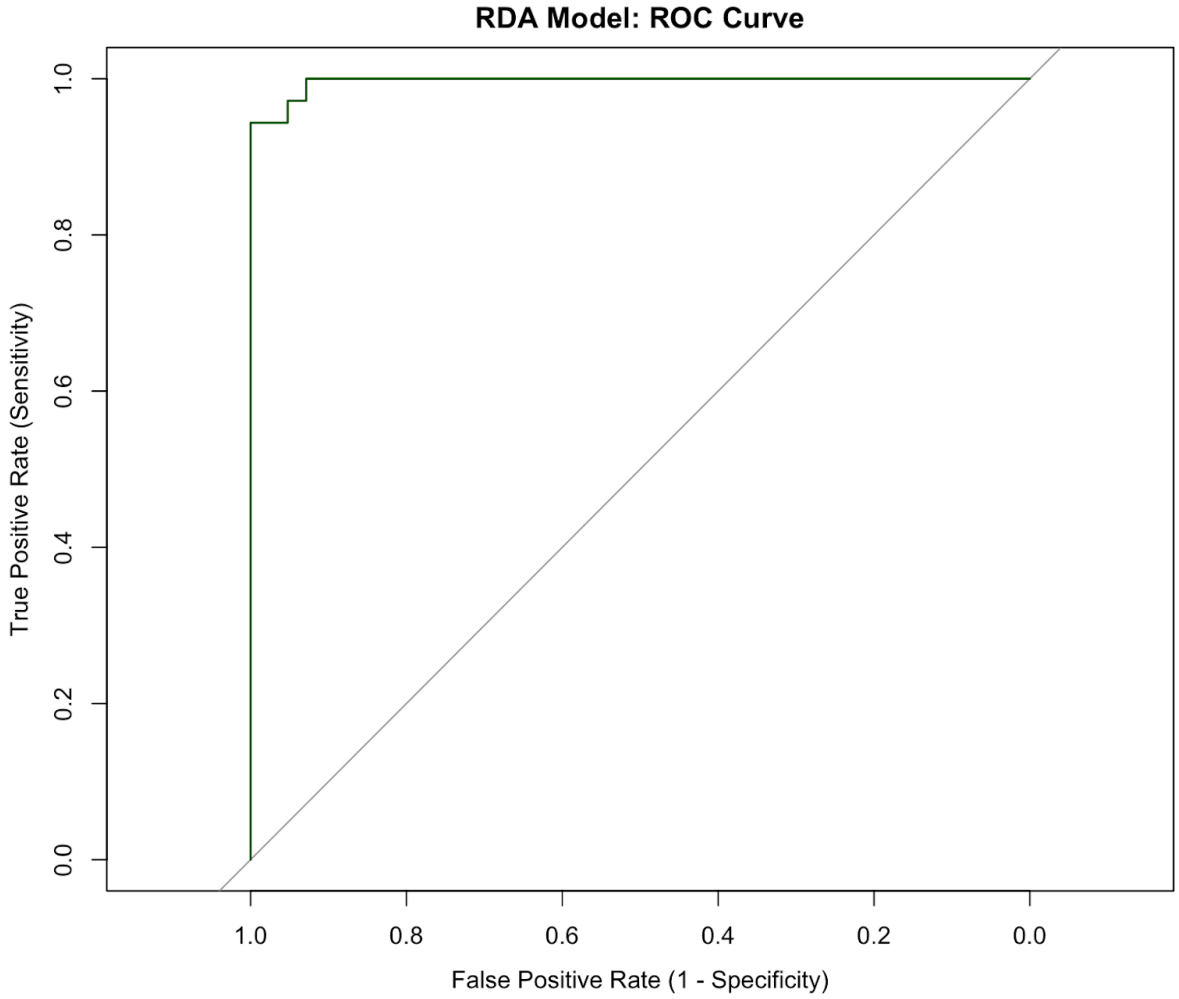
Model Tuning Plot



Tuning Parameters Variation—Heatmap



The ROC Curve



Appendix 3: R-Code

```
## dataset  ---

# [local... downloaded dataset]

# dataset: UCI -- breast+cancer+wisconsin+diagnostic

# wdbc.names: contains descriptions of the features and metadata of the data

# ------------------------------------------------------------------------

#dev.off()


## load data

#ds_src <- "/Users/coderoom/Codespace/MA5790/Project/dataset/wdbc.data"

bre_can_dgn_o <- read.table(ds_src, header = F, sep = ",")

dim(bre_can_dgn_o)

str(bre_can_dgn_o)


##ifelse(brecanY=='B', 'Benign','Malignant')

## B-Benign [no cancer], M-Malignant [cancerous]

## replace col_names ex. X17.99 with descriptive names ex. radius_mean

bre_can_dgn_o <- setNames(bre_can_dgn_o,

c("id","diagnosis","radius_mean","texture_mean","perimeter_mean","area_mean","smoothn
ess_mean","compactness_mean","concavity_mean","concave_points_mean","symmetry_mean","
fractal_dimension_mean",


"radius_error","texture_error","perimeter_error","area_error","smoothness_error","com
pactness_error","concavity_error","concave_points_error","symmetry_error","fractal_di
mension_error",
```

```
"radius_worst","texture_worst","perimeter_worst","area_worst","smoothness_worst","com
pactness_worst","concavity_worst","concave_points_worst","symmetry_worst","fractal_di
mension_worst"))


#### 1. Explore Data

## data structure -- predictors: continuous/categorical

## convert the response[chr] to factor col with levels

#x <- factor(c('B','M'), levels=c('B','M'))

#x <- as.factor(brecanX$diagnosis)

#x <- as.factor(brecanX[, 'cyl'])


bre_can_dgn_o$diagnosis <- as.factor(bre_can_dgn_o$diagnosis)

brecan <- bre_can_dgn_o[,]

str(brecan)


# get the response var

brecanY <- brecan$diagnosis

print(brecanY)


# get predictors... drop [id and diagnosis] columns

brecanX <- brecan[, -c(1,2)]

str(brecanX)

colnames(brecanX)


## class distribution -- response var: balanced/imbalanced.
```

```
# Bar Plot

#dev.off()

tbl <- table(ifelse(brecanY=='B', 'Benign','Malignant'))

barplot(tbl, main="Breast Cancer Diagnosis: Class Distribution",

        xlab="Class", ylab = "Obs. Count",

        col = c('#e177bc', 'firebrick'), border = 0)

grid(col = 'lightgrey', lty = 3, lwd = par("lwd"), equilogs = TRUE) #nx = NULL, ny =
nx,

# 'whitesmoke', 'lightpink' pink -- #e177bc

## data distribution



#### 2. Data Pre-processing


## check for missing data  ---add plot

missVals <- sum(is.na(brecanX)==TRUE)

missVals


# missing vals--plot

par(mfrow = c(1,1), pin=c(5,3))

image(is.na(brecanX), main = "Missing Values",

      xlab = "Observation", ylab = "Variable",

      xaxt = "n", yaxt = "n", bty = "n")

axis(1, seq(0, 1, length.out = nrow(brecanX)), 1:nrow(brecanX), col = 'gainsboro')
```

