CARLETON UNIVERSITY

SCHOOL OF
MATHEMATICS AND STATISTICS

HONOURS PROJECT

TITLE: Comparing Machine Learning Algorithms for Diagnosing Breast Cancer: Logistic Regression, Support Vector Machine, Naïve Bayes

AUTHOR: Lara Habashy

SUPERVISOR: Dr. Shirley Mills

DATE: August 3rd, 2020

Abstract

Cancer was first recognized in Egypt in early 1500 BC and since that date, researchers have been searching for a cure. The first cases were of cancer tumours located in the breast. Since then, there have been numerous new types of cancer and cancer cases. The number of cancer patients has been increasing rapidly over the past few decades and has become the leading cause of death for women in Canada. The objective of this study is to predict breast cancer diagnoses by applying machine learning algorithms such as Logistic Regression, Support Vector Machine, and Naïve Bayes on patient tumour measurements data. The performances of the algorithms are compared to determine the best classifier and the best predictive model. The experimental results suggest that the support vector machine classifier performs the best at diagnosing breast cancer tumours, with 98.82% accuracy.

Acknowledgments

I would like to thank my project supervisor, Dr. Shirley Mills, for providing support, guidance, and encouragement whenever needed. Your dedication and commitment to your work is truly inspirational.

I would like to thank the professors who have been instrumental in educating me in the fields of Mathematics and Statistics. In particular, I would like to thank Dr. Brandon Fodden, Dr. Patrick Pharrell, Dr. Natalia Stepanova, Dr. Mohamedou Ould Haye, and Dr. Jason Nielsen. Thank you for mentoring me, it's been an honour being your student. I would like to express my deepest gratitude to Dr. Song Cai for reviewing my project and providing valuable feedback.

I would also like to extend many thanks to all the members of staff at Carleton's School of Mathematics and Statistics, especially Kylie Harvey, Kevin Crosby, and Tracie Barkley. Thank you for assisting me during my time as a student and a teaching assistant at the School of Mathematics and Statistics.

This page is intentionally left blank.

Contents

1 Introduction

Machine learning is a subset of artificial intelligence (AI) which provides algorithms with the ability to improve through experience and without human intervention. The purpose of machine learning is to implement methods to detect insightful patterns in any form of data (i.e. structured, unstructured, complex). In practice, most machine learning algorithms are used for classification, regression analysis, pattern recognition, and computer vision. Machine learning algorithms can be applied in various areas such as fraud detection, spam e-mail filtering, classification of text documents, disease modelling, computer vision and pattern recognition.

Most of the applications mentioned so far are examples of supervised machine learning. Supervised learning is a type of machine learning that builds a model using both inputs and outputs of the data to predict future events. Suppose a given dataset has input variables x and an output variable y. A supervised learning algorithm's purpose is to learn or approximate the mapping function $y = f(x)$ such that for a new x, y can be easily predicted. That is, to make predictions, a supervised learning algorithm builds a model based on training data or observed data. The model uses statistical, probabilistic, and optimization techniques to keep learning and improving its predictions. To summarize, a predictive model is built by learning a dataset with known labels and predicting unlabelled data.

In this study, we implement three different supervised learning algorithms to predict whether a given breast tumour is cancerous or non-cancerous. A cancerous tumor is called a *malignant* tumour. Malignant tumours are dangerous and can grow quite fast. They can also be difficult to treat and may spread to other parts of the body. Benign tumours, on the other hand, are non-cancerous tumours that do not spread to other parts of the body. Malignant tumours have the ability to grow back whereas benign tumours do not. Therefore, the classification of a given tumour as either *malignant* or *benign* is needed to correctly diagnose and treat patients.

The structure of this paper is as follows. Section 2 presents the methods used and a description of the Wisconsin Breast Cancer dataset. Section 3 presents the data preprocessing before modelling and explores feature selection techniques. Sections 4, 5, and 6 present the classification algorithms Logistic Regression, Support Vector Machine, and Naïve Bayes, respectively. The structure of the three sections is as follows. First, a brief description of the algorithm is provided through motivating examples and scenarios. Next, the derivation of the classifier is presented. An application of the technique in R

is seen on the Wisconsin Breast Cancer dataset to predict a given breast cancer tumour. Last but not least, each section also highlights the advantages and limitations of applying each of the three algorithms. Following this section, a single model for each algorithm is selected for comparison in efficiency, effectiveness, and computational complexity in section 7. Section 7 aims to compare three machine learning algorithms and identify the optimal classification model for the Wisconsin Breast Cancer dataset. Section 8 briefly mentions ideas for future improvements on various topics discussed throughout this study. Section 9 cites a comprehensive list of all the references made in the paper. Lastly, Section 10 serves as an appendix with R documentation.

2 Method & Material

2.1 Breast Cancer Data

The Wisconsin Breast Cancer dataset consisting of 569 observations can be found here. The dataset is made up of 32 features, one of which is categorical and 31 continuous. Those attributes contain 10 real-valued features for each tumour. The other attributes represent the mean, standard error, and worst values for each tumour, and other attributes. The *worst* variable is equal to the maximum value of the mean and standard error, for a given diagnosis observation. For example, for the variable *radius*, we have variables *radius_mean*, *radius_se* and *radius_worst*, representing radius's mean, standard error, and worst values respectively. In total, the dataset is made up of 10 x 3 = 30 descriptive features and one identifier variable *ID*. The main purpose of this analysis is to predict the binary variable *diagnosis*. Observations with diagnosis = M imply this particular patient had a diagnosis of a cancerous tumour, **m**alignant. Observations with diagnosis = B imply the tumour detected is not cancerous, that is a **b**enign tumour diagnosis. As such, the variable *diagnosis* is a binary classification of whether a diagnosis is cancerous or not.

Variable	Description	Type
ID	unique identifier for every diagnosis	numeric
diagnosis	tumour type	categorical
radius	mean of distances from center to points on the perimeter	numeric
texture	standard deviation of gray-scale values	numeric
perimeter	distance around the shape	numeric
area	space occupied by tumour	numeric
smoothness	local variation in radius lengths	numeric
compactness	perimeter^2/ area -1.0	numeric
concavity	severity of concave portions of the contour	numeric
concave points	number of concave portions of the contour	numeric
symmetry	similarity measure	numeric
fractal dimension	index for characterizing fractal patterns	numeric

2.2 Experiment

The goal of this study is to use machine learning techniques to classify tumours as either malignant or benign. More specifically, this experiment aims to find the best possible predictive model of the Wisconsin Breast Cancer dataset using one of logistic regression, support vector machine, or naïve Bayes algorithms. For a real-life application of a predictive model for cancer diagnosis, the predictive abilities of the model must be as accurate as possible. In some cases, a wrong classification may lead to serious health complications for patients.

The choices of algorithms were chosen at varying levels of complexity. For instances, the naïve Bayes classification algorithm implements a simple probabilistic classifier whereas the support vector machine algorithm implements a more sophisticated classifier. Based on past research and related work, support vector machine is seen to be the best performing classifier on breast cancer data. The last algorithm we examine is Logistic Regression. The logistic regression algorithm is chosen for analysis as it's been successfully applied for classifying medical data such as the WBC data. It also gained a lot of popularity as a supervised machine learning algorithm for classification purposes. All three machine learning algorithms in this study implement binary classifiers.

Each of the three algorithms is trained using six different training subsets, to select the best performing model. Methods of selecting the training subsets are discussed in section 3 [Feature Selection]. Methods of data sampling such as *stratified sampling* and *k-fold cross-validation* are also discussed in section 3 [Sampling Methodology]. As a rule of thumb, we apply *k-fold cross-validation* re-sampling procedure with the fixed value $k = 10$ to initially evaluate the models. For simplicity, a value of $k = 10$ has been arbitrary selected as its experimentally proven to generate the best result.

Various R packages are utilized in this experiment, including the packages `e1071`, `naivebayes`, and `caret` for the modelling stage. The modelling results are described at the end of their respective sections.

To determine the ultimate predictive model for the Wisconsin Breast Cancer dataset, we use various performance evaluation metrics, described in section 7 [Results Comparison]. Using R, the three algorithms are compared with respect to their classification accuracy, efficiency, and effectiveness. The results of the comparisons are discussed in section 7.

3 Pre-processing & Exploratory Analysis

Prior to building the classification models, it is necessary to explore the dataset and perform some exploratory analysis.

3.1 Data Imbalance

When examining the completeness of the Wisconsin Breast Cancer data, we find no missing values. We also examine the proportions of data in each *diagnosis* class to find that 62.745% of the data is classified as *benign* (non-cancerous) and 37.258% as *malignant* (cancerous). The target variable class proportions should be preserved when taking subsets of the original dataset. Furthermore, class imbalance is a problem recognized in data mining techniques since most machine learning algorithms assume the data is equally distributed among the classes. In most cases where data is imbalanced, classification of classes is inaccurate as the major classes will dominate the smaller classes, and prediction methods will be biased. When analyzing the WBC dataset, we notice there is no significant imbalance with the diagnosis class distribution of 357 *benign* and 212 *malignant* observations. However, using imbalanceRatio function in R, we see that *diagnosis* has a 59.384% imbalance ratio which we attempt to correct. To handle this imbalance in our data and improve accuracy, we consider oversampling the less dominant class. Reducing the sample size of the more dominant class is called *under-sampling*. *Under-sampling* is often used to correct the imbalance in the data. Thus when sampling from the dataset, we consider all samples of diagnosis *malignant* (the less dominant class) and randomly select an equal number of *benign* samples. This yields a fully balanced sample for training machine learning models.

3.2 Sampling Methodology

Before fitting a model, we split the dataset into two parts, a training set consisting of 70% of the data and a testing set with the remaining 30% to validate the models' predictions. This is required to avoid the problem known as **overfitting**. There are various techniques for dealing with overfitting such as decreasing the number of features, or parameters, included in the model. We may also restrict the model's complexity to overcome overfitting. For instance, if a given model is trained and tested on the same dataset, the true error rate will be underestimated, presenting the problem of overfitting. Thus, we split the dataset and set the random state parameter to 123 for consistent results. Furthermore, we must maintain the proportions of data seen in the entire dataset. If simple random sampling is used to split the data, it is probable that the training

and testing subsets have diagnosis proportions higher or lower than the proportions seen in the complete dataset. To demonstrate this in R, we compare the class proportions of the response variable *diagnosis* taken from a simple random training sample to the entire dataset's *diagnosis* class proportions. The results indicate that the benign class is slightly over-represented while the malignant class is slightly under-represented. Further, looking at *diagnosis* class proportions in the testing set, we see that both *diagnosis* classes are misrepresented, with 56.98% benign and 43% malignant. Hence we use the `createDataPartition` function from the `caret` package in R to implement a *stratified split*. A stratified split is a method that robustly splits the data such that the training and testing sets have approximately the same percentage of samples of the target variable *diagnosis*.

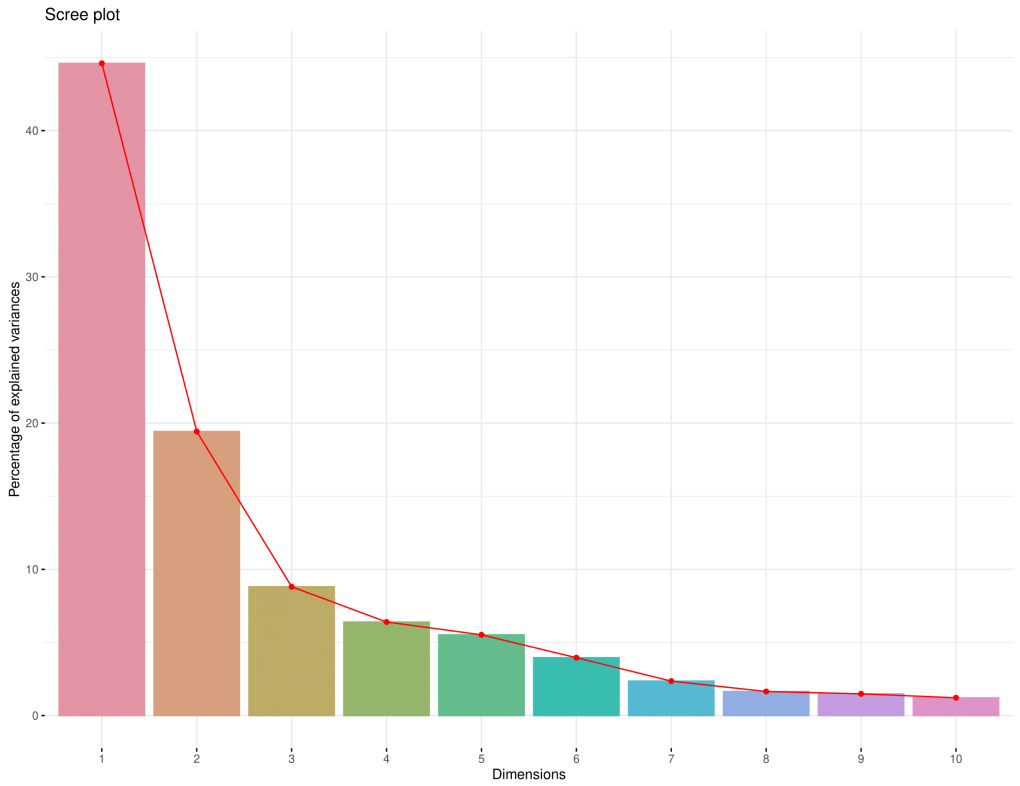
Another sampling, or rather re-sampling, approach we implement is *k-fold cross-validation*. This method of re-sampling is used to ensure every data point in the full dataset has a chance of getting sampled into either the training or testing sets. For a small a dataset such as the Wisconsin BC dataset, *k-fold cross-validation* can be used to evaluate machine learning models. This technique is especially useful in applications where we would like to analyze the performance of a predictive model that's been trained on a small dataset. The goal is to ensure the model can make predictions on new unseen data with consistent accuracy.

3.3 Principal Components Analysis

Principal Components Analysis or PCA, is a standard method for dimension reduction, which determines eigenvectors of a covariance matrix and gives directions in which the data has a maximal variance. An *eigenvector* is a vector that maintains its direction when a linear transformation is applied to it. The data is manipulated by identifying mutually orthogonal directions of decreasing variance and forming new variables, which are linear combinations of given variables. The newly created variables are all orthogonal to one another. Any two vectors are said to be *orthogonal* if they are perpendicular to each other. In summary, we wish to transform possibly correlated variables into a set of linearly uncorrelated variables, which will be the principal components.

