**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**

Abstract

[1.0 Background 3](#_Toc153403280)

[2.0 Exploring the Data – Structure and Definitions 4](#_Toc153403281)

[3.0 Data Preprocessing 6](#_Toc153403282)

[4.0 Spending the Data 15](#_Toc153403283)

[5.0 Building (Training) Models 16](#_Toc153403284)

[6.0 Choosing Best Models: Top Two (2) Models 17](#_Toc153403285)

[7.0 Summary, Recommendations and References 26](#_Toc153403286)

[Appendix 1: Data Exploration and Preprocessing Outputs 29](#_Toc153403287)

[Appendix 2: Building Models Outputs 34](#_Toc153403288)

[Appendix 3: R-Code 34](#_Toc153403289)

### **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]](#footnote-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]](#footnote-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.

### **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”.

A graph showing a pink and a red graph

Description automatically generated with medium confidence

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]](#footnote-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** |
| **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.

### **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.

A graph with numbers and a missing value

Description automatically generated

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.

A group of graphs showing different sizes

Description automatically generated with medium confidence

A group of graphs showing the different types of data

Description automatically generated with medium confidence

A group of graphs showing different sizes

Description automatically generated with medium confidence

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.

A group of diagrams with text

Description automatically generated with medium confidence

A group of black and white graphs

Description automatically generated

A group of black and white diagrams

Description automatically generated

**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.

A screenshot of a computer

Description automatically generated

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

### A blue and white squares with red text Description automatically generated

### **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.

|  |  |
| --- | --- |
| A group of pink graphs  Description automatically generated | A group of purple graphs  Description automatically generated with medium confidence |
| A group of purple graphs  Description automatically generated with medium confidence |  |

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

**A screenshot of a computer

Description automatically generated**

### **b. “spatialSign” Transformation**

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

A group of black and white lines

Description automatically generated

A group of black and white lines

Description automatically generated

A group of black lines

Description automatically generated with medium confidence

### **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.

### A close-up of a list Description automatically generated

**“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.

A close up of a grid

Description automatically generated

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.

A white background with black text

Description automatically generated

### **5.2 Linear Discriminant Analysis (LDA) Model**

No model tuning plot available — no tuning parameter used.

A screenshot of a computer

Description automatically generated

### **5.3 Partial Least Square Discriminant Analysis (PLSDA) Model**

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

A graph with a line

Description automatically generated

### **5.4 Penalized Model**

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

A graph of different colored lines

Description automatically generated

### **5.5 Nearest Shrunken Centroids (NSC) Model**

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

A graph with a line

Description automatically generated

### **5.6 Quadratic Discriminant Analysis (QDA) Model**

No model tuning plot available — no tuning parameter used.

A screenshot of a computer

Description automatically generated

### **5.7 Regularized Discriminant Analysis (RDA) Model**

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

A graph of a diagram

Description automatically generated with medium confidence

### **5.8 Mixture Discriminant Analysis (MDA) Model**

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

A graph with a red line

Description automatically generated

### **5.9 Flexible Discriminant Analysis (FDA) Model**

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

A graph of a graph

Description automatically generated

### **5.10 Neural Networks (NNet) Model**

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

A graph of a number of colored lines

Description automatically generated

### **5.11 Support Vector Machines (SVM) Model**

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

A graph with a line

Description automatically generated

### **5.12 K-Nearest Neighbors (KNN) Model**

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

A graph with red lines

Description automatically generated

### **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.

A screenshot of a computer

Description automatically generated

### **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.

A screenshot of a computer

Description automatically generated

### **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 |

A graph with lines and dots

Description automatically generated

### **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.

**A screenshot of a computer

Description automatically generated**

**Removing high correlated data**

**Threshold: 0.75**

**A computer screen shot of a computer code

Description automatically generated**

**Threshold: 0.8**

**A screenshot of a computer code

Description automatically generated**

**Threshold: 0.85**

**A computer code with blue text

Description automatically generated**

Correlation coefficients for the first 10 predictors:

A screenshot of a computer

Description automatically generated

**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.

**A group of blue graphs

Description automatically generated**

**11 predictors retained by PCA – feature reduction:**

**A screenshot of a computer

Description automatically generated**

**After data was partitioned:**

A screenshot of a computer code

Description automatically generated

**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%)

**A screenshot of a computer code

Description automatically generated**

**A number with a blue stripe

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

***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%)

A screenshot of a computer

Description automatically generated

***Predicted data:***

A group of black letters

Description automatically generated

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

A close-up of a text

Description automatically generated

***Model Tuning Plot***

A graph of different colored lines

Description automatically generated

***Tuning Parameters Variation—Heatmap***

A diagram of a heat-map

Description automatically generated

**The ROC Curve**

**A diagram of a model

Description automatically generated**

**Regularized Discriminant Analysis (RDA) Model**

***Fitting the model—preprocessing on training data***

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

A screenshot of a computer

Description automatically generated

**A close-up of numbers

Description automatically generated**

**A close-up of numbers

Description automatically generated**

***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%)

A screenshot of a computer

Description automatically generated

***Predicted data***

A group of black letters

Description automatically generated

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

A close-up of a text

Description automatically generated

***Model Tuning Plot***

A graph of a diagram

Description automatically generated with medium confidence

***Tuning Parameters Variation—Heatmap***

A graph of blue squares

Description automatically generated

***The ROC Curve***

**A graph of a curve

Description automatically generated**

### **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","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"))

### #### 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"

)

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 [↑](#footnote-ref-1)
2. See url: <http://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic> [↑](#footnote-ref-2)
3. Section: “1.0 Background” has details about the data source. [↑](#footnote-ref-3)