```
## impute data -- knn impute, mean impute etc.

#library(caret)

## impute missing by knn

#missImp <- preProcess(brecanX, method='knnImpute')

#missImp

#brecanX <- predict(missImp, brecanX)

#dim(brecanX)



## check for missing data--after impute...: fixed ---add plot

#sum(is.na(brecanX)) # any(is.na(brecanX))

# none ---

#



## duplicates

dup_cols = sum(duplicated(brecanX)==TRUE)

dup_cols



## check for negative values

# checking negative values

neg_cols_count = 0

for (col in 1:ncol(brecanX)) {

  neg_cols_count = neg_cols_count + length(which(brecanX[,col] < 0))
```

```
}
pos_cols_count = 0
for (col in 1:ncol(brecanX)) {
  pos_cols_count = pos_cols_count + length(which(brecanX[,col] >= 0))
}

# checking negative values
neg_cols_count
pos_cols_count
pos_cols_count + neg_cols_count
nrow(brecanX)*ncol(brecanX)
dim(brecanX)

## impute by Yeo-Johnson or manual fix [add positive number]
# none
## get CONTINUOUS, CATEGORICAL predictors
# no categorical data
#
#brecanXcat <- brecanX[,] #brecanXcat <- brecanX[0,]
brecanXcont <- brecanX[,]
dim(brecanXcont)
#dim(brecanXcat)

## check for skewness: ...
install.packages("e1071")
```

```
library(e1071)
skew_vals <- apply(brecanXcont, 2, skewness)
skew_vals



#dev.off()
## check for skewness: ...draw histogram
chart_div = 10
for (col in 1:ncol(brecanXcont)) {
  if (col%%chart_div==1) par(mfrow = c(3,4), pin=c(2,1))
  hist(brecanXcont[,col],
       main=colnames(brecanXcont[col]),
       xlab=paste("brecanXcont$",colnames(brecanXcont[col])),
       col=8, border=0)
  mtext(paste('Skewness: ', round(skewness(brecanXcont[,col]), 4)), cex=0.6)
#grid(col = 'lightgrey', lty = 3, lwd = par("lwd"), equilogs = TRUE) #nx = NULL, ny =
nx,
}

## check for outliers: ...draw boxplots
for (col in 1:ncol(brecanXcont)) {
  if (col%%chart_div==1) par(mfrow = c(3,4), pin=c(2,1))
  boxplot(brecanXcont[,col], main=colnames(brecanXcont[col]),
          col=8, boxwex=0.7)
  mtext(paste('Outliers:', length(boxplot.stats(brecanXcont[,col])$out)), cex=0.7)
}
```

```
# total outliers

tot_out = 0

for (col in 1:ncol(brecanXcont)){

  tot_out = tot_out + length(boxplot.stats(brecanXcont[,col])$out)

}

print (tot_out)


#### 3. Data Transformation

### CATEGORICAL data

# remove degenerate [nzr] predictors


#install.packages("caret")

#library(caret)

#nzv_brecanXcat <- nearZeroVar(brecanXcat)

#

## nzv cols to remove

#length(nzv_brecanXcat)

#colnames(brecanXcat[,nzv_brecanXcat])

#

## remove nzv...

#if (length(nzv_brecanXcat) > 0){

#  brecanXcatTrans <- brecanXcat[,-nzv_brecanXcat]

#}

#
```

```
## after nzv cols removed

#dim(brecanXcat)

#dim(brecanXcatTrans)



#### if categorical predictors exist... ENCODE dummy variables...

# no categorical variables/predictors --> no dummy

#


## dummy encode template

#dummyTrans <- dummyVars("~[col1] + [col2]", data = brecanXcatTrans, fullRank = TRUE)

#brecanXcatTrans <- data.frame(predict(dummyTrans, newdata = brecanXcatTrans))

#dim(brecanXcatTrans)

#head(brecanXcatTrans)




#### CONTINUOUS vars

## center, scale, BoxCox, spatialSign #Yeo Johnson

# skewness, outliers, numerical stability ---


install.packages("caret")

library(caret)

preprocTrans <- preProcess(brecanXcont, method=c('center','scale', 'BoxCox',
'spatialSign'))

preprocTrans
```

```
brecanXcontTrans <- predict(preprocTrans, brecanXcont)

dim(brecanXcontTrans)


## check histogram after trans --- skewness etc.

#

chart_div = 10

for (col in 1:ncol(brecanXcontTrans)) {

  if (col%%chart_div==1) par(mfrow = c(3,4), pin=c(2,1))

  hist(brecanXcontTrans[,col],

       main=colnames(brecanXcontTrans[col]),

       xlab=paste("brecanX$",colnames(brecanXcontTrans[col])),

       col=14, border=0)

  mtext(paste('Skewness after Trans: ', round(skewness(brecanXcontTrans[,col]), 4)),
cex=0.6)

#grid(col = 'lightgrey', lty = 3, lwd = par("lwd"), equilogs = TRUE) #nx = NULL, ny =
nx,

}


## check for outliers: ...draw boxplots

for (col in 1:ncol(brecanXcontTrans)) {

  if (col%%chart_div==1) par(mfrow = c(3,4), pin=c(2,1))

  boxplot(brecanXcontTrans[,col], main=colnames(brecanXcontTrans[col]),

          col=0, border=1, boxwex=0.8)

  mtext(paste('Outliers after Trans:',
length(boxplot.stats(brecanXcontTrans[,col])$out)), cex=0.7)

}
```

```
# total outliers

tot_out_trans = 0

for (col in 1:ncol(brecanXcontTrans)){

  tot_out_trans = tot_out_trans + length(boxplot.stats(brecanXcontTrans[,col])$out)

}

print (tot_out_trans)




### -- option 2 ------

#

#chart_div = 15

#for (col in 1:ncol(brecanXcontTrans)) {

#  if (col%%chart_div==1) par(mfrow = c(5,3), pin=c(3,1))

#  col_index = substring(colnames(brecanXcontTrans[col]), first=2)

#  col_alias_name = col_alias_o[strtoi(col_index)]

#

#  hist(brecanXcontTrans[,col],

#       main=paste(paste(paste(colnames(brecanXcontTrans[col]),":"),col_alias_name)),

#       xlab=paste("x_trans_pca_bre_can_dgn$",colnames(brecanXcontTrans[col])),

#       col="#e177bc", border=0)

#  mtext(paste('skewness: ', round(skewness(brecanXcontTrans[,col]), 8)), cex=0.6)

#}
```

```
## remove too high correlated predictors, compare with PCAs [feature reduction]

install.packages(c('corrplot','RANN'))

library(RANN)

library(caret)

library(corrplot)


dev.off()

par(mfrow = c(1,1))

## “circle”, “square”, “ellipse”, “number”, “shade”, “color”, “pie”

corrplot(cor(brecanXcontTrans), method = "color")


# find high corr data := .75, .8, .85, .9

corThresh <- .8

tooHigh <- findCorrelation(cor(brecanXcontTrans), corThresh) #pred to remove

length(tooHigh)

(dim(brecanXcontTrans)[2]-length(findCorrelation(cor(brecanXcontTrans), corThresh)))
#pred to retain

corrPred <- names(brecanXcontTrans)[tooHigh]

names(brecanXcontTrans)[tooHigh]  # too high predictors...