We begin the PCA analysis in R by considering a subset of the data that excludes the response variable *diagnosis*. Next, we scale the continuous features and apply PCA. It is important to scale, or *normalize* the data when performing PCA because the algorithm projects the data onto directions which maximize the variance. When using unscaled data, we can see that the first principal component, PC1, explains a large amount of

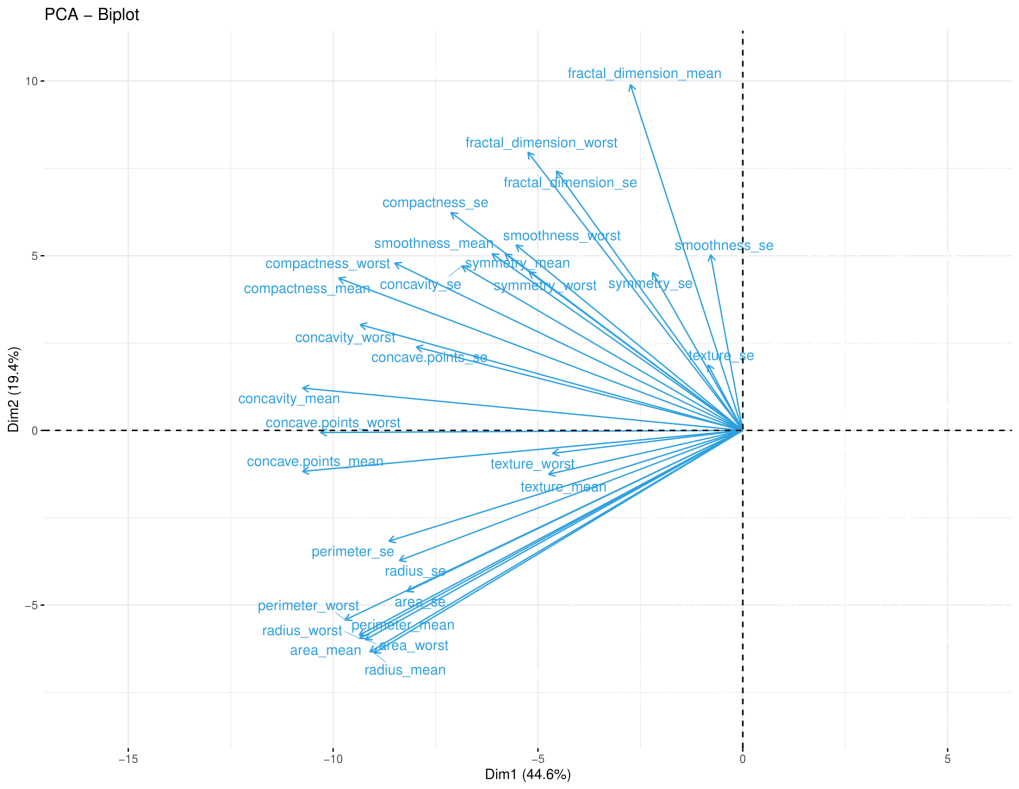
the variance. That is, not each component's contribution is maximized, and contributions are weighted inaccurately. By applying PCA on a normalized numerical dataset, we wish to find directions that would spread the data as much as possible. Using the `prcomp` function in R returns 30 principal components called $PC1$–$PC30$, with each explaining a percentage of the total variation of the data. Often, the number of principal components in rotation is equal to the number of features in the dataset, which is 30. However, when there's collinearity in the features, the number of principal components would be less than 30. Note that the first principal component accounts for most of the variability in the data. Each succeeding principal component accounts for less and less of the variability. The cumulative proportions of the PC's show that $PC1 - PC16$ account for approximately 99% of the variance in the data, and $PC1 - PC10$ accounting for approximately 95% of the variance.



Principal components are low dimensional and try to capture as much information about the original dataset, as possible. In this case, focusing our attention on the first 10 principal components is sufficient to explain the variability, since any more principal components provide diminishing returns. Also, since PCA calculates a new projection of the

data, the new axis will be based on the standard deviation of the features. Thus, features with relatively higher standard deviations will take up more weight in the calculation of the axes. This problem is resolved by normalizing our data.

Next, we look at an eigenvector representation of the principal components.



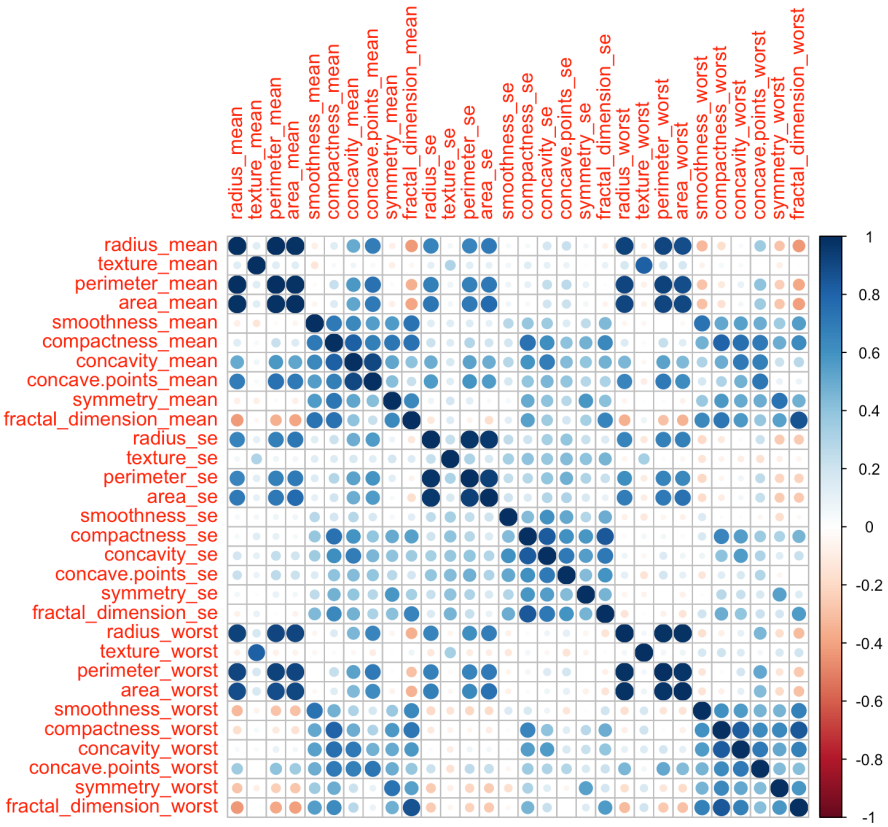
We also examine the frequency of features in the principal components. In other words, the number of times a given feature was used in the construction of the linear combination of features for the 10 principal components. The most frequently used features are seen to be *radius_mean*, followed by *texture_mean*, *symmetry_mean*, *texture_se*, and *texture_worst* [see Partial Output].

3.4 Feature Selection

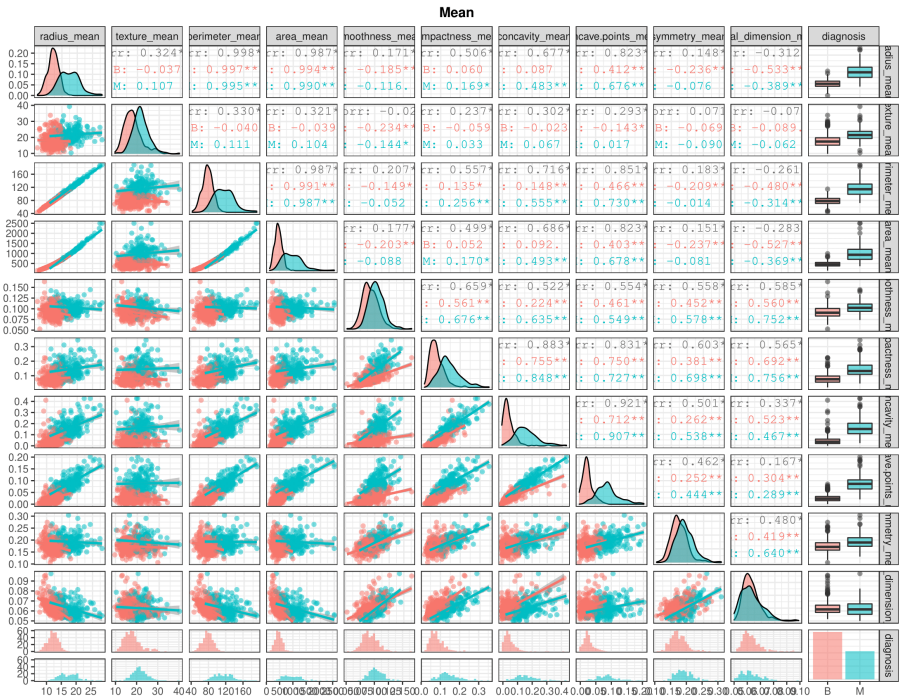
In this section, we examine a few feature selection methods to pick the best features for classification. Having the right features in the construction of classification models is essential to optimize prediction accuracy. First, we look at the correlation between variables to identify any dependencies. Next, we use a random forest classifier to find the optimal number of features to include in any given classification model that will result in the best classification possible. The classifier allows us to identify important features that best explain the variability in the dataset. Note that in the subsequent stages of the analysis, model tuning methods may be implemented in an attempt to further improve the model's prediction accuracy.

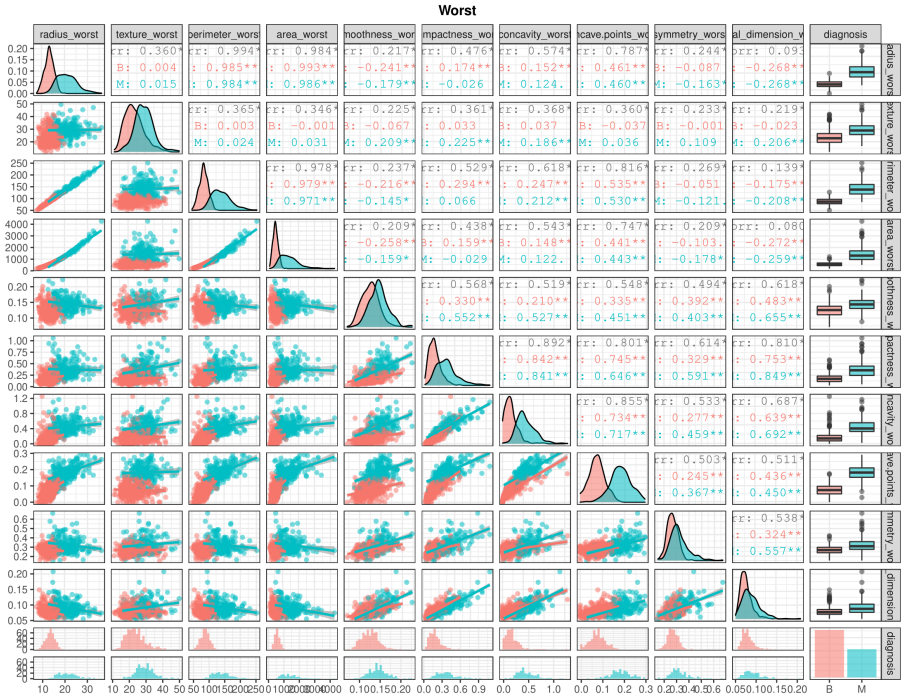
3.4.1 Correlation

As a part of the pre-processing stage, we examine the correlation amongst all features in the dataset and create a subset S_2, of uncorrelated features.



We also look at the correlation amongst features within *mean*, *standard error*, and *worst* groups. [see Appendix]





Looking at the correlation charts above, we see that *radius*, *area*, and *perimeter*, are highly correlated. The feature *radius_mean* is perfectly correlated to *perimeter_mean* and *area_mean*. This is not surprising as all three columns describe the size of a breast tumour. The mean variables for area, radius, and perimeter are highly correlated. As a result, only *area_mean* will be selected for the subset S_2. Further, the corresponding *worst* features are high correlated with one another, so we will only include *area_worst* in the analysis. We also see that the standard error features for those area, radius, and perimeter are highly correlated with one another, so we will only include *area_se* in the analysis. [see se variables correlation plot in Appendix]

The mean, standard error, and worst variables for *compactness*, *concavity* and *concave* points are all highly correlated with one another. As a result, only *concavity* variables are considered for inclusion in the subset S_2 (i.e. only include one of *concavity_mean*, *concavity_se*, *concavity_worst*). Further, *texture_mean* and *texture_worst* are highly correlated so we exclude *texture_worst* from S_2. Moreover, analyzing the correlation between the three groups of variables (mean, se, worst), we see that each features' mean and worst values are highly correlated. This is also not surprising as the *worst* columns are subsets of the *mean* columns.

After removing all the correlated features from the dataset, the subset S_2 consists of the remaining 16 uncorrelated features: *area_mean*, *concavity_mean*, *fractal_dimension_mean*, *smoothness_mean*, *symmetry_mean*, *texture_mean*, *area_se*, *concavity_se*, *smoothness_se*, *symmetry_se*, *texture_se*, *concavity_worst*, *fractal_dimension_worst*, *smoothness_worst*, *symmetry_worst*, *texture_worst*.

Note that when examining the principal components of the data, we also conclude that a total of 16 principal components represented the data as an uncorrelated linear combinations.

3.4.2 Individual Feature Evaluation

Next, we consider a feature selection method that allows for individual evaluation of each feature. We apply the function `selectKBest` on the full dataset to select a subset for modelling that utilizes the most significant features. To determine an optimal number of features, or the best k, that will yield that strongest predictive powers, we first look at the *value per features* attribute of all the features in the model. The top feature's value, *concave.points_mean* is seen to be 99.5% and the least valuable feature, *smoothness_se* at 80.7%. We arbitrarily select all features with a value of approximately at least 97%. We refer to the subset consisting of those features as S_4 of 569 observations and 17 features.

We also consider applying the `selectKBest` function to select an even smaller subset of strong predictive features from S_2, the subset with no correlated features. We refer to a subset of those features as S_3 of 569 observations and 10 features. The subset S_3 consists of the following 10 features: *area_mean*, *concavity_mean*, *area_se*, *concavity_se*, *smoothness_se*, *texture_se*, *concavity_worst*, *fractal_dimension_worst*, *symmetry_worst*, *texture_worst*. The value per feature for the features in S_3 range from 98.5% to 89%. This range is much lower than that of the features in subset S_4. This tells us that there are strong predictive features that are uncorrelated and hence, are not a part of the set S_2, such as *concave.points_mean*.