# remove high corr data

brecanXcontTransCorr <- brecanXcontTrans[, -tooHigh]

dim(brecanXcontTransCorr)
```

```
# after removing high corr pred

par(mfrow = c(1,1))

corrplot(cor(brecanXcontTransCorr), method = "color")


dim(brecanXcont)

dim(brecanXcontTrans)

dim(brecanXcontTransCorr)




## confirm with PCA ??? instead of removing high corr pred manually

# Applying Transformation -- PCA

#

pca_brecanXcontTrans <- preProcess(brecanXcontTrans, method = c("pca")) #"center",
"scale", "BoxCox",

pca_brecanXcontTrans


# Apply the transformations:

brecanXcontTransPCA <- predict(pca_brecanXcontTrans, brecanXcontTrans)  # 10 PCs,
default value: C = 95%


dim(brecanXcontTrans)

dim(brecanXcontTransPCA)

head(brecanXcontTransPCA)

str(brecanXcontTransPCA)
```

```
# ----------------------------------------

# after pca... dimension reduction histogram

#dev.off()

chart_div = 11

for (col in 1:ncol(brecanXcontTransPCA)) {

  if (col%%chart_div==1) par(mfrow = c(3,4), pin=c(2,1))

  hist(brecanXcontTransPCA[,col],

       main=colnames(brecanXcontTransPCA[col]),

       xlab=paste("brecanTransPCA$",colnames(brecanXcontTransPCA[col])),

       col=4, border=0, cex=0.7)

  #mtext(paste('Skewness: ', round(skewness(brecanXcontTransPCA[,col]), 4)), cex=0.6)

  #grid(col = 'lightgrey', lty = 3, lwd = par("lwd"), equilogs = TRUE) #nx = NULL, ny
= nx,

  #'#e177bc'

  }

#----------------------------------------




#### join [categorical] + [continuous]

# cont... either [brecanXcontTransCorr] or [brecanXcontTransPCA]

# cat... brecanXcatTrans


dim(brecanXcontTransCorr)

#dim(brecanXcontTransPCA)
```

```
#dim(brecanXcatTrans) # brecanXcatTrans <- brecanX[0,0]

#brecanX <- cbind(brecanXcontTransCorr, brecanXcatTrans)


##brecanX <- brecanXcontTransPCA[,]

brecanX <- brecanXcontTransCorr[,]


#Predictors retained for modeling after data transformation

dim(brecanX)

length(brecanY)




#### 4. Data Partition --- Spending data

## sampling methods... data with strata [categories... mean, error, worst]

# partition data ---- random/stratified sampling

set.seed(100)

trainRows <- createDataPartition(brecanY, p=0.8, list=FALSE)


# train

brecanX_train <- brecanX[trainRows,]

brecanY_train <- brecanY[trainRows]


# test

brecanX_test <- brecanX[-trainRows,]

brecanY_test <- brecanY[-trainRows]
```

```
# partioned dataset into 80% training and 20% testing...

dim(trainRows)

dim(brecanX_train)

length(brecanY_train)

dim(brecanX_test)

length(brecanY_test)



## for pretty large ds... Random sampling using sample function

#set.seed(101)  # Set seed for reproducibility

#sample_indices <- sample(nrow(brecanX), size = 0.8 * nrow(brecanX))

## Create training and testing sets

#train_X <- brecanX[sample_indices, ]

#test_X <- brecanX[-sample_indices, ]

#train_Y <- brecanY[sample_indices]

#test_Y <- brecanY[-sample_indices]

#cat("Training set:\n")

#dim(train_X)

#dim(test_X)


## RE-SAMPLING method for the modeling

# since dataset is not too large... 569 obs.

# to obtain a more robust estimate of the model performance

# k-fold: 10-fold cv with 5x repeat
```

```
##### BUILDING MODELS

#### BUILDING MODELS... ## Linear Classification Models


set.seed(400)

ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 5,

                     # summaryFunction = twoClassSummary, #defaultSummary

                      classProbs = TRUE,

                      savePredictions = TRUE)


### LGOCV- repeated training/test splits (25 reps, 75%) ## Leave Group Out cross-validation

#ctrl <- trainControl(method = "LGOCV",

#                      summaryFunction = twoClassSummary,

#                      classProbs = TRUE,

#                      ##index = list(simulatedTest[,1:4]),

#                      savePredictions = TRUE)







############ 1. Logistic Regression ###############


install.packages("MLmetrics")

library(MLmetrics)

library(caret)
```

```
#tuning parameter: ?? ...none

set.seed(410)
#lrFit <- caret::train(brecanX_train, brecanY_train,
#                      method = "multinom", metric = "ROC",
#                      #preProcess = c("center","scale"),
#                      trControl = ctrl,
#                      trace = FALSE)

lrFit <- caret::train(brecanX_train, brecanY_train,
                      method = "glm",
                      metric = "Kappa",
                      trControl = ctrl)

lrFit

## The predict fxn... no need to manually specify the shrinkage amount
#predict on test data
lrPredTest <- predict(lrFit, newdata = brecanX_test)
lrPredTest #sum((lrPredTest == hepabrecanY_test)==TRUE)/length(hepabrecanY_test)

postResample(lrPredTest, brecanY_test)

lrConfMatrix <- confusionMatrix(data = lrPredTest, reference = brecanY_test)
lrConfMatrix
```

```
# Predict probs on the test set

lrProbPred <- predict(lrFit, brecanX_test, type = "prob")

lrProbPred_TgtRes <- lrProbPred$M #lrProbPred_TgtRes <- lrProbPred[, 2]


# Create a ROC object

library("pROC") ##lrROC <- multiclass.roc(response = brecanY_test, predictor =
lrProbPred_TgtRes, levels=rev(levels(brecanY_test)))

lrROC <- roc(response = brecanY_test, predictor = lrProbPred_TgtRes,
levels=rev(levels(brecanY_test)))

print(lrROC)


# find AUC

lrAUC <- auc(lrROC)

print(lrAUC)


## Plotting ------------------------------------------------------------

# tuning plot... no tuning

plot(lrFit, legacy.axes = TRUE, col=14, lwd=1.2,

     main="Logistic Regression Model")


# tuning heatmap

plot(lrFit, plotType = "level", legacy.axes = TRUE,

     main="lr Model: Tuning Heat-map")


#ROC curve
```

```
par(mfrow = c(1,1))
plot(x = lrROC, #$predictor, y = lrROC$response,
     main = "Logistic Regression Model: ROC Curve",
     xlab = "False Positive Rate (1 - Specificity)",
     ylab = "True Positive Rate (Sensitivity)",
     col = "darkorange", # c("darkorange", "darkgreen", "darkred"),
     lty = 1:3, lwd=1.5)



#legend("topright", legend = levels(brecanY_test), \
#       col = c("darkorange", "darkgreen", "darkred"),
#       pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
#       inset = c(0, -0.08), xpd = TRUE)









############ 2. Linear Discriminant Analysis #############



# install.packages("MLmetrics")
library(MLmetrics)
library(caret)
```

```
# tuning params: none


set.seed(420)
ldaFit <- caret::train(brecanX_train, brecanY_train,
                method = "lda", metric = "Kappa",
                #preProcess = c("center","scale"),
                trControl = ctrl,
                trace = FALSE)


ldaFit


#predict on test data
ldaPredTest <- predict(ldaFit, newdata = brecanX_test)
ldaPredTest #sum((ldaPredTest == hepabrecanY_test)==TRUE)/length(hepabrecanY_test)


postResample(ldaPredTest, brecanY_test)


ldaConfMatrix <- confusionMatrix(data = ldaPredTest, reference = brecanY_test)
ldaConfMatrix


# Predict probs on the test set
ldaProbPred <- predict(ldaFit, brecanX_test, type = "prob")
ldaProbPred_TgtRes <- ldaProbPred$M #ldaProbPred_TgtRes <- ldaProbPred[, 2]


# Create a ROC object
```

```
library("pROC") ##ldaROC <- multiclass.roc(response = brecanY_test, predictor =
ldaProbPred_TgtRes, levels=rev(levels(brecanY_test)))

ldaROC <- roc(response = brecanY_test, predictor = ldaProbPred_TgtRes,
levels=rev(levels(brecanY_test)))

print(ldaROC)


# find AUC

ldaAUC <- auc(ldaROC)

print(ldaAUC)


## Plotting ----------------------------------------------------------

# tuning plot... no tuning ****

plot(ldaFit, legacy.axes = TRUE, col=14, lwd=1.3,

     main="Linear Discriminant Analysis Model")


# tuning heatmap

plot(ldaFit, plotType = "level", legacy.axes = TRUE, main="Linear Discriminant
Analysis Model: Tuning Heat-map")


#ROC curve

plot(x = ldaROC, #$predictor, y = ldaROC$response,

     main = "Linear Discriminant Analysis Model: ROC Curve",

     xlab = "False Positive Rate (1 - Specificity)",

     ylab = "True Positive Rate (Sensitivity)",

     col = 4, # c("violet", "darkgreen", "darkred"),

     lty = 1:3, lwd=1.5)
```

```
#legend("topright", legend = levels(brecanY_test),
#       col = c("darkorange", "darkgreen", "darkred"),
#       pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
#       inset = c(0, -0.08), xpd = TRUE)




############ 3. PLS Discriminant Analysis (PLSDA) #############

install.packages(c("MLmetrics", "glmnet", "pamr", "rms", "sparseLDA", "subselect",
"MASS", "pls", "pROC"))

library(pls)
library(MLmetrics)
library(caret)



# tuning param: components retained...
plsdaGrid <- expand.grid(.ncomp = 1:18)

#set.seed(400)
#ppctrl <- trainControl(summaryFunction = twoClassSummary,
#                       classProbs = TRUE)
```

```
#ppctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3,
#                       #summaryFunction = twoClassSummary, #defaultSummary
#                       classProbs = TRUE,
#                       savePredictions = TRUE)

set.seed(430)
plsdaTune <- caret::train(brecanX_train, brecanY_train,
                          method = "pls", metric = "Kappa",
                          preProcess = c("center","scale"),
                          tuneGrid = plsdaGrid,
                          trControl = ctrl,
                          trace = FALSE)

plsdaTune

#predict on test data
plsdaPredTest <- predict(plsdaTune, newdata = brecanX_test)
plsdaPredTest #sum((plsdaPredTest ==
hepabrecanY_test)==TRUE)/length(hepabrecanY_test)

postResample(plsdaPredTest, brecanY_test)

plsdaConfMatrix <- confusionMatrix(data = plsdaPredTest, reference = brecanY_test)
plsdaConfMatrix
```

```
# Predict probs on the test set

plsdaProbPred <- predict(plsdaTune, brecanX_test, type = "prob")

plsdaProbPred_TgtRes <- plsdaProbPred$M #plsdaProbPred_TgtRes <- plsdaProbPred[, 3]


# Create a ROC object

library("pROC") ##plsdaROC <- multiclass.roc(response = brecanY_test, predictor =
plsdaProbPred_TgtRes, levels=rev(levels(brecanY_test)))

plsdaROC <- roc(response = brecanY_test, predictor = plsdaProbPred_TgtRes,
levels=rev(levels(brecanY_test)))

print(plsdaROC)


# find AUC

plsdaAUC <- auc(plsdaROC)

print(plsdaAUC)


## Plotting ------------------------------------------------------------

# tuning plot

plot(plsdaTune, legacy.axes = TRUE, col=14, lwd=1.3,

     main="PLS Discriminant Analysis Model")


# tuning heatmap

plot(plsdaTune, plotType = "level", legacy.axes = TRUE,

     main="PLS Discriminant Analysis Model: Tuning Heat-map")


#ROC curve

plot(x = plsdaROC, #$predictor, y = plsdaROC$response,
```

```
     main = "plsda Model: ROC Curve",
     xlab = "False Positive Rate (1 - Specificity)",
     ylab = "True Positive Rate (Sensitivity)",
     col = 3, #c("darkorange", "darkgreen", "darkred"),
     lty = 1:3, lwd=1.5)

#legend("topright", legend = levels(brecanY_test),
#       col = c("darkorange", "darkgreen", "darkred"),
#       pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
#       inset = c(0, -0.08), xpd = TRUE)




############ 4. Penalized Models ##########