3.4.3 Random Forests

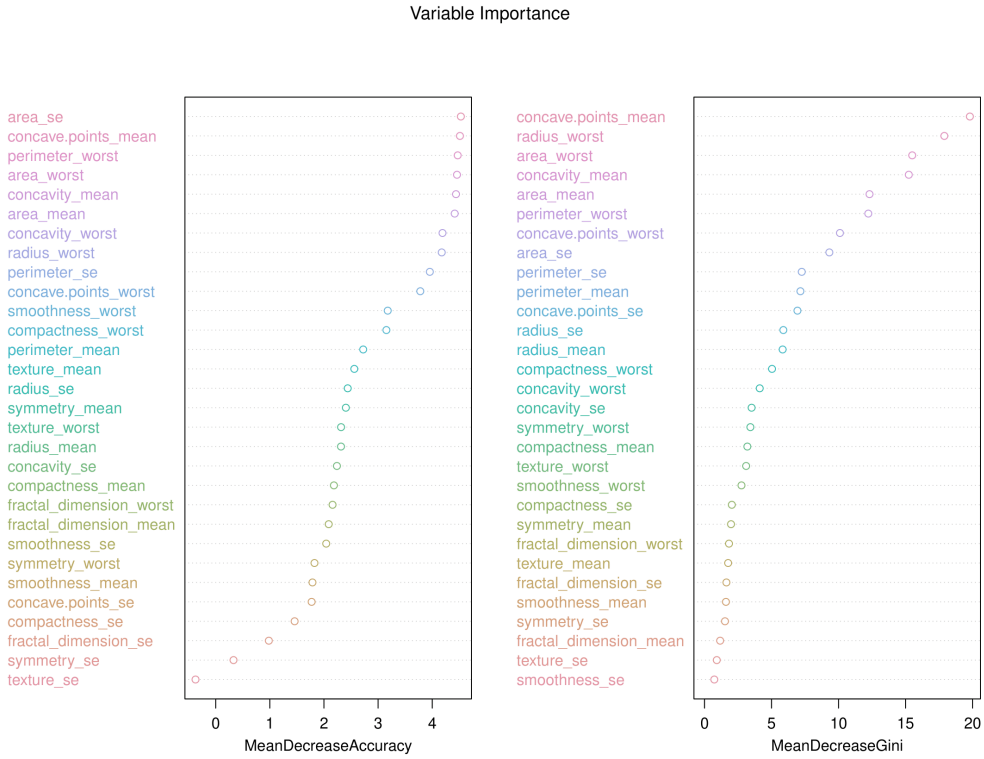
Random Forests is a classification model that is built by aggregating multiple decision trees. These trees are built on *bootstrapped* training data samples. When building these decision trees, each time a split in a tree is considered, a random sample of features is chosen as split candidates from the full set of features. This is done to prevent all of the *bagged* trees from looking very similar to one another, in the case that there is one very strong feature. The method allows the other (possibly weaker) features to have a better

chance of influencing the model. Therefore, the average tree is less variable and hence more reliable. In this subsection of feature selection, we use a random forest classifier as a method of selecting the best features for prediction.

The random forest classifier trained on the Wisconsin Breast Cancer dataset (S_1 training set) yields a the prediction accuracy of 95.29%. Next, we utilize two different techniques to determine the most significant features.

3.4.3.1 Method #1: Accuracy & Gini Importance

The first approach we take to evaluate variable importance uses accuracy and gini importance.

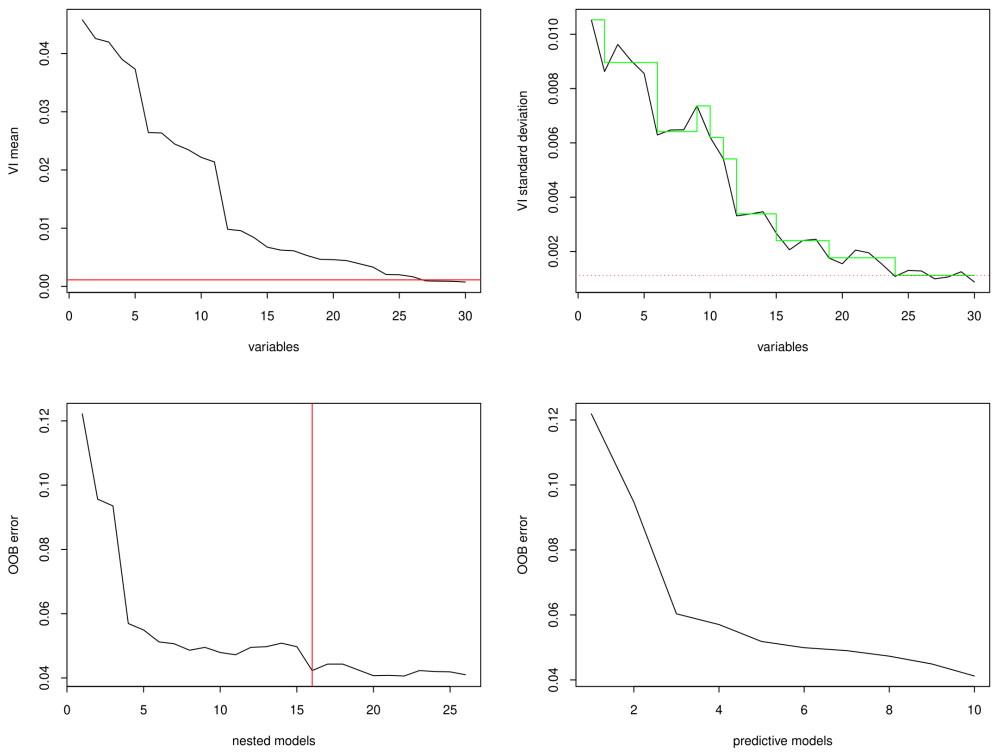


The importance of each variable in the random forest model is displayed in the figure above. The importance function outputs a list of features, along with their corresponding *Mean Decrease Gini* and *Mean Decrease Accuracy* values. *Mean Decrease Accuracy* suggests that if the variable is not important, then rearranging its values should not degrade the model's prediction accuracy. The features at the top of the figure have the most predictive power in the model. Eliminating these features would significantly decrease the

predictive power of the model. The variable importance results using random forest are fairly consistent with the Individual Feature Evaluation results when applied to the full dataset. The variable importance method suggests that the important features are those describing the size of tumours (i.e. area, radius, perimeter). This suggests that the size of tumours is a strong predictor of whether or not a given tumour is cancerous. Also, the size of the tumour is usually indicative of the treatment the patient must undergo as bigger tumours may require more severe treatment. We select the most important features to form subset S_5 for training the classifications models.

3.4.3.2 Method #2: VSURF (Variable Selection Using Random Forests)

The second approach we take to evaluate variable importance uses the R package `VSURF`. The recently publish library (2019) implements a 3-step feature selection process using random forests. At the first step, the algorithm removes any insignificant features in the model. For the WBC dataset, the total number of features is reduced from 30 to 25 features. This step is sometimes referred to as the *thresholding* step. In the figure below, the top 2 graphs are a result of the first step of the algorithm, the *thresholding* step where variable importance mean and standard deviations are plotted against all the features in the dataset. The *threshold* level is shown by a red line. The *fitted* green line corresponds to the predictions of the black line made by a CART tree. The second step is the *interpretation* step. The algorithm selects features that explain the response variable *diagnosis* the most. The third graph (bottom left) plots the mean *out of bag* error rate of embedded random forest models. After this step, the total feature count is reduced once more from 25 to 16, outlined by the red line. The third and final step is the prediction step. The algorithm further reduces the number of features in the model by eliminating the weak predictors. After this step, we're left with 10 features which are used to form subset S_6. [see Partial Output for function output]



Remarks:

1. Using a random forest classifier for feature section assigns correlated features similar importance levels. The algorithm may randomly select one feature from each group of correlated features to be used as predictors in the model. However, the overall importance of all correlated features may be underestimated. This should be accounted for when building a random forest model with significant, highly correlated features. In real-life applications of this measure, the underestimation of significant, yet highly correlated, features may create issues in interpreting the model's results. However to reduce overfitting, the random forest variable importance method for feature selection is able to handle correlated features smoothly.

2. It is also important to note that the random forest algorithm prefers features with high cardinality. In this case, the Wisconsin Breast Cancer dataset's features do not have high cardinality as tumour measurements are unique to every patient.

3. The VSURF selection process is computationally expensive and so implementing it as a part of the feature selection process on large datasets may be limiting.

4. Although the VSURF algorithm is computationally expensive and may be extremely slow in practice, it is praised for its versatility, or more specifically, its' ability to handle high dimensional data. In fact, `VSURF` was first developed for the purpose of handling high dimensional data.

4 Logistic Regression

Logistic Regression is a supervised learning method for classification problems. Logistic regression is often used in cases where the response variable is categorical or binary, the data is linearly separable, or we are interested in determining the probability of each prediction. It measures the relationship between the response variable and one or more input features.

Logistic regression has the ability to make binary predictions using a binary response variable when we're also interested in computing the associated probabilities. We will be using logistic regression to predict whether or not, a given diagnosis from the Wisconsin Breast Cancer dataset is *malignant* or *benign*. Note that logistic regression is unable to predict continuous variables. The algorithm fits a logistic function and estimates a probability between 0 and 1 for every prediction. Therefore, the models' output can only be a discrete variable. Often in classification problems, if the probability of diagnosis being *malignant* is greater than 50%, it is classified as *malignant*, otherwise *benign*. Further, we can use a logistic regression model to extract variables in the data that most explain the variability to build a predictive model.

4.1 Motivation

Linear Regression models can be represented in the form $y = \vec{x}\vec{\beta} + \epsilon$ for response variable y, independent variables $\vec{x}$, coefficients $\vec{\beta}$, and random error ϵ. In Logistic Regression, ϵ_i's are assumed to be independent and normally distributed with mean equal to 0 and variance σ^2, ϵ_i $N(0, \sigma^2)$. Naturally, this assumption translates into using continuous variables for response variable y. However, y is often a categorical variable classifying data into groups of "success" or "failure". In such cases, the normality assumption is no longer valid. Instead, we use *Logistic Regression*. To demonstrate why simple linear regression is used instead of classification, we consider the following example.

Suppose we wish to classify the *weather* which has classes *sunny, cloudy, rainy, snowy* encoded as integer values $1, 2, 3, 4$, respectively. This implies that the difference between any two of the classes (say *sunny* and *cloudy*) is equivalent to the difference between any other pair of classes (say *rainy* and *snowy*). For such data, linear regression is not sufficient.

Let y_i be the number of successes in n_i trials. Assume $y_i's$ are independent and follow a binomial distribution, $y_i \sim binomial(n_i, \theta_i)$, where θ is the probability of success

$(0 \leq \theta \leq 1)$. We model θ_i as a function of predictors x_i in the following form,

$$\theta_i = E(\frac{y_i}{n_i}) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_N X_N}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_N X_N}} \qquad i = 1, \ldots, N \qquad (4.1)$$

This form of the *logistic* function is known as the *sigmoid* function. The logistic regression algorithm uses this function (4.1) to transform a given test sample with N features into a probabilistic range between 0 and 1. Setting a threshold will then lead to a conclusive classification of said sample.

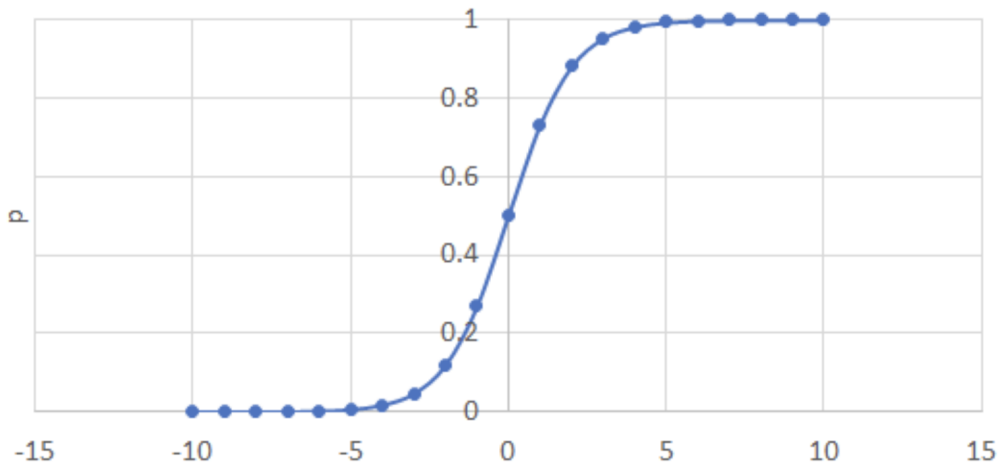
We can also derive an expression for the probability of failure as:

$$1-\theta_i = \frac{1}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_N X_N}} \implies \frac{\theta_i}{1-\theta_i} = e^{\beta_0+\beta_1 X_1+\ldots+\beta_N X_N}$$

When modelling linear separable data with logistic regression, model (4.1) will have to be transformed to a *linear* model. This is done by modelling the logarithm of the odds of success, called **logit**, where the odds of success is defined as the ratio of the probability that an event occurs to the probability that an event does not occur.

$$logit(\theta_i) = \log\left(\frac{\theta_i}{1-\theta_i}\right) = \beta_0 + \beta_1 X_1 + \ldots + \beta_N X_N \qquad (4.2)$$

This sigmoid function gives an S-shaped linear curve with θ_i's, the probabilities of success, take on the y-axis $(0 \leq y \leq 1)$, and x_i's take on the x-axis $(-\infty \leq x \leq +\infty)$. By setting a *threshold* level, the values of x are transformed to either 0 or 1. That is, if the probability of a data point is greater than the *threshold* value, the output is transformed to 1, otherwise 0.