## The family argument is related to the distribution of the outcome
## For two classes, use family="binomial" corresponds to logistic regression,
## For three or more classes, use family="multinomial" is appropriate.
## glmnet defaults this parameter to alpha = 1, corresponding to a
## complete lasso penalty.

library(glmnet)
library(MLmetrics)
library(caret)
```

```
# tuning param:
#glmnGrid <- expand.grid(.alpha = seq(0, 1, by=0.1), # c(0, .1, .2, .4, .6, .8, 1),
#                        .lambda = seq(.01, .2, length = 30))

glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .3, .4, .6, .8, 1),
                        .lambda = seq(.01, .2, length = 10))

set.seed(440)
glmnTune <- caret::train(brecanX_train, brecanY_train,
                         method = "glmnet", metric = "Kappa",
                         preProcess = c("center","scale"),
                         tuneGrid = glmnGrid,
                         trControl = ctrl,
                         trace = FALSE)

glmnTune

#predict on test data
glmnPredTest <- predict(glmnTune, newdata = brecanX_test)
glmnPredTest #sum((glmnPredTest == hepabrecanY_test)==TRUE)/length(hepabrecanY_test)

postResample(glmnPredTest, brecanY_test)
```

```
glmnConfMatrix <- confusionMatrix(data = glmnPredTest, reference = brecanY_test)

glmnConfMatrix


# Predict probs on the test set

glmnProbPred <- predict(glmnTune, brecanX_test, type = "prob")

glmnProbPred_TgtRes <- glmnProbPred$M #glmnProbPred_TgtRes <- glmnProbPred[, 3]


# Create a ROC object

library("pROC") ##glmnROC <- multiclass.roc(response = brecanY_test, predictor =
glmnProbPred_TgtRes, levels=rev(levels(brecanY_test)))

glmnROC <- roc(response = brecanY_test, predictor = glmnProbPred_TgtRes,
levels=rev(levels(brecanY_test)))

print(glmnROC)


# find AUC

glmnAUC <- auc(glmnROC)

print(glmnAUC)


## Plotting ------------------------------------------------------------

# tuning plot

plot(glmnTune, legacy.axes = TRUE, lwd=1.3,

     main="Penalized Model")


# tuning heatmap

plot(glmnTune, plotType = "level", legacy.axes = TRUE, main="Penalized Model: Tuning
Heat-map")
```

```
#ROC curve
plot(x = glmnROC, #$predictor, y = glmnROC$response,
     main = "Penalized Model: ROC Curve",
     xlab = "False Positive Rate (1 - Specificity)",
     ylab = "True Positive Rate (Sensitivity)",
     col =  2, #"darkred", #c("darkorange", "darkgreen", "darkred"),
     lty = 1:3, lwd=1.5)

#legend("topright", legend = levels(brecanY_test),
#       col = c("darkorange", "darkgreen", "darkred"),
#       pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
#       inset = c(0, -0.08), xpd = TRUE)




########### Nearest Shrunken Centroids ###########
library(caret)

# tuning param: threshold...
nscGrid <- data.frame(.threshold = seq(0, 4, by=0.1))

set.seed(470)
nscTune <- caret::train(x = brecanX_train, y = brecanY_train,
```

```
                        method = "pam", metric = "Kappa",
                        tuneGrid = nscGrid,
                        preProc = c("center", "scale"),
                        trControl = ctrl)
nscTune

#predict on test data... no need to manually specify the shrinkage amount
nscPredTest <- predict(nscTune, newdata = brecanX_test)
nscPredTest #sum((nscPredTest == hepabrecanY_test)==TRUE)/length(hepabrecanY_test)

postResample(nscPredTest, brecanY_test)

nscConfMatrix <- confusionMatrix(data = nscPredTest, reference = brecanY_test)
nscConfMatrix

# Predict probs on the test set
nscProbPred <- predict(nscTune, brecanX_test, type = "prob")
nscProbPred_TgtRes <- nscProbPred$M #nscProbPred_TgtRes <- nscProbPred[, 3]

# Create a ROC object
library("pROC") ##nscROC <- multiclass.roc(response = brecanY_test, predictor =
nscProbPred_TgtRes, levels=rev(levels(brecanY_test)))
nscROC <- roc(response = brecanY_test, predictor = nscProbPred_TgtRes,
levels=rev(levels(brecanY_test)))
print(nscROC)
```

```
# find AUC
nscAUC <- auc(nscROC)
print(nscAUC)

## Plotting ----------------------------------------------------------
# tuning plot
plot(nscTune, legacy.axes = TRUE, col=14, lwd=1.3,
     main="Nearest Shrunken Centroids Model")

# tuning heatmap
plot(nscTune, plotType = "level", legacy.axes = TRUE,
     main="Nearest Shrunken Centroids Model: Tuning Heat-map")

#ROC curve
plot(x = nscROC, #$predictor, y = nscROC$response,
     main = "Nearest Shrunken Centroids Model: ROC Curve",
     xlab = "False Positive Rate (1 - Specificity)",
     ylab = "True Positive Rate (Sensitivity)",
     col = 5, #c("darkorange", "darkgreen", "darkred"),
     lty = 1:3, lwd=1.5)

#legend("topright", legend = levels(brecanY_test),
#       col = c("darkorange", "darkgreen", "darkred"),
#       pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
#       inset = c(0, -0.08), xpd = TRUE)
```

```
## The predictors function will list the predictors used in the prediction equation

predictors(nscTune)


## variable importance based on the distance between the class centroid and the
overall centroid:

varImp(nscTune, scale = FALSE)









## ************************************

## d) For the optimal model for the biological predictors, what are

## the top five important ## predictors?



impVals=varImp(knnTune)

str(impVals)

impVals$importance


impVals
```

```
# top 5.


plot(impVals,
     top = 5,
     scales = list(y = list(cex = .8)),
     col = 14,
     main="Optimal Model: KNN Model with Top 5 Predictors Importance"
)
```

```
## Non-Linear  Classification Models


##### dataset


dim(trainRows)


dim(brecanX_train)
length(brecanY_train)
dim(brecanX_test)
length(brecanY_test)



##### BUILDING MODELS


set.seed(500)
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 5,
                     # summaryFunction = twoClassSummary, #defaultSummary
                     classProbs = TRUE,
                     savePredictions = TRUE)



## Non-Linear  Classification Models



## 1.1 Quadratic Discriminant Analysis -- QDA
## ************************************************************
install.packages("qda")
library(qda)
library(caret)
```

```
# has no tuning...
set.seed(503)
qdaFit <- caret::train(x = brecanX_train, y = brecanY_train,
                       method = "qda", metric = "Kappa",
                       #tuneGrid = rdaTuneGrid,
                       #preProc = c("center", "scale"),
                       trControl = ctrl)
qdaFit

#predict on test data
qdaPredTest <- predict(qdaFit, newdata = brecanX_test)
qdaPredTest

postResample(qdaPredTest, brecanY_test)

qdaConfMatrix <- confusionMatrix(data = qdaPredTest, reference = brecanY_test)
qdaConfMatrix

# Predict probs on the test set
# qdaProbPred <- as.numeric(predict(qdaFit, brecanX_test, type = "prob")$M)
#$posterior
qdaProbPred <- predict(qdaFit, brecanX_test, type = "prob")
qdaProbPred_TgtRes <- qdaProbPred$M #qdaProbPred_TgtRes <- qdaProbPred[, 3]

# Create a ROC object
library("pROC")
qdaROC <- roc(response = brecanY_test,
              predictor = qdaProbPred_TgtRes,
              levels=rev(levels(brecanY_test)))
```

```
print(qdaROC)

# find AUC
qdaAUC <- auc(qdaROC)
print(qdaAUC)



## Plotting...
# ------------------------------------------------------------

# plot model... verify optimal pt.
plot(qdaFit,
     main="Quadratic Discriminant Analysis (QDA) Model",
     col=2,lwd=1.3)

plot(qdaFit, plotType = "level",
     main="QDA Model per Tuning Parameters")



#  ROC curve
plot(x = qdaROC, #$predictor, y = qdaROC$response,
     main = "QDA Model: ROC Curve",
     xlab = "False Positive Rate (1 - Specificity)",
     ylab = "True Positive Rate (Sensitivity)",
     col = "deeppink",
     lty = 1:3, lwd=1.5)
```

```
## 1.2 Regularized Discriminant Analysis -- RDA
## ************************************************************
install.packages("rda")
library(rda)
library(caret)



rdaTuneGrid <- expand.grid(lambda = seq(0, 1, by = 0.2),
                           gamma = seq(0, 1, by = 0.1))


set.seed(505)
rdaTune <- caret::train(x = brecanX_train, y = brecanY_train,
                        method = "rda", metric = "Kappa",
                        tuneGrid = rdaTuneGrid,
                        preProc = c("center", "scale"),
                        trControl = ctrl)
rdaTune


#predict on test data
rdaPredTest <- predict(rdaTune, newdata = brecanX_test)
rdaPredTest


postResample(rdaPredTest, brecanY_test)


rdaConfMatrix <- confusionMatrix(data = rdaPredTest, reference = brecanY_test)
rdaConfMatrix
```

```
# Predict probs on the test set

# rdaProbPred <- as.numeric(predict(rdaTune, brecanX_test, type = "prob")$M)
#$posterior

rdaProbPred <- predict(rdaTune, brecanX_test, type = "prob")

rdaProbPred_TgtRes <- rdaProbPred$M #rdaProbPred_TgtRes <- rdaProbPred[, 3]


# Create a ROC object

library("pROC")

rdaROC <- roc(response = brecanY_test,

              predictor = rdaProbPred_TgtRes,

              levels=rev(levels(brecanY_test)))


print(rdaROC)


# find AUC

rdaAUC <- auc(rdaROC)

print(rdaAUC)



## Plotting...

# ----------------------------------------------------------


# plot model... verify optimal pt.

plot(rdaTune,

     main="Regularized Discriminant Analysis (RDA) Model",

     lwd=1.3)


plot(rdaTune, plotType = "level",

     main="RDA Model per Tuning Parameters")
```

```
#  ROC curve
  plot(x = rdaROC, #$predictor, y = rdaROC$response,
       main = "RDA Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
       col = "darkgreen",
       lty = 1:3, lwd=1.5)
```

```
## 1.3 Mixture Discriminant Analysis -- MDA
## ************************************************************
library(mda)
library(caret)

## tuning parameters: no. of distributions = subclasses
#mdaCtrl <- trainControl(summaryFunction = multiClassSummary, classProbs = TRUE)

#the potential subpopulations or clusters within each class
mdaTuneGrid = expand.grid(.subclasses = 1:3) # 3 >> cntDist

set.seed(510)
mdaTune <- caret::train(x = brecanX_train, y = brecanY_train,
                        method = "mda", metric = "Kappa",
                        tuneGrid = mdaTuneGrid,
                        #preProc = c("center", "scale"),
```

```
                        trControl = ctrl)
mdaTune


#predict on test data
mdaPredTest <- predict(mdaTune, newdata = brecanX_test)
mdaPredTest


postResample(mdaPredTest, brecanY_test)


mdaConfMatrix <- confusionMatrix(data = mdaPredTest, reference = brecanY_test)
mdaConfMatrix


# Predict probs on the test set
# mdaProbPred <- as.numeric(predict(mdaTune, brecanX_test, type = "prob")$M)
#$posterior
mdaProbPred <- predict(mdaTune, brecanX_test, type = "prob")
mdaProbPred_TgtRes <- mdaProbPred$M #mdaProbPred_TgtRes <- mdaProbPred[, 3]



# Create a multiclass ROC curve
library(pROC)
mdaROC <- roc(response = brecanY_test,
                          predictor = mdaProbPred_TgtRes,
                          levels=rev(levels(brecanY_test)))


print(mdaROC)


# find AUC
mdaAUC <- auc(mdaROC)
print(mdaAUC)
```

```
## Plotting...
# ----------------------------------------------------------

# plot model... verify optimal pt.
plot(mdaTune,
     main="Mixture Discriminant Analysis (MDA) Model",
     col = 2, lwd=1.3)

# tuning heatmap... 2 params needed
plot(mdaTune, plotType = "level", legacy.axes = TRUE,
     main="Mixture Discriminant Analysis (MDA) Model: Tuning Heat-map")

#ROC curve
tryCatch({

  plot(x = mdaROC, #$predictor, y = mdaROC$response,
       main = "MDA Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
       col = "darkorange", # c("darkorange", "darkgreen", "red"),
       lty = 1:3, lwd=1.5)

  #legend("topright", legend = levels(brecanY_test),
  #       col = c("darkorange", "darkgreen", "red"),
  #       pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
  #       inset = c(0, -0.08), xpd = TRUE)

},
```

```
error = function(e) {
  print(paste("Error:", e))
})
```

```
## 2. Neural Networks model -- NNet
## ************************************************************
## R packages... nnet, RSNNS, qrnn, and neuralnet
install.packages("nnet")
library(nnet)
library(caret)