Decision Boundaries

1. If θ_i increases, $logit(\theta_i)$ also increases
2. If $\theta_i < 0.5$, then $logit(\theta_i)$ is negative
3. If $\theta_i > 0.5$, then $logit(\theta_i)$ is positive
4. $logit(\theta_i)$ varies over the real line $\quad (-\infty < logit(\theta_i) < \infty)$

4.2 Parameter Estimation using Maximum Likelihood

To maximize the likelihood of correctly classifying a given data example, we need to estimate the coefficients vector $\hat{\beta}$. Suppose we have a set of N independent observations $(y_i, x_i, n_i)(i = 1, \ldots, N)$ labeled 1 or 0. For samples labelled 1, we wish to estimate $\hat{\beta}$ such that $\hat{\theta}$, or the product of all conditional probabilities of class 1 samples, is as close to 1 as possible. For samples labelled 0, we wish to estimate $\hat{\beta}$ such that $1 - \hat{\theta}$, or the product the compliment of all conditional probabilities of class 1 samples, are as close to 1 as possible. In summary, we want to find $\hat{\beta}$ such that the product of the above mentioned requirements is maximum over all elements of the dataset. This is done by optimizing the likelihood function. The *likelihood function* is defined as

$$L = \prod_{i=1}^{N} f(y_i) = \prod_{i=1}^{N} \binom{n_i}{y_i} \theta_i^{y_i} (1-\theta_i)^{n_i - y_i} = \prod_{i=1}^{N} \binom{n_i}{y_i} \left(\frac{\theta_i}{1-\theta_i}\right)^{y_i} (1-\theta_i)^{n_i}$$

The maximum likelihood estimators of $\beta_0, \beta_1, \ldots, \beta_N$ are obtained by maximizing L or $\log L$.

$$\log L = \sum_{i=1}^{N} \left\{ \log \binom{n_i}{y_i} + y_i \log\left(\frac{\theta_i}{1-\theta_i}\right) + n_i \log(1-\theta_i) \right\}$$

Deriving $\log L$ with respect to $\vec{\beta}$:

$$\frac{\partial \log L}{\partial \vec{\beta}} = \sum_{i=1}^{N} \left\{ 0 + y_i \frac{\partial}{\partial \vec{\beta}} (x_i^t \vec{\beta}) + n_i \frac{\partial}{\partial \vec{\beta}} \log(1-\theta_i) \right\}$$

$$= \sum_{i=1}^{N} \left\{ y_i x_i + n_i \frac{\partial}{\partial \vec{\theta_i}} \log(1-\theta_i) \frac{\partial \theta_i}{\partial \vec{\beta}} \right\} = \sum_{i=1}^{N} \left\{ y_i x_i + n_i \frac{(-1)}{1-_i} \theta_i (1-\theta_i) x_i \right\}$$

$$\implies \frac{\partial \log L}{\partial \vec{\beta}} = \sum_{i=1}^{N} (y_i - n_i \hat{\theta}_i) x_i =: 0 \qquad (4.3)$$

Recall that $\hat{\theta}_i = \frac{e^{x_i^t \hat{\vec{\beta}}}}{1+e^{x_i^t \hat{\vec{\beta}}}}$ and note that there is no *explicit* solution for $\hat{\vec{\beta}}$. The goal is to optimize equation (4.3) by finding the value of β that maximizes it. Numerical estimation methods are used to find an approximation for β.

4.3 Optimization using IRWLS algorithm

To solve equation (4.3), we use an iterative algorithm, called *iteratively reweighted least squares algorithm*, denoted IRWLS.

First, let $w_i = n_i\hat{\theta}_i(1-\hat{\theta}_i)$. Next, rewrite equation (4.3) in the following form

$$\sum_{i=1}^{N} x_i n_i \hat{\theta}_i(1-\hat{\theta}_i)\left\{\frac{y_i - n_i\hat{\theta}_i}{n_i\hat{\theta}_i(1-\hat{\theta_i})}\right\} =: 0$$

$$\implies \sum_{i=1}^{N} x_i w_i \left\{\frac{y_i - n_i\hat{\theta}_i}{n_i\hat{\theta}_i(1-\hat{\theta_i})} + \log\left(\frac{\hat{\theta}_i}{1-\hat{\theta}_i}\right) - \log\left(\frac{\hat{\theta}_i}{1-\hat{\theta}_i}\right)\right\}$$

Or;

$$\sum_{i=1}^{N} x_i w_i \left\{\frac{y_i - n_i\hat{\theta}_i}{n_i\hat{\theta}_i(1-\hat{\theta_i})} + logit(\hat{\theta}_i)\right\} = \sum_{i=1}^{N} x_i w_i logit(\hat{\theta}_i)$$

$$\implies \sum_{i=1}^{N} x_i w_i z_i = \sum_{i=1}^{N} x_i w_i x_i^t \hat{\vec{\beta}} \tag{4.4}$$

where $w_i = n_i\hat{\theta}_i(1-\hat{\theta}_i)$ and $z_i = logit(\hat{\theta}_i) + \frac{y_i - n_i\hat{\theta}_i}{n_i\hat{\theta}_i(1-\hat{\theta_i})}$.
Equation (4.4) can be rewritten in matrix form as

$$x^t w\vec{z} = x^t w x\hat{\vec{\beta}}$$

where $\vec{z} = (z_1, \ldots, z_N)^t, w = diag(w_1, \ldots, w_N), x = (x_1, x_2, \ldots, x_N)^t$. Rearranging we get,

$$\hat{\vec{\beta}} = (x^t w x)^{-1} x^t w\vec{z} \tag{4.5}$$

Since w and z are functions of β itself, we need to use an iterative method to derive an expression from equation (4.5). We treat z_i as the response variable and regress z_i on x_i with weight w_i.

IRWLS algorithm

1. Obtain initial estimate of $\vec{\beta}$ and calculate $\hat{\theta}_i = \frac{e^{x_i^t \hat{\vec{\beta}}}}{1+e^{x_i^t \hat{\vec{\beta}}}}$.
   Call the estimates $\widetilde{\beta}$ and $\widetilde{\theta_i}$.

2. Given $(\widetilde{\beta}, \widetilde{\theta_i})$, calculate the so called "pseudo-observation" $z_i = logit(\hat{\theta_i}) + \frac{y_i - n_i\hat{\theta_i}}{n_i\hat{\theta_i}(1-\hat{\theta_i})}$

3. Set $w_i = n_i\widetilde{\theta_i}(1-\widetilde{\theta_i})$. Find the *least squares* estimator of β using z_i as the response variable, w_i as the weight variable and x_i as the covariate. $\hat{\beta} = (x^t wx)^{-1} x^t wx$

4. Using the new estimate for β, repeat steps $1 \rightarrow 3$ until convergence is achieved. At convergence, the variance of the estimator $\hat{\beta}$ is given by $V(\hat{\beta}) = (x^t wx)^{-1}$

4.4 Deviance

In logistic regression, however, the ANOVA (analysis of variance) method used by linear regression is replaced by ANODEV or the analysis of deviance method. The change in deviance between any given two models is used to assess the significance of the models. Deviance, D, is defined as

$$D = -2\left\{\log L(\hat{\theta}) - \log L(\theta^*)\right\} \tag{4.6}$$

where L is the likelihood function and $L(\hat{\theta})$ is its value at the maximum likelihood estimator $\hat{\theta}$, and $L(\theta^*)$ is its value at the probability of success in the i^{th} trial $\theta_i = \frac{y_i}{n_i}$. The likelihood function is given by

$$L = \prod_{i=1}^{N} f(y_i|n_i, \theta_i) = \prod_{i=1}^{N} \binom{n_i}{y_i} {\theta_i}^{y_i} (1-\theta_i)^{n_i - y_i}$$

$$\log L(\theta) = \sum_{i=1}^{N} \left[y_i \log \theta_i + (n_i - y_i)\log(1-\theta_i) + \log \binom{n_i}{y_i} \right]$$

So,

$$\log L(\hat{\theta}) = \sum_{i=1}^{N} \left[y_i \log \hat{\theta}_i + (n_i - y_i)\log\left(1-\hat{\theta}_i\right) + \log \binom{n_i}{y_i} \right]$$

and,

$$\log L(\theta^*) = \sum_{i=1}^{N} \left[y_i \log\left(\frac{y_i}{n_i}\right) + (n_i - y_i)\log\left(1-\frac{y_i}{n_i}\right) + \log \binom{n_i}{y_i} \right]$$

Substituting the above equations into equation (6) yields

$$D = -2\left\{ \sum_{i=1}^{N} \left[y_i \left(\log \hat{\theta}_i - \log\left(\frac{y_i}{n_i}\right)\right) + (n_i - y_i)\log\left(1-\frac{y_i}{n_i}\right) - \log\left(1-\frac{y_i}{n_i}\right)\right]\right\}$$

$$\implies D = 2\left\{\sum_{i=1}^{N}\left[y_i\left(\log\left(\frac{y_i}{n_i}\right) - \log\hat{\theta}_i\right) + (n_i - y_i)\log\left(\frac{n_i - y_i}{n_i}\right) - \log\left(1 - 1\hat{-}\theta_i\right)\right]\right\}$$

$$\implies D = 2\left\{\sum_{i=1}^{N}\left[y_i\left(\log\left(\frac{y_i}{n_i\hat{\theta}_i}\right)\right) + (n_i - y_i)\log\left(\frac{n_i - y_i}{n_i(1-\hat{\theta}_i)}\right)\right]\right\} \tag{4.7}$$

Deviance, D, follows an asymptotic chi-squared distribution with $N - P$ degrees of freedom, for P number of parameters in $\vec{\beta}$. That is, $D \sim X^2_{N-P}$.

4.5 Advantages and Limitations

The logistic regression algorithm is compatible with both continuous and discrete data. It is often used for classification modelling problems however, it can be used as a feature selection technique to identify strong predictive features. Logistic regression is popular and widely applied to classification problems. It is simple, efficient, easy to implement, and not as computationally expensive as other machine learning algorithms. Further, input features used to build a logistic regression model do not need to be normalized before modelling and hyperparameters do not require any tuning. A limitation of logistic regression appears when attempting to solve non-linear problems, as the logistic regression is a linear model in nature. The algorithm also has a strong dependence on features and so the selection of the most significant features, before modelling, is an essential step for logistic regression.

4.6 Modelling in R

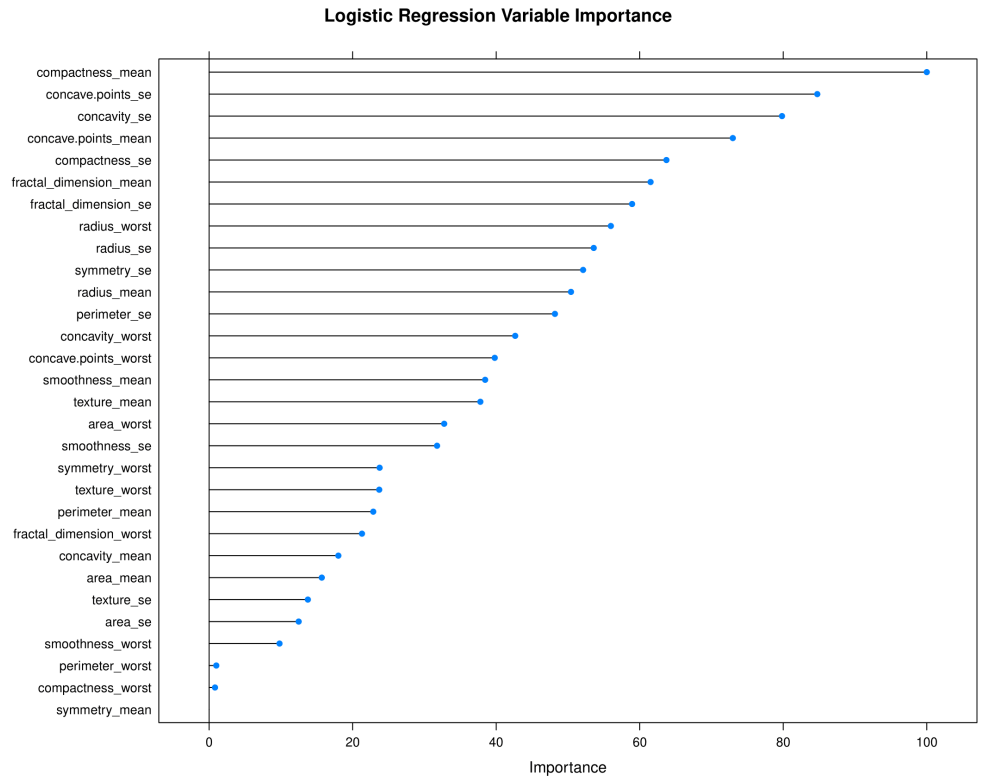
Consider an application of the logistic regression algorithm on the Wisconsin Breast Cancer dataset. Using the `caret` library in R, we build logistic regression models for each of the training subsets S_1 to S_6, as defined in the Feature Selection section. Next, we create predictions using the testing sets for each model.

Classification accuracy is the ratio of the number of correct predictions to the total number of predictions made. We use classification accuracy as a goodness of fit measure for identifying the optimal training set for prediction. The accuracy of the models gives an initial indication of model performance for selection of a single logistic regression model. Using the *k-fold cross-validation* technique with $k = 10$, we get the following classification accuracy results.

Set	No. of Features	Classification Accuracy
S_1	30	94.12%
S_2	16	95.29%
S_3	10	97.06%
S_4	17	93.53%
S_5	12	94.71%
S_6	10	96.47%

The best model is the one that maximizes the classification accuracy, which is 97.06% for the S_3 based model. The best performing model is trained on the most significant features selected from the subset of uncorrelated features, using the Individual Feature Evaluation method. For further analysis, we select the model trained using the reduced dataset S_3 of 10 significant uncorrelated features.

To investigate why the best model was the one based on S_3 training set, we can look at variable importance from the logistic regression perspective. We see that the features in subset S_3 are the strongest predictive features seen in the figure below. Logistic regression performs the best when using features that are closely correlated to one another.



5 Support Vector Machine

This section presents the supervised machine learning algorithm Support Vector Machine (SVM). SVMs are used for classification, regression analysis, and outliers detection. In this study, we will focus on the classification application of Support Vector Machines.

Support Vector Machine algorithm can classify linear and nonlinear data by constructing a hyperplane or set of hyperplanes in a high dimensional space, or infinite-dimensional space. For n features in the data, the algorithm maps each data point into an *n-dimensional* feature space. Next, the algorithm finds the *optimal hyperplane* that separates the data into one of two classes. The *optimal hyperplane* maximizes the *marginal distance*, the distance between the nearest data point and the hyperplane, and minimizes the incorrect classification of test data.

Support Vector Machine was first introduced by Vladimir N. Vapnik and Alexey Y. Chervonenkis in 1963. Unfortunately, the technology at the time was not advanced enough to implement the algorithm. It was not until the early 1990's that SVMs began to gain popularity. In 1992, Bernhard E. Boser attempted to implement a support vector machine with a slightly non-linear *kernel*. As a result, nonlinear classifiers for support vector machines were created. At the time, the main use of the algorithm was seen on handwriting recognition. Today, Support Vector Machines are known for being universal approximators of any multivariate function to any desired degree of accuracy. SVMs are useful for modelling complex, nonlinear, unknown processes. Further, the supervised learning method is implemented in various pattern recognition areas such as computer vision.

In this section, we provide a review on vector notation, define the term *margin* and derive an expression to optimize. Methods of dealing with nonlinearly separable data will be discussed, however, the application is focused on that of linearly separable data. Further, we implement the SVM algorithm for the classification of cancer tissues as either cancerous (*malignant*), or non-cancerous (*benign*).

5.1 Review of Vector Definitions

Def. A *vector* is an ordered finite list of numbers.

Def. Given the two n-dimensional vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$, the *dot product* is defined as

$$\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

Remark: The dot product is also known as the *scalar product*, or the *inner product*.

Def. A *unit vector* is a vector for which all the elements are equal to zero, except for one which is equal to one.

Def. The *Euclidean norm* of the n-dimensional vector $\vec{a} = (a_1, a_2, \ldots, a_n)$ is denoted by $||a||$ and defined as

$$||a|| = \sqrt{\vec{a} \cdot \vec{a}} = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$$

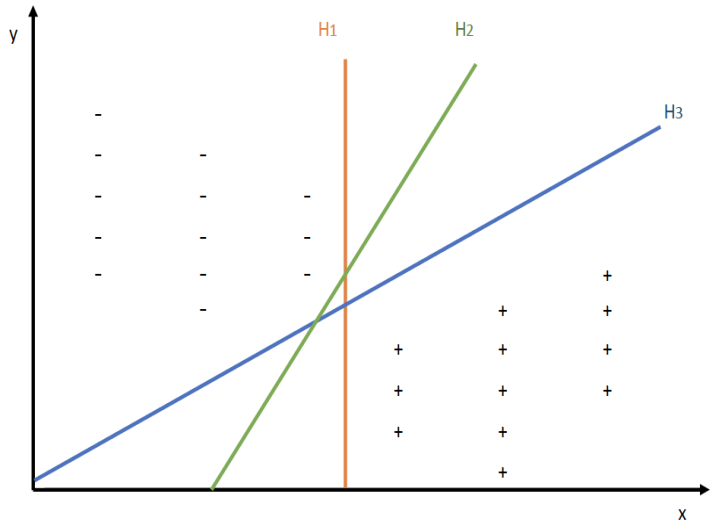
Remark: The *Euclidean norm* is also referred to as the *length*, or *magnitude* of $\vec{a}$.

Def. The *Euclidean distance* between the two n-dimensional vectors $\vec{a} = (a_1, a_2, \ldots, a_n)$ and $\vec{b} = (b_1, b_2, \ldots, b_n)$ is the norm of the difference, defined as

$$dist(\vec{a}, \vec{b}) = ||a - b||$$

5.2 Motivation

Classifying data into sub-groups is a machine learning task. Suppose we are given some training data points that belong to one of two classes. We can think of these data points as *p-dimensional* vectors. The goal is to be able to correctly classify new data points into their respective sub-groups. Support Vector Machine suggests that a linear classifier can separate the *p-dimensional* data points using a *(p-1)-dimensional* hyperplane. In the case of linearly separable data points, many hyperplanes can *split* the data linearly and classify it into distinct sub-groups. Note that vectors that lie closest to the hyperplane are called **support vectors**. To further illustrate this idea, consider the figure of hyperplanes separating the positive and negative data points below.



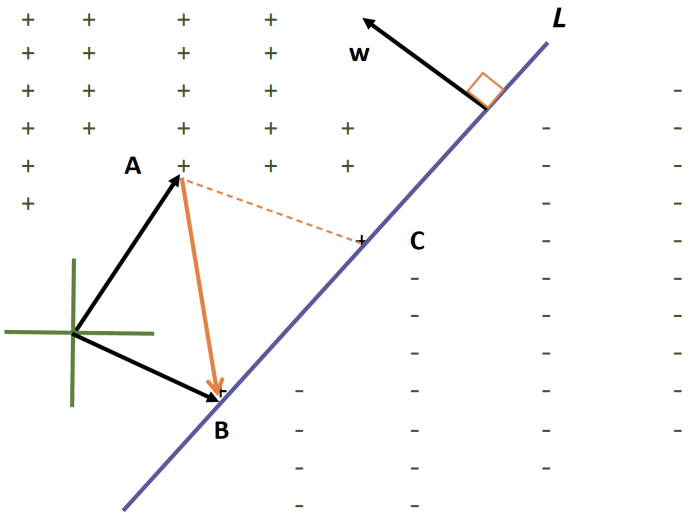
In the above figure, we have at least three candidates for a separating hyperplane. We see that H_1 separates the positive and negative data points with a small margin. H_2 also separates the data points however, the least possible distance is not as large as it could be. The third hyperplane, H_3 provides the "best" separation as the distance to the nearest +/- is as large as possible. To achieve the highest prediction accuracy possible, we consider picking the "best" hyperplane. The "best" hyperplane is the hyperplane that gives the largest separation of classes or provides the largest margins. In other words, an optimal hyperplane is one that maximizes the distance between the nearest data points and the hyperplane itself. Thus, the main objective is to find the optimal hyperplane that linearly separates the data points by maximizing the margin.

5.3 Intuition

Suppose we are given a training dataset consisting of positive and negative data points $S = (x_1, y_1), ..., (x_n, y_n)$. For every $i \in (1, \ldots, n)$, define a real-valued vector $\vec{x}$ as the feature vector such that $x_i = (x_i^{(1)}, \ldots, x_n^{(d)})$.

Remark: The square of the features sums to one, so $\vec{x}$ has Euclidean length 1.

Define y_i as a binary vector indicating whether the corresponding x_i is a positive or negative data point. That is, $y_i \in \{-1, 1\} \ \forall \ i \in (1, \ldots, n)$.
Define a line L by $\vec{w} \cdot \vec{x} + b = 0$ for weight vector $\vec{w}$ perpendicular to L, real-valued feature vector $\vec{x}$, and a bias element $b \in R$. Consider the illustration below.



Here, $\vec{w} = (w^{(1)}, w^{(2)})$ so the line L is given by

$$w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + b = 0 \qquad (5.1)$$

Next, pick a data point A such that $A = (x_A^{(1)}, x_A^{(2)})$. The location of point A is simply the vector from the origin to the point, call it $\vec{x}$.

To measure the margin, choose an arbitrary data point $B = (x_B^{(1)}, x_B^{(2)})$ that lies on the line L. Recall that any data points on L are *support vectors*.

The *marginal distance*, or the *margin*, is said to be the distance between the point A and

line L. Suppose the closest point on the line to point A is point C. Then, the distance between these two points is equal to the distance of the vector connecting them. That is, $d(A, L) = |AC|$.

To compute the length of this vector, we may consider projecting the vector connecting A and B onto the weight vector. Thus we have,

$$d(A, L) = |AC| = |(A - B) \cdot \vec{w}| = |(x_A^{(1)} - x_B^{(1)}) \cdot w^{(1)} + (x_A^{(2)} - x_B^{(2)}) \cdot w^{(2)}|$$

Applying equation (5.1) to point B as it lies on the line, we get equation (5.2)

$$d(A, L) = |(x_A^{(1)} - x_B^{(1)}) \cdot w^{(1)} + (x_A^{(2)} - x_B^{(2)}) \cdot w^{(2)}| = x_A^{(1)} w^{(1)} + x_A^{(2)} w^{(2)} + b = \vec{w} \cdot \vec{x} + b$$

The distance between point A and the line L is simply the dot product of the perpendicular vector $\vec{w}$ and the vector $\vec{x}$ from the origin to point A.

There are many possible candidates for a perpendicular vector $\vec{w}$ since it can be of any length. To find the optimal $\vec{w}$ (pick a particular vector and particular constant b), we consider the constraints imposed by the decision rule.

5.4 Decision Rule

To find the optimal $\vec{w}$, the line that will best separate the data, we impose the following decision rule: For a given vector $\vec{x}$, if $\vec{w} \cdot \vec{x} + b \geq 0$, then $\vec{x}$ represents a positive data point, otherwise it represents a negative data point.

Let $\vec{x}_+$ denote a sample of only positive data points and $\vec{x}_-$ denote a sample of only negative data points. For mathematical convenience, we introduce an indicator variable y_i such that $y_i = +1$ for positive data points and $y_i = -1$ for negative data points. That is, class labels are selected from $\{-1, 1\}$. Then, by the decision rule we have,

$$\begin{cases} \vec{w} \cdot \vec{x}_+ + b \geq 1, & y_i = +1 \quad (5.3) \\ \vec{w} \cdot \vec{x}_- + b \leq -1, & y_i = -1 \quad (5.4) \end{cases}$$

Next, we multiply the left hand side of equations (5.3) and (5.4) by y_i which yield the following.

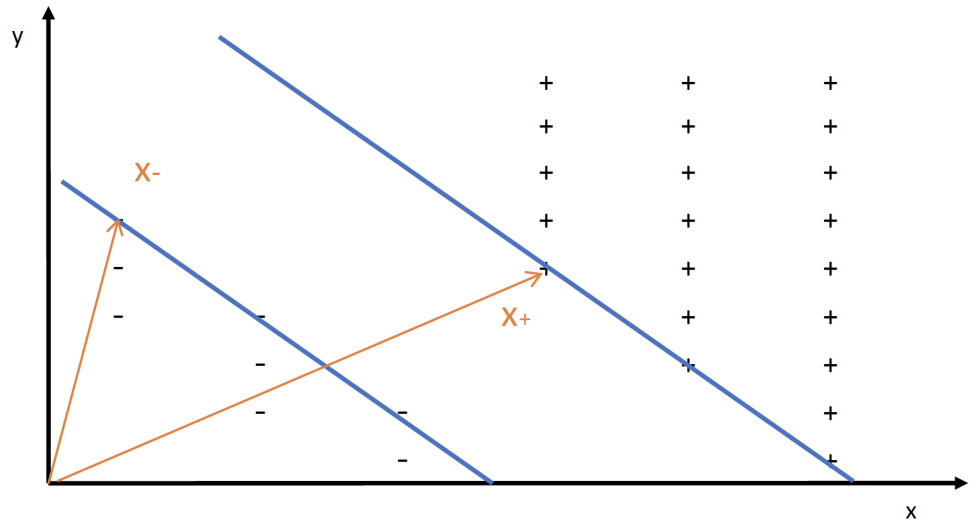
$$\begin{cases} y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 & \implies y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0 \quad (5.5) \\ y_i(\vec{w} \cdot \vec{x}_i + b) \leq -1 & \implies y_i(\vec{w} \cdot \vec{x}_i + b) - 1 \geq 0 \quad (5.6) \end{cases}$$

An additional constraint is subjected to the support vectors. We have,

$$y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0 \tag{5.7}$$

for all data points which lie on the plane, i.e. support vectors. In the context of the illustration above, we have $y_i(\vec{w} \cdot \vec{x}_i + b) - 1 = 0$ for data points B and C.

Suppose we have the following plane with given vectors $\vec{x}_+$ and $\vec{x}_-$ representing the distance from the origin to the respective positive and negative data points.



To determine the margin, as we've done before, we consider taking the dot product $\vec{x}_+ + \vec{x}_-$ and a perpendicular unit vector. Since $\vec{w}$ is a *normal* vector, it needs to be divided by the magnitude of $\vec{w}$ to make it a *unit* vector. Therefore, the **margin** is given by

$$(\vec{x}_+ + \vec{x}_-) \cdot \frac{\vec{w}}{||\vec{w}||} \tag{5.8}$$

Remark: The margin, (5.8) yields a scalar quantity.

For the positive sample $\vec{x}_+$, using constraint (5.7) with $y_i = +1$, we get $\vec{x}_+ \cdot \vec{w} = 1 - b$. Similarly, for the negative sample $\vec{x}_-$, using constraint (5.7) with $y_i = -1$, we get $\vec{x}_- \cdot \vec{w} = -1 - b$. Then, (5.8) becomes

$$(\vec{x}_+ + \vec{x}_-) \cdot \frac{\vec{w}}{||\vec{w}||} = [(1 - b) - (-1 - b)]\frac{1}{||\vec{w}||} = \frac{2}{||\vec{w}||}$$

The problem now becomes a convex quadratic programming problem, or in other words, an optimization problem. How can we maximize the margin subject to the above mentioned constraint? How do we find the optimal vector $\vec{w}$?

Maximizing the Margin
Claim: We can optimize the margin by subjecting result (5.2) to the constraints imposed by the Decision Rule.

First, note that the problem of maximizing the margin $\frac{2}{||\vec{w}||}$ is equivalent to maximizing $\frac{1}{||\vec{w}||}$ since 2 is a scalar. This expression is equivalent to minimizing the reciprocal, $||\vec{w}||$. For mathematical convenience, we can rewrite the $||\vec{w}||$ as $\frac{1}{2}||\vec{w}||^2$.

That is, $\max \frac{2}{||\vec{w}||} \rightarrow \max \frac{1}{||\vec{w}||} \rightarrow \min ||\vec{w}|| \rightarrow \min \frac{1}{2}||\vec{w}||^2$.
The problem is now transformed into a solvable form, an optimization problem subject to linear constraints. The solution to this problem yields the **optimal margin classifier**.

5.5 The Lagrangian Method

In optimization problems, the method of **Lagrange multipliers** is often used to find the local maxima or minima of a multivariate function subject to constraints. This method is widely used as it transforms a constrained problem into a form free of any constraints and easy to optimize. We define the Lagrangian to be

$$\mathcal{L} = \frac{1}{2}||\vec{w}||^2 - \sum_{i}^{\infty} \alpha_i [y_i(\vec{w} \cdot \vec{x} + b) - 1] \tag{5.9}$$

Here, $\alpha_i' s$ are called **lagrange multipliers**. To find a solution to this expression, we take the partial derivatives with respect to $\vec{w}$ and b, and set them to zero.

$$\frac{\partial \mathcal{L}}{\partial \vec{w}} = \vec{w} - \sum_i \alpha_i y_i \vec{x} := 0 \implies \vec{w} = \sum_i \alpha_i y_i \vec{x}_i \tag{5.9.1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_i \alpha_i y_i := 0 \implies \sum_i \alpha_i y_i = 0 \tag{5.9.2}$$

Substituting equations (a) and (b) into (9) we get,

$$\mathcal{L} = \frac{1}{2}\left(\sum_{i=1}^{n} \alpha_i y_i \vec{x}_i\right) \cdot \left(\sum_{j=1}^{n} \alpha_j y_j \vec{x}_j\right) - \sum_{i=1}^{n} \alpha_i y_i \vec{x}_i \left(\sum_{j=1}^{n} \alpha_j y_j \vec{x}_j\right) - \sum_i \alpha_i y_i b + \sum_i \alpha_i$$

$$\implies \mathcal{L} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \tag{5.10}$$

Therefore, the initial Lagrange problem (5.9) is now transformed to problem (5.10). We wish to maximize (5.10) subject to the following constraints: $\alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$. Note that the expression to optimize (5.10), depends only on the dot product of pairs of training samples. Combining the expression for $\vec{w}$ (5.9.1) with the decision rule, we get the following expression for classifying positive data points for unknown vector $\vec{u}$.

$$\sum_{i=1}^{n} \alpha_i y_i \vec{x_i} \cdot \vec{u} + b \geq 0 \tag{5.11}$$

Many optimization algorithms can be used to solve this *dual* optimization problem.

5.6 Kernel Functions

The purpose of a kernel function is to transform input data into the desired form in a different space. Kernel functions are often used with the implementation of support vector machine in cases where the data is not linearly separable. Kernel function transformations provide a more convenient representation of the data. We define this transformation as $\phi(\vec{x})$. Note that we do not necessarily need to have information regarding the transformation function $\phi(\vec{x})$. It is sufficient to know the new representation of the data as the dot product of the given vectors in another space. Therefore, we can use kernels to approximate nonlinear decision boundaries.

The following are some of the most common kernel functions.

1. Polynomial kernel:
$$K(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^n$$

2. Gaussian radial basis kernel:
$$K(\vec{x_i}, \vec{x_j}) = e^{\left(-\frac{||\vec{x_i} - \vec{x_j}||^2}{2\sigma}\right)}$$

3. Laplace RBF kernel
$$K(\vec{x_i}, \vec{x_j}) = e^{\left(-\frac{||\vec{x_i} - \vec{x_j}||}{\sigma}\right)}$$

4. Sigmoid kernel
$$K(\vec{x_i}, \vec{x_j}) = \tan n(k\vec{x_i} \cdot \vec{x_j} - \delta)$$

Remark: Valid kernel functions must have positive semi-definite similarity matrices so that the data is expressed as dot products. We say a given matrix is *positive semi-definite* if and only if it is symmetric and non-negative. A *symmetric* matrix is a square matrix (i.e. equal dimensions) whose entries are symmetric with respect to the main diagonal.

5.7 Advantages and Limitations

The support vector machine algorithm is one of the best machine learning tools for linearly separable data. It is especially useful as a binary classifier. The algorithm can handle high dimensional data well. As a result, SVM is often used in application to medical data such as the WBC dataset. Although the WBC dataset does not contain any outliers as the numeric data measures tumours located in the breast, the support vector classifier suppresses the impact of outliers in any given model.

Furthermore, as a result of having a constraint for every single training example in a given dataset, the support vector machine algorithm is computationally expensive. This may present a limitation to applications on big datasets as the algorithm can be extremely slow. A second limitation of support vector machines is its dependence on hyperparameters. For kernel-based SVM, the selection of the appropriate kernel function is not a simple task.

5.8 Modelling in R

Consider an application of support vector machine on the dataset of interest, the Wisconsin Breast Cancer dataset. Using the SVM library `e1071` from R, we create support vector machine models for each of the training subsets S_1 to S_6, as defined in the Feature Selection section. Next, we create predictions using the testing subsets for each model.

Recall that classification accuracy is the ratio of the number of correct predictions to the total number of predictions made. We use classification accuracy as a goodness of fit measure for identifying the optimal training set for prediction. The accuracy of the models gives an initial indication of model performance for selection of a single SVM model.

Set	No. of Features	Classification Accuracy
S_1	30	97.06%
S_2	16	96.47%
S_3	10	93.53%
S_4	17	93.53%
S_5	12	96.47%
S_6	10	98.24%

The best model is the one that maximizes the classification accuracy, which is 98.24% for the S_6 based model. The best performing model is trained using features that were selected based on a random forest classifier. The model trained using all the features in the dataset, S_1 performed the second-best at 97.06% classification accuracy, with S_3 and S_4 models performing the worst at 93.53% classification accuracy.

Therefore, for further analysis, we select the model trained using the reduced dataset S_6 of 10 significant features. To further elevate the models' classification accuracy, we consider applying a kernel function to transform the data. However, the dataset is linearly separable and applying a kernel function does not improve the model's accuracy. The `svm` function allows us to control the hyperparameters and explicitly specify the kernel type to use and associated parameters such as *degree* and *gamma*. We attempt to improve the SVM's model accuracy by applying *k-fold cross-validation* with $k = 10$ with the `caret` package. After tuning the selected model, we see a small increase in the classification accuracy, an improvement nonetheless, from 98.24% is 98.82%.

6 Naïve Bayes

Naïve Bayes is a machine learning algorithm that uses Bayesian classification where prior knowledge and observed data can be combined. Naïve Bayes algorithm is often used for text classification such as spam filtering, classifying documents, and sentiment analysis. It's also widely applied for real-time prediction as it's not as computationally expensive as other classification algorithms. The algorithm learns the probability of a new instance with particular features belonging to a class. The naïve Bayes classifier (NBC) is known for being a simple yet effective classifier amongst other machine learning classifiers as it is robust to noise in the data and can be extremely fast to execute. It is a probabilistic classifier that is based on Thomas Bayes' probability theorem, presented in section 6.1. The Bayesian classifier calculates explicit probabilities for a given hypothesis. In other words, the classifier assigns inputs into one of the m classes $\{C_1, C_2, \ldots, C_m\}$ based on features of those inputs. Further, the naïve Bayes algorithm makes a strong conditional independence assumption, which gives it the term "naïve". The underlying probability model is known as *the independent feature model*. The NBC assumes the occurrence of a select feature in a particular class is independent of the occurrence of any other feature. In reality, the features or predictive variables, may in fact depend on one another. However, we assume they independently contribute to the probability of existing in a particular class.

Moreover, with the conditional independence assumption, the complexity of the algorithm is reduced by substantially decreasing the number of parameters to be estimated when modelling a conditional probability. The number of parameters is reduced to $2n$, from $2(2^n - 1)$.

Throughout this section, we assume any given training data is discrete or real-valued.

6.1 The axioms of Kolmogorov (1933) and Definitions

The axioms of Kolmogorov

Let Ω denote the set of all possible events in the sample space, with a probability measure P defined over it. The probability of an event $A \in \Omega$ occurring is given by $P(A)$.

1. The probability of an event is a real non-negative number

$$P(A) \in \mathbb{R}, P(A) \geq 0 \ \ \forall A \in \Omega$$

2. $P(\Omega) = 1$

3. Any countable sequence of disjoint sets $\{A_1, A_2, \ldots, A_j, \ldots\}$ such that $\forall i, j \in \mathbb{R}, A_i \cap A_j = 0$ satisfies

$$P(A_1 \cup A_2 \cup \ldots \cup A_j \cup \ldots) = P(A_1) + P(A_2) + \ldots P(A_j) + \ldots$$

Or alternatively,

$$P(\bigcup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$$

4. The probability of an event A occurring, given that an event B occurred is given by the **conditional probability**

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

5. Events A and B are said to be independent if

$$P(A \cap B) = P(A)P(A)$$

Def. *Class probabilities* are frequencies of instances that belong to each class divided by the total number of instances.

Bayes Theorem

For any two events $A \subset S$ and $B \subset S$, the probability of A given B is given by

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Proof: By axiom 4, we know that the probability of events A and B occurring is the probability of event A occurring, times the probability of A occurring, yielding

$$P(A \cap B) = P(A)P(B|A) \tag{6.1}$$

Alternatively, this probability can be represented by the probability of B occurring, times the probability of event A occurring, given that B occurred. That is,

$$P(A) = P(B)P(A|B) \tag{6.2}$$

Equating the right-hand sides (1) and (2) yields,

$$P(A)P(B|A) = P(B)P(A|B) \tag{6.3}$$

Rearranging (6.3), we get the desired expression

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Def. Let X, Y,and Z be random variable. We say X is **conditionally independent** of Y given $Z \iff$ the probability distribution of X is independent of the $Y|X$.
That is, $\forall i, j, k \in \mathbb{Z}$,

$$P(X = x_i|Y = y_j, Z = z_k) = (X = x_i|Z = z_k)$$

6.2 Motivation

Bayesian statistics often uses *probability distributions* rather than point probabilities to compute the quantities used in Bayes Theorem. The purpose of a Bayesian statistic is to represent prior uncertainty regarding model parameters with probability distributions. The prior uncertainty is then updated with new data to produce a posterior probability. In other words, in Bayesian statistics, probability distributions can be used to represent any uncertain quantities or attributes.

Let $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)$ denote the n dimensional feature vector representing data samples. Let C be a random variable of representing m classes $C_1, C_2, \ldots, C_m$. Assume each θ_j is conditionally independent of each of the other $\theta_j's$ given C, and of the each subset of other $\theta_j's$ given C for every $j \in \{1, \ldots, n\}$. Expressing the Bayes theorem in terms of probability distributions, we get the **Bayes Formula**: $\forall\ i \in \{1, \ldots, m\}$

$$P(C = c_i|\vec{\theta}) = \frac{P(C = c_i)P(\vec{\theta}|C = c_i)}{P(\vec{\theta})} \qquad (6.4)$$

where $P(C = c_i)$ is the *prior probability* which describes the degree to which we believe the model describes reality, given all prior information. $P(\vec{\theta}|C = c_i)$ is the likelihood which describes how well the model predicts classes. $P(\vec{\theta})$ is a *normalizing constant*, meaning it's purpose is to ensure the posterior density integrates to one.

By applying Bayes theorem, we wish to calculate the posterior probability, or the degree to which we believe the model accurately describes reality, given all prior information and data. That is, we shall compute the *posterior probability* $P(C = c_i|\vec{\theta})$ using $P(\vec{\theta}|C = c_i), P(C = c_i)$ and $P(\vec{\theta})$. Using Bayes' theorem, we will compute the *posterior*

probability by updating the *prior probability*. In Bayesian statistics, a *posterior probability* is defined to be the updated probability of an event occurring, accounting for new information. In other words, the probability of event A occurring give event B occurred.

To implement equation (6.4) in application, we will have to impose the conditional independence assumption in situations where unknown attributes need to estimated. This assumption also simplifies the representation of equation (6.4).

Under the conditional independence assumption, Bayes formula becomes

$$P(C = c_i|\theta_1, \ldots, \theta_n) = \frac{P(\theta_1, \ldots, \theta_n|C = c_i)P(C = c_i)}{P(\theta_1, \ldots, \theta_n)}$$

Computationally, it is much easier to take the product of the individual conditional probabilities for each term from the training set. Thus we rewrite the posterior probability as follows.

$$P(C = c_i|\theta_1, \ldots, \theta_n) = \frac{[P(\theta_1|C = c_i)P(\theta_2|C = c_i) \ldots P(\theta_n|C = c_i)]P(C = c_i)}{P(\theta_1, \ldots, \theta_n)}$$

$$\implies P(C = c_i|\theta_1, \ldots, \theta_n) = \frac{P(C = c_i)\prod_{j=1}^{n} P(\theta_j|C = c_i)}{P(\theta_1, \ldots, \theta_n)}$$

$$\propto P(C = c_i)\prod_{j=1}^{n} P(\theta_j|C = c_i) \qquad (6.5)$$

Note that the proportionality in the last expression follows by independence of the class variable C since $P(\vec{\theta}) = P(\theta_1, \ldots, \theta_n)$ is constant for all classes $C_1, \ldots, C_m$.

6.3 Naïve Bayes Classifier

Recall the assumption that the features $\vec{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)$ are any **real** or **discrete-valued** attributes and C is any discrete-valued variable. Each θ_j is an n-dimensional vector for any integer $j \in \{1, \ldots, n\}$.

For a new instance $\vec{\theta}_{new} = (\theta_1, \theta_2, \ldots, \theta_n)$, Bayes formula (6.5) can be used to compute the probability C will take on for any given value, given the observed θ_j's of $\vec{\theta}_{new}$ and estimates for $P(C = c_i)$ and $P(\theta_j|C = c_i)$, for integer $j \in \{1, \ldots, n\}$. We are interested in the "best" classifier which will output the most probable value of C. That is, we wish to optimize the following expression.

$$C \leftarrow \underset{c_i}{\operatorname{argmax}}\ P(C = c_i) \prod_{j=1}^{n} P(\theta_j | C = c_i) \tag{6.6}$$

Remarks:

1. If $P(C = c_i)$'s are unknown, assume all prior probabilities are uniform. That is, $P(C = c_1) = P(C = c_2) = \ldots = P(C = c_m)$
2. $P(C = c_i)$'s may be estimated using $P(\hat{C}_i) = \frac{N_i}{N}$ where N_i is the total number of training samples in class C_i and N is the total number of samples sampled.

6.4 Parameter Estimation using Maximum Likelihood Estimation

Recall a naïve Bayes model consists of m possible classes (or labels), n features, and parameters. We define the model's parameter as follows;

- Let $q(c) = P(C)$ be the probability of observing the class C for any $i \in \{1, \ldots, m\}$. As this quantity is a probability, we impose the constraints $q(c) \geq 1$ and $\sum_{c=1}^{m} q(c) = 1$.
- Let $q_i(\theta_j|c) = P(\theta_j|C)$ be the probability of feature $j \in \{1, \ldots, n\}$ taking value θ_j, given class C. As this quantity is a probability, we impose the constraints $q_i(\theta_j|c) \geq 1$ and $\sum_{j=1}^{n} q_j(\theta_j|c) = 1$.

Equation (6.6) can be rewritten as

$$C \leftarrow \underset{c}{\operatorname{argmax}}\ q(c) \prod_{j=1}^{n} q_j(\theta_j|c) \tag{6.7}$$

We introduce an *identity function* for mathematical convenience. Let $I = 1$ if $I(c^{(r)} = c)$, and 0 otherwise. The maximum likelihood estimate for the prior and posterior distributions are given by

$$q(y) = \frac{\sum_{i=1}^{N} I(c^{(r)} = c)}{N} \quad \text{and} \quad q_j(\theta_j|c) = \frac{\sum_{i=1}^{N} I(c^{(r)} = c, \theta_j^{(r)} = \theta)}{\sum_{i=1}^{N} I(c^{(r)} = c)}$$

Since $I(c^{(r)})$ is the sum of observations having class c in the training data, we can rewrite the parameter estimates as

$$q(c) = \frac{count(c)}{N} \quad \text{and} \quad q_j(\theta_j|c) = \frac{count_j(\theta_j|c)}{count(y)}$$

Let $\vec{\beta} = (q(c), q_j(\theta_j|C))$. The *log-likelihood* function, which measures how well the estimated likelihood parameters fit the training data, is then given by

$$L(\vec{\beta}) = \sum_{r=1}^{N} \log \left\{ q(c^{(r)}) \prod_{j=1}^{n} q_j \left(\theta_j^{(r)} | c^{(r)} \right) \right\} \qquad \forall \quad r \in \{1, \ldots, N\}$$

$$= \sum_{r=1}^{N} \log q(c^{(r)}) + \sum_{r=1}^{N} \log \left\{ \prod_{j=1}^{n} q_j \left(\theta_j^{(r)} | c^{(r)} \right) \right\}$$

$$= \sum_{r=1}^{N} \log q(c^{(r)}) + \sum_{r=1}^{N} \sum_{j=1}^{n} \log \left\{ q_j \left(\theta_j^{(r)} | c^{(r)} \right) \right\} \quad (6.8)$$

The maximum-likelihood estimates are the parameter values that maximize $L(\vec{\beta})$.

6.5 Gaussian Naïve Bayes

A naïve Bayes model can either be Gaussian, multinomial, or Bernoulli. We focus on the Gaussian based model as it is often used for classification problems in which the input features θ_j's are real-valued. We assume θ_j's follow a normal, or Gaussian, distribution

$$P(\theta_j(\mu, \sigma)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{\theta-\mu}{\sigma})^2}$$

where μ and σ estimate the mean and variance. For all n features,

$$P(\vec{\theta}(\mu, \sigma)) = \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \prod_{j=1}^{n} e^{-\frac{1}{2}\frac{(\theta_j-\mu)^2}{\sigma^2}}$$

Similar to the fully discrete naïve Bayes classifier, we estimate the parameters using *maximum likelihood estimates*. The *log-likelihood* function is given by

$$\log P(\vec{\theta}(\mu, \sigma)) = -\frac{n}{2} \log 2\pi\sigma^2 - \sum_{j=1}^{n} -\frac{(\theta_j - \mu)^2}{2\sigma^2} \quad (6.9)$$

To determine $\hat{\mu}$, the estimate for the mean μ, we set the partial derivative of equation (6.9) with respect to μ to 0.

$$\frac{\partial}{\partial\mu} \log P(\vec{\theta}(\mu, \sigma)) = \frac{\partial}{\partial\mu} \left\{ -\frac{n}{2} \log 2\pi\sigma^2 - \sum_{j=1}^{n} -\frac{(\theta_j - \mu)^2}{2\sigma^2} \right\}$$

$$= \frac{\partial}{\partial \mu}\left\{ -\sum_{j=1}^{n} \frac{(\theta_j - \mu)^2}{2\sigma^2} \right\} = -\sum_{j=1}^{n} \frac{(\theta_j - \mu)}{\sigma^2}$$

$$\frac{\partial}{\partial \mu} \log P(\vec{\theta}(\mu, \sigma)) = -\sum_{j=1}^{n} \frac{(\theta_j - \mu)}{\sigma^2} := 0 \implies \frac{n\mu - \sum_{j=1}^{n} \theta_j}{\sigma^2} = 0$$

$$\implies \hat{\mu} = \frac{1}{n} \sum_{j=1}^{n} \theta_j$$

To get an estimate for the variance σ^2, we consider the partial derivative of equation (6.9) with respect to σ.

$$\frac{\partial}{\partial \sigma} \log P(\vec{\theta}(\mu, \sigma)) = \frac{\partial}{\partial \sigma}\left\{ -\frac{n}{2} \log 2\pi\sigma^2 - \sum_{j=1}^{n} -\frac{(\theta_j - \mu)^2}{2\sigma^2} \right\}$$

$$= -\frac{n}{\sigma} + \frac{\partial}{\partial \sigma}\left\{ -\sum_{j=1}^{n} \frac{(\theta_j - \mu)^2}{2\sigma^2} \right\}$$

$$= -\frac{n}{\sigma} + \sum_{j=1}^{n} \frac{(\theta_j - \mu)^2}{\sigma^3} \tag{6.10}$$

Next, set equation (6.10) equal to zero and solve for $\widehat{\sigma^2}$.

$$\frac{\partial}{\partial \sigma} \log P(\vec{\theta}(\mu, \sigma)) = -\frac{n}{\sigma} + \sum_{j=1}^{n} \frac{(\theta_j - \mu)^2}{\sigma^3} := 0$$

$$\implies \widehat{\sigma^2} = \frac{1}{n} \sum_{j=1}^{n} (\theta_j - \hat{\mu})^2$$

6.6 Advantages and Limitations

A disadvantage of using a naïve Bayes classifier is what is called *Zero Frequency*. The *Zero Frequency* problem occurs if the response variable is categorical, such as *diagnosis* from the WBC dataset, has an unobserved class in the training set, i.e. *diagnosis* has (say) a third class in the test set for inconclusive results. In such situations, the naïve Baye classifier fails by assigning a zero probability to the observed classes and so, the model is unable to make predictions for such data. Smoothing techniques such as *Laplace* estimation can be used to solve this problem and make the data more uniform. *Laplace smoothing* integrates a small sample correction, known as *pseudo-count*, into all the conditional probability estimates so that no 0 probabilities are assigned.

Another disadvantage of the naïve Bayes algorithm is its strong assumption of features in the dataset. This assumption does not necessarily reflect reality in most situations. However, if the independence assumption holds, a naïve Bayes model is likely to perform better than other classification models, especially when applied to small-scale datasets. However, even with the assumption of independence in place, a good naïve Bayes can be built using estimations. The assumption is also very powerful as it reduces the number of parameters in the model dramatically. The naïve Bayes algorithm also performs better on categorical input features and hence, it is often applied in the field of text mining. Furthermore, for real-valued input features as seen in the Wisconsin Breast Cancer dataset, naïve Bayes algorithm will perform best if the real-valued input variable distributions are Gaussian or close to Gaussian. Tuning a naïve Bayes model may involve the transformation of features that do not have a Gaussian distribution so that they could have a Gaussian distribution. Other methods to improve a given classifier include *bagging* and *boosting*. However, the naïve Bayes classifier is unresponsive to those techniques since there is no variance that needs to be minimized. Also, including highly correlated features in the model could worsen its performance as the importance of such features will be double-counted. Another way to improve the performance of a naïve Bayes model is to apply the logarithmic function to the probabilities. This trick is usually done to avoid running out of precision by multiplying multiple small probabilities. When using Naïve Bayes for prediction, we're interested in class probabilities as a sum, rather than a specific value. The expression derived for a naïve Bayes, equation (6.6), would take on the following form.

$$C \leftarrow \underset{c_i}{\operatorname{argmax}} \ \log P(C_i) \prod_{m=1}^{n} \log P(\theta_m | C_i) \tag{6.11}$$

Overall, the algorithm is fast and very robust. Due to its fast speed, it is could be used to make predictions in real-time. It is computationally extremely fast and easy to implement. It is also worth noting that the naïve Bayes classifier works well with high dimensional datasets and handles missing values easily.

6.7 Modelling in R

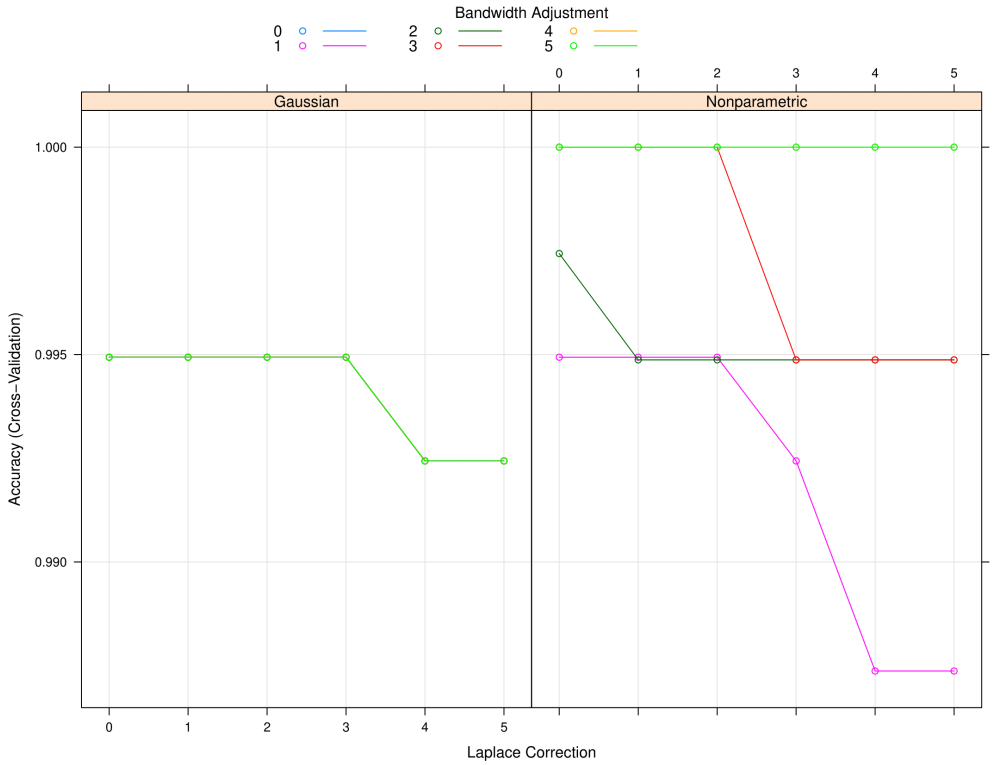
First, consider an efficient implementation of the naïve Bayes classifier in R to carry out the naïve Bayes modelling of the WBC dataset. Using the `naive_bayes()` function from the `naivebayes` library, we fit a model for each of the training subsets S_1 to S_6, as defined in the Feature Selection section. The features are assumed to be independent within each class C_i. The function assumes every feature is *possibly* distributed differently. This assumption may hold for any training subsets that do not contain similarly distributed

features, or possibly not contain any correlated features such as S_2. The `naive_bayes()` function also supports categorical conditional class distribution for discrete features and Gaussian distribution for continuous features. A different approach to building a Naïve Bayes model in R for the continuous features of the Wisconsin Breast Cancer data uses the specialized function `gaussian_naive_bayes()`. However, `naive_bayes()` models features with Gaussian distributions by default. Next, we also create predictions using the testing/validation sets for each model using the procedure defined in the Sampling Methodology section. Once again, the accuracy of the models gives an initial indication of model performance for selection of a single naïve Bayes model for the prediction of Breast Cancer.

The model trained using the complete training set of all 30 features performed the worst at 90.59%. The best performing model with 94.12% classification accuracy is seen to be the S_2 based model, using the top uncorrelated features in the dataset. This range of accuracy percentage is fairly low, therefore, we consider using k-fold cross-validation as described in Sampling Methodology, as well as tuning approaches using a different R libraries.

Using the `caret` library in R, we build an NB model with a 10-fold cross-validation step. The prediction accuracy the of models based on the training subsets S_1 to S_6, ranges from 95.29% for the full model (S_1) and 98.24% for the reduced model based on S_2. Not surprisingly, the accuracy results for all 6 subsets are an improvement to the results of the models trained without k-fold cross-validation. Using cross- validation, we improve the best naïve Bayes model's classification accuracy by approximately 4% (from 94.12% to 98.24%). We select the best performing S_2 based model for further analysis.

In an attempt to further improve the predictive performance of the naïve Bayes model, we can tune some of its' hyperparameters. We consider the use of a kernel density estimate to estimate class conditional densities of the features in the model. We also consider implementing a Gaussian density estimate. The other two hyperparameters of interest include adjusting the bandwidth of the kernel density and applying Laplace smoothing.



We analyze the reduced model's performance under different possibilities and we find that the optimal model uses a kernel density estimate with a *bandwidth* adjusted to 1 and no Laplace smoothing (see R Partial Output in Appendix). The final naïve Bayes model has classification accuracy of 98.24%.

7 Results Comparison

In this section, we compare the three machine learning models with respect to their efficiency and effectiveness. We refer to the table below as the **efficiency table** as we proceed in this section.

	TP	FP	Precision	Recall	F1 score
Logistic Regression	61 104	3 2	0.9531 0.9811	0.9683 0.9720	0.9606 0.9765
Support Vector Machine	61 107	2 0	1.0000 0.9817	0.9683 1.000	0.9839 0.9907
Naïve Bayes	61 106	2 1	0.9683 0.9907	0.9839 0.9815	0.9760 0.9860

The test set consists of 170 observations, 63 of which are *malignant*, and 107 which are *benign*. For the following calculations, we label the *malignant* scores in the colour pink and *benign* in blue.

7.1 Efficiency

ROC curve

A *receiver operating characteristic curve*, or *ROC curve*, is a graphical illustration of the diagnostic ability of a classifier. Given a new sample from the testing set, with known features and unknown diagnosis, the ROC curve will tell us how probable a given tumour is of being cancerous.

To further illustrate this idea, suppose we fit a logistic regression model to the Wisconsin Breast Cancer dataset. Plotting the ROC curve, the y-axis shows the probability that a given diagnosis is *malignant*, with the x-axis representing the feature vector. To classify unlabeled test samples, we need to transform the class probabilities to distinct classifications by setting a threshold of 0.5. Note that the threshold is between 0 and 1, as it is a probability. Next, we classify the data based on the defined threshold. Using a given model's *confusion matrix*, we calculate the sensitivity and specificity to evaluate the model when the threshold for cancerous diagnosis is 0.5. However, altering the threshold may result in possibly different classification of the data, and hence a different *confusion matrix*. It may result in better (or worse) classification of a class but it increases the number of false positives. For example, when lowering the models' threshold from 0.5 to (say) 0.1, the model corrects 3 incorrectly classified diagnoses, reducing the number of false positives and true positives. Therefore, we proceed with caution when finding the threshold level that will yield the best classification possible. In the real world, it is essential to correctly classify testing samples to ensure patients receive the

right treatment. To determine the best threshold, the ROC curve summarizes all the possible confusion matrices at every possible threshold.

The ROC curve plots the *True Positive Rate* (Sensitivity) against the *False Positive Rate* (1-Specificity) at various classification thresholds. The y-axis plots the *True Positive Rate*, defined as

$$TruePositiveRate = \frac{TruePositive}{TruePositive + FalseNegative}$$

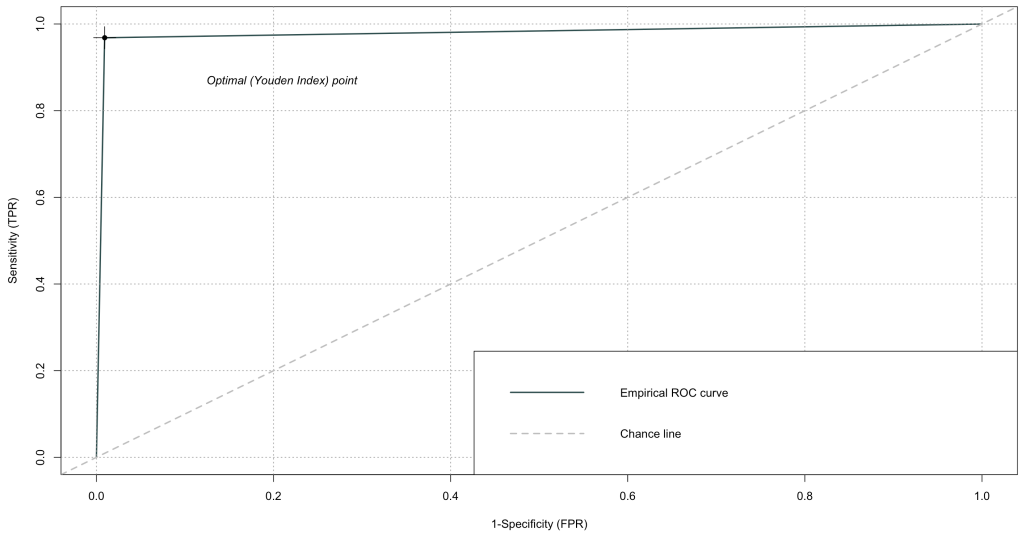
where *True Positives* are testing samples that were correctly classified diagnosis as *malignant* and *False Negatives* are the samples that were incorrectly classified as *benign*.

The *True Positive Rate* tells us the proportion of testing samples that were correctly classified as *malignant*. The x-axis plots the *False Positive Rate*, defined as

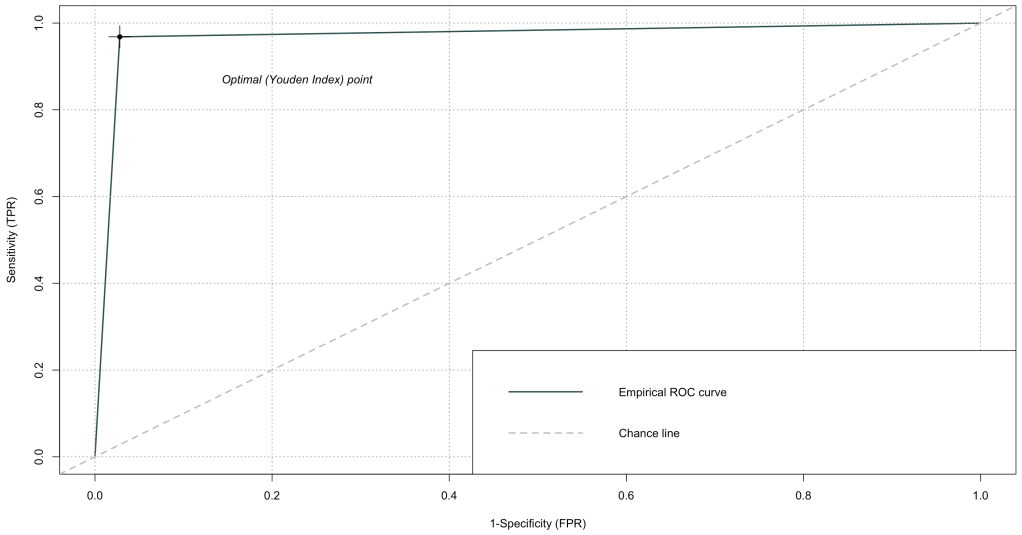
$$FalsePositiveRate = \frac{FalsePositive}{FalsePositive + TrueNegative}$$

where *False Positives* are testing samples that were incorrectly classified diagnosis as *malignant*, *True Negatives* were correctly as *benign*.

The *False Positive Rate* tells us the proportion of testing samples that were incorrectly classified. The ROC curve allows us to get a better sense of the diagnostic abilities of the three classifiers of interest. For instance, consider the ROC curve of "best" support vector machine model below. The curve is close to the top left of the graph, indicating good performance.



Comparing the ROC curve of the SVM model above, with the ROC curve of the logistic regression model, we see that the latter is further away from the top-left corner of the graph, indicating a weaker diagnostic ability. Referring to the *efficiency table*, we can confirm that the false positive rate for logistic regression of 104 is higher than that of support vector machine 107. The naïve Bayes model ROC curve is in between the two models, with a false positive rate of 106 out of 170 observations. [see Evaluation Metrics in the Appendix for more detail]



Remarks:

1. The true positive rate is also referred to as *sensitivity*, or *recall*.
2. The false positive rate is calculated by taking one minus the specificity rate, defined as
$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives}$$
Specificity is a measure of the proportion of negative data samples that got classified correctly by a given model.
3. For a different evaluation metric, the *False Positive Rate* is often replaced with *Precision*, which is defined as
$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$
Precision measures the proportion of positive (in this case *malignant*) test samples that were correctly classified by the model.

4. For heavily imbalanced data, *Precision* tends to be a better evaluation metric than the *False Positive Rate*. Unlike the *false positive rate*, *precision* does not account for the number of true negative testing samples.

5. In practice, studying rare diseases may result in a significant class imbalance. In such cases, *precision* can be a useful metric.

F1- Score

The F1-score is the harmonic mean of *Precision* and *Recall*, defined as follows.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

The F1-score is computed using the following equation

$$F1 = \frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

or equivalently,

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Remark: The F1-score is also known as F measure, or F score.

7.2 Effectiveness

	Features	Run-time	Accuracy	Kappa	AUC	F1 score
Logistic Regression	10	1.041s	0.9706	0.9243	0.9701	0.9765
Support Vector Machine	10	2.002s	0.9882	0.9620	0.9795	0.9907
Naïve Bayes	16	1.772s	0.9824	0.8722	0.9304	0.9860

7.2.1 Accuracy

To measure the accuracy of the three classification models, we evaluate the classification accuracy, Kappa statistic, and AUC score.

Classification Accuracy

Classification accuracy is defined as the ratio of the number of correctly predicted instances to the total number of predictions made. Analyzing the accuracy results shown in the *efficiency table* above, we see that the SVM model has a classification accuracy of 98.82%, followed by the naïve Bayes model with 98.24%, and the logistic regression model with 97.02%. Note that the results recorded in the table above correspond to the tuned models for each respective algorithm.

Cohne's Kappa Statistic (1892)

The Kappa statistic is a measure of inter-rater reliability between the actual class labels and the predictions. Here, we use the Kappa statistic as a comparison method for the overall accuracy of a given model and expected accuracy. The Kappa statistic ranges from 0 to 1, with 1 implying perfect agreement between the actual class and the predicted class and 0 implying the agreement was done by chance. In other words, a classifier with a Kappa statistic close to 1 is better than a classifier with a Kappa statistic closer to 0. Note that it is possible to have a negative Kappa value.

The equation used to calculate the kappa statistic is given by

$$K = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e}$$

where p_o is the *observed* proportional agreement between two rates and p_e is the *chance* agreement.

Analyzing the Kappa statistic corresponding to the three selected models, we see that all three models have high Kappa statistic values, indicating almost perfect agreement between the actual *diagnosis* and the predicted *diagnosis*. These results are consistent

with the F1-scores, and somewhat consistent with the accuracy results, with the SVM model leading. However, the logistic regression model has a higher Kappa statistic than that of the naïve Bayes model. This may be a result of only using uncorrelated features for the training of the naïve Bayes model, as those features are not necessarily the strongest predictive features in the dataset.

AUC

To evaluate the accuracy of a given model, we measure the area under the ROC curve, known as the Area Under the Curve, or AUC. The AUC efficient sorting-based algorithm is a widely used method of performance evaluation for classification models. The AUC measure provides a degree of separability of classes. The higher the AUC score is, the better the diagnosis prediction of new test data.

Both the logistic regression and SVM models have a relatively high AUC score of approximately 0.97, indicating their predictions are 97% correct. The naïve Bayes model has the lowest relative AUC score of 0.9304. Although the naïve Bayes algorithm generally performed better than logistic regression, it's AUC score is quite disappointing. This could be a result of the application on numeric data. Furthermore, recall that the naïve Bayes algorithm implements a very powerful independence assumption may not hold for numeric features as seen in the WBC dataset. [see naïve Bayes Advantages and Limitations]

7.2.2 Computational Complexity

In application, predictive models are often built on large-scale datasets, making the base algorithms more computationally expensive. A high run-time model can cost businesses resources, time, and money. Thus, considering run-time when analyzing the performance of a given classification model is essential.

The computational complexity of training a logistic regression model will depend on the run-time of the optimization problem used. Generally, training complexity is estimated by $\mathcal{O}(nd)$ where n is the total number of data points and d is the number of features.

The computational complexity of training a linear SVM model will depend on the run-time of the *dual* optimization which is given by $\mathcal{O}(\max(n,d), \min(n,d)^2)$ where n is the total number of data points and d is the number of features. The complexity of a linear SVM model is $\mathcal{O}(n_{sv}^3 + n(n_{sv} + n_{nsv}))$ where n_{sv} is the number of support vectors. For each training point, the algorithm requires $\mathcal{O}(n_{sv} + n_{nsv})$. For n training data points, we have $\mathcal{O}(n(n_{sv} + n_{nsv}))$. Note that approximations of an SVM model will also depend

on the number of iterations. The computational complexity of a kernel-based SVM, on the other hand, will depend on the choice of kernel used. Generally, it turns out that most kernel-based SVM take $\mathcal{O}(sd)$ time to run, where s is proportional to the number of support vectors and d is the number of features. In practice, different approximations of the SVM model are taken to reduce the computational complexity. An example of such approach is an SVM model with an RBF kernel [see Kernel Functions #4].

The computation complexity of the naïve Bayes classifier for training and testing are given by $\mathcal{O}(nd)$ and $\mathcal{O}(kd)$ respectively, where d is the number for features in the dataset, n is the total number of data points and k represents the label count of the response variance or *classes*. In the WBC dataset, k corresponding to the variable *diagnosis* is two since a given *diagnosis* example could be classified as either *malignant* or *benign*. This demonstrates how efficient the algorithm is. Being an efficient classifier, the naïve Bayes classifier it is extremely valuable for application to data of high dimension.

The run-time of the optimal naïve Bayes model chosen for the prediction of breast cancer diagnosis numeric data as seen in the WBC dataset is 1.772 seconds. In comparison with the SVM run-time, the naïve Bayes model had a relatively long run-time, given the algorithm is not as computationally expensive. This might be due to the number of features selected for the model. Unlike the other two models, our feature selection techniques result with 16 retained features that would yield the best possible classification of the data and the best prediction of unseen data. As expected, the SVM model has the longest run-time of 2.002 seconds. However, since the WBC dataset is fairly small, the training time for the selected SVM is approximately 0.012 seconds. Though it is important to note that for a large training set, one should avoid using an SVM model and rather implement a less computationally expensive algorithm such as naïve Bayes or logistic regression. We also see that with a larger training set, SVM training time decreases.

Based on the findings in this section, we conclude that a Support Vector Machine model is the best classification algorithm for classifying breast cancer tumours. The support vector machine classifier is followed by the logistic regression classifier and finally the poorest performer, the naïve Bayes classifier. Recall the selected SVM model is built using strong predictive features as determined by the `VSURF` algorithm. This experimental result is consistent with the results of related studies that suggest support vector machine dominates other machine learning algorithms in applications to medical data such as the WBC dataset.

8 Future Work

- Although the Wisconsin Breast Cancer dataset does not have significant data imbalance, exploring different techniques of ridding the dataset of any imbalance is probably worthwhile in the future.
- Another feature selection method that would be interesting to apply to the WBC dataset is a recursive one using a random forest classifier. As discussed earlier, the random forest algorithm assigns correlated features relatively lower importance, or significance. Applying this technique recursively, we can account for correlated features in the dataset.
- Naturally, the "best", or optimal, parameters for a logistic regression model are parameters that will minimize the objective function known as the cost function. Gradient descent is one of the most widely used optimization methods. Generally, it's often used to reduce the error in the model. In machine learning however, *gradient descent optimization* can be used update the parameters of a model. In the future, we may consider implementing the optimization technique gradient descent rather than the iteratively reweighted least squares (IRWLS) technique.
- One study of the WBC dataset suggests that an SVM-RBF kernel is the most accurate classifier of the data. It would also be worth while to implement an RBF kernel that uses *Euclidean* distance rather than *dynamic time warping* as a measure of distance. Although a *dynamic time warping* based RBF kernel would provide somewhat sufficient results, it does not satisfy the triangle inequality and is not positive semi-definite. Recall that valid kernel function must have positive semi-definite similarity matrices so that the data is expressed as dot products.
- In this study, we see the maximum likelihood estimation derivation of the naïve Bayes classifier [Parameter Estimation using Maximum Likelihood Estimation]. It would be interesting to examine the maximum a posteriori estimation (MAP) derivation of the classifier as it also estimates the prior distribution $P(C = c_i)$.

9 References

[1] Agarwal, A. (2019, September 24). Support Vector Machine - Formulation and Derivation. Retrieved July 13, 2020, from https://towardsdatascience.com/support-vector-