## tuning parmeters: size[...], decay[...]

##nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
#nnetGrid <- expand.grid(.size = seq(1, 10, by = 2), .decay = c(0.1, 1, 5, 10))

nnetGrid <- expand.grid(.size = 1:10,
                        .decay = c(0, .1, 1, 2, 4, 6, 8, 11))


#numWts <- (maxSize * (4 + 1)) + ((maxSize+1)*2) ## 4 is the number of predictors ##
((p+1)*H) + ((H+1)*C)
maxSize <- max(nnetGrid$.size) ## maxSize --> H, hinge fxn
cntPred = dim(brecanX_train)[2]
cntClass = length(levels(brecanY_train))
numWts <- (maxSize * (cntPred + 1)) + ((maxSize + 1) * cntClass) ## 4 is the number
of predictors
```

```
#nnetCtrl <- trainControl(summaryFunction = defaultSummary,classProbs = TRUE)
#twoClassSummary

set.seed(330)

nnetTune <- caret::train(x = brecanX_train, y = brecanY_train,

                         method = "nnet", metric = "Kappa",

                         maxit = 2000, MaxNWts = numWts,

                         trace = FALSE,

                         tuneGrid = nnetGrid,

                         preProcess = c("center", "scale", "spatialSign"),

                         trControl = ctrl)

nnetTune


#predict on test data

nnetPredTest <- predict(nnetTune, newdata = brecanX_test)

nnetPredTest


postResample(nnetPredTest, brecanY_test)


nnetConfMatrix <- confusionMatrix(data = nnetPredTest, reference = brecanY_test)

nnetConfMatrix


# Predict probs on the test set

# nnetProbPred <- as.numeric(predict(nnetTune, brecanX_test, type = "prob")$M)
#$posterior

nnetProbPred <- predict(nnetTune, brecanX_test, type = "prob")

nnetProbPred_TgtRes <- nnetProbPred$M #nnetProbPred_TgtRes <- nnetProbPred[, 3]


# Create a ROC object

library("pROC")

nnetROC <- roc(response = brecanY_test,

                           predictor = nnetProbPred_TgtRes,
```

```
                          levels=rev(levels(brecanY_test)))

print(nnetROC)

# find AUC
nnetAUC <- auc(nnetROC)
print(nnetAUC)



## Plotting...
# ----------------------------------------------------------

# plot model... verify optimal pt.

plot(nnetTune,
     main="NNet Model",
     lwd=1.3)

#grid(col = 'lightgrey', lty = 3, lwd = par("lwd"), equilogs = TRUE)

plot(nnetTune, plotType = "level",
     main="NNet Model: Tuning Heat-map")

#  ROC curve
tryCatch({

  plot(x = nnetROC, #$predictor, y = nnetROC$response,
       main = "NNet Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
```

```
       col = 3, #c("orange", "darkgreen", "red"),
       lty = 1:3, lwd=1.5)


},
error = function(e) {
  print(paste("Error:", e))
})
```

```
## 3. Flexible Discriminant Analysis model -- FDA
## ************************************************************

library(MASS)
library(mda)
library(earth)
library(caret)

## tuning paramenters: degree [...], nprune [terms retained]...
## similar to the MARS model
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:30)

#fdaCtrl <- trainControl(method = "cv")
fdaCtrl <- trainControl(method = "cv", number = 10, #repeats = 5,
                        #summaryFunction = twoClassSummary,
```

```
                        classProbs = TRUE, savePredictions = TRUE)


set.seed(340)
fdaTune <- caret::train(x = brecanX_train, y = brecanY_train,
                        method = "fda", metric = "Kappa",
                        tuneGrid = marsGrid,
                        #preProcess = c("center", "scale"),
                        trControl = fdaCtrl)

fdaTune

#predict on test data
fdaPredTest <- predict(fdaTune, newdata = brecanX_test)
fdaPredTest

postResample(fdaPredTest, brecanY_test)

fdaConfMatrix <- confusionMatrix(data = fdaPredTest, reference = brecanY_test)
fdaConfMatrix

# Predict probs on the test set
# fdaProbPred <- as.numeric(predict(fdaTune, brecanX_test, type = "prob")$M)
#$posterior
fdaProbPred <- predict(fdaTune, brecanX_test, type = "prob")
fdaProbPred_TgtRes <- fdaProbPred$M #fdaProbPred_TgtRes <- fdaProbPred[, 3]

# Create a ROC object
library("pROC")
fdaROC <- roc(response = brecanY_test,
```

```
                          predictor = fdaProbPred_TgtRes,
                          levels=rev(levels(brecanY_test)))

print(fdaROC)

# find AUC
fdaAUC <- auc(fdaROC)
print(fdaAUC)



## Plotting...
# ----------------------------------------------------------

# plot model... verify optimal pt.
## degree = 2 and nprune = 3.
optMdlDeg = 1 #optimal model from training
optMdlTerm = 18 #optimal model from training
plot(fdaTune,
     main=paste(paste(paste("FDA Model: Degree =", optMdlDeg),
                      " and nprune ="), optMdlTerm),
     lwd=1.3, col=c(2,4))



#  ROC curve
tryCatch({
  plot(x = fdaROC, #$predictor, y = fdaROC$response,
       main = "FDA Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
       col = 3, #c("orange", "darkgreen", "red"),
```

```
       lty = 1:3, lwd=1.5)


},
error = function(e) {
  print(paste("Error:", e))
})



plot(fdaTune, plotType = "level",
     main="FDA Model per Tuning Parameters")




## 4. Support Vector Machines model -- SVM
## ************************************************************

## R packages for SVM and other kernels: e1071, kernlab, klaR, and svmPath

library(MASS)
library(kernlab)
library(caret)

#ctrl <- trainControl(summaryFunction = defaultSummary, classProbs = TRUE)

## sigest estimates the range of values for the sigma parameter
sigmaRangeReduced <- sigest(as.matrix(brecanX_train))
```

```
## SVM (ksvm). The estimation is based upon the 0.1 and 0.9 quantile of ||x -x'||^2.

## Basically any value in between those two bounds will produce good results.

## Given a range of values for the "sigma" inverse width parameter in the Gaussian
Radial Basis kernel for use with SVM


## log 2 base

minAbsSSE <- -4

maxAbsSSE <- 8 # changed from 6 to 8

#svmRGridReduced <- expand.grid(.sigma = sigmaRangeReduced[1], .C = 2^(seq(-4, 6)))

svmRGridReduced <- expand.grid(.sigma = min(sigmaRangeReduced),

                               .C = 2^(seq(minAbsSSE, maxAbsSSE))) # minimum val of
||x -x'||^2


library(caret)

set.seed(350)

svmRTune <- caret::train(x = brecanX_train,

                         y = brecanY_train,

                         method = "svmRadial",

                         metric = "Kappa",

                         preProc = c("center", "scale"),

                         tuneGrid = svmRGridReduced,

                         fit = FALSE,

                         trControl = ctrl)

svmRTune


## When the outcome is a factor, the function automatically uses prob.model = TRUE.

## Other kernel functions can be defined via the kernel and kpar arguments.

#library(kernlab)


#predict on test data
```

```
svmRPredTest <- predict(svmRTune, newdata = brecanX_test)

svmRPredTest


postResample(svmRPredTest, brecanY_test)


svmConfMatrix <- confusionMatrix(data = svmRPredTest, reference = brecanY_test)

svmConfMatrix


# Predict probs on the test set

# svmRProbPred <- as.numeric(predict(svmRTune, brecanX_test, type = "prob")$M)
#$posterior

svmRProbPred <- predict(svmRTune, brecanX_test, type = "prob")

svmRProbPred_TgtRes <- svmRProbPred$M #svmRProbPred_TgtRes <- svmRProbPred[, 3]


# Create a ROC object

library("pROC")

svmRROC <- roc(response = brecanY_test,

                          predictor = svmRProbPred_TgtRes,

                          levels=rev(levels(brecanY_test)))


print(svmRROC)


# find AUC

svmRAUC <- auc(svmRROC)

print(svmRAUC)



## Plotting...

# ----------------------------------------------------------
```

```
# plot model... verify optimal pt.
plot(svmRTune,
     main= "SVM Model (with radial kernel)",
     lwd=1.3, col=2)



# multiclass ROC curve
tryCatch({
  plot(x = svmRROC, #$predictor, y = svmRROC$response,
       main = "SVM Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
       col = 4, #c("orange", "darkgreen", "red"),
       lty = 1:3, lwd= 1.5)



},
error = function(e) {
  print(paste("Error:", e))
})



# Needs at least 2 tuning parameters with multiple values
plot(svmRTune, plotType = "level",
     main="SVM Model per Tuning Parameters")
```

```
## 5. K-Nearest Neighbors model -- KNN
## ************************************************************

#ctrl <- trainControl(summaryFunction = twoClassSummary, classProbs = TRUE)

## tuning param: k components
#tuneGrid = data.frame(.k = c(4*(0:5)+1, 10*(0:5)+1, 20*(1:5)+1, 25*(2:9)+1,
50*(2:9)+1)), ## 21 is the best
knnGrid = data.frame(.k = 1:50)

library(caret)
set.seed(360)
knnTune <- caret::train(x = brecanX_train,
                        y = brecanY_train,
                        method = "knn",
                        metric = "Kappa",
                        preProc = c("center", "scale"),
                        tuneGrid = knnGrid,
                        trControl = ctrl)

knnTune

#predict on test data
knnPredTest <- predict(knnTune, newdata = brecanX_test)
knnPredTest

postResample(knnPredTest, brecanY_test)

knnConfMatrix <- confusionMatrix(data = knnPredTest, reference = brecanY_test)
knnConfMatrix
```

```
# Predict probs on the test set

# knnProbPred <- as.numeric(predict(knnTune, brecanX_test, type = "prob")$M)
#$posterior

knnProbPred <- predict(knnTune, brecanX_test, type = "prob")

knnProbPred_TgtRes <- knnProbPred$M #knnProbPred_TgtRes <- knnProbPred[, 3]


# Create a ROC object

library("pROC")

knnROC <- roc(response = brecanY_test,

                          predictor = knnProbPred_TgtRes,

                          levels=rev(levels(brecanY_test)))


print(knnROC)


# find AUC

knnAUC <- auc(knnROC)

print(knnAUC)



## Plotting...

# ----------------------------------------------------------


# plot model... verify optimal pt.

plot(knnTune,

     main= "KNN Model",

     lwd=1.3, col=2)



# Needs at least 2 tuning parameters with multiple values
```

```
plot(knnTune, plotType = "level",
     main="KNN Model per Tuning Parameters")



# multiclass ROC curve
tryCatch({
  plot(x = knnROC, #$predictor, y = knnROC$response,
       main = "KNN Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
       col = 5, #c("orange", "darkgreen", "red"),
       lty = 1:3, lwd=1.5)



},
error = function(e) {
  print(paste("Error:", e))
})



# Needs at least 2 tuning parameters with multiple values
plot(knnTune, plotType = "level",
     main="KNN Model per Tuning Parameters")



## 6. Naive Bayes model
## ************************************************************
```

```
# naiveBayes in the e1071 package and NaiveBayes in the klaR package. Both offer
Laplace corrections

# klaR package ~ flexible, uses conditional density estimates


install.packages("klaR")

library(klaR)


## Tuning parameters: fL (Laplace Correction), usekernel (Distribution Type)

# adjust (Bandwidth Adjustment)


# tuning params: no tuning param needed. fL takes care of nzv in cat vars

nbGrid <- data.frame(.fL = 2,

                     .usekernel = TRUE,

                     .adjust = TRUE)


library(klaR)

library(caret)

set.seed(370)

nbFit <- caret::train( x = brecanX_train,

                       y = brecanY_train,

                       method = "nb",

                       metric = "Kappa",

                       # preProc = c("center", "scale"),

                       tuneGrid = nbGrid,

                       trControl = ctrl)


nbFit


#predict on test data

nbPredTest <- predict(nbFit, newdata = brecanX_test)
```

```
nbPredTest


postResample(nbPredTest, brecanY_test)


nbConfMatrix <- confusionMatrix(data = nbPredTest, reference = brecanY_test)

nbConfMatrix


# Predict probs on the test set

# nbProbPred <- as.numeric(predict(nbFit, brecanX_test, type = "prob")$M)
#$posterior

nbProbPred <- predict(nbFit, brecanX_test, type = "prob")

nbProbPred_TgtRes <- nbProbPred$M #nbProbPred_TgtRes <- nbProbPred[, 3]


# Create a ROC object

library("pROC")

nbROC <- roc(response = brecanY_test,

                        predictor = nbProbPred_TgtRes,

                        levels=rev(levels(brecanY_test)))


print(nbROC)


# find AUC

nbAUC <- auc(nbROC)

print(nbAUC)


## Plotting...

# ----------------------------------------------------------

# plot model... verify optimal pt. --- no tuning param***

plot(nbFit,

     main= "Naive Bayes Model",
```

```
     lwd=1.3, col = 2)

# multiclass ROC curve
tryCatch({
  plot(x = nbROC, #$predictor, y = nbROC$response,
       main = "Naive Bayes Model: ROC Curve",
       xlab = "False Positive Rate (1 - Specificity)",
       ylab = "True Positive Rate (Sensitivity)",
       col = 6, #c("orange", "darkgreen", "red"),
       lty = 1:3, lwd=1.5)
 # legend("topright", legend = levels(brecanY_test),
 #        col = c("darkorange", "darkgreen", "red"),
 #        pch=15, horiz=TRUE, bty='n', cex=0.7, lwd = 1,
 #        inset = c(0, -0.08), xpd = TRUE)

},
error = function(e) {
  print(paste("Error:", e))
})

# Needs at least 2 tuning parameters with multiple values
plot(nbFit, plotType = "level",
     main="Naive Bayes Model per Tuning Parameters")

## ************************************
## d) For the optimal model for the biological predictors, what are
## the top five important ## predictors?


impVals=varImp(knnTune)
```

```
str(impVals)
impVals$importance


impVals


# top 5.
plot(impVals,
     top = 5,
     scales = list(y = list(cex = .8)),
     col = 14,
     main="Optimal Model: KNN Model with Top 5 Predictors Importance"
)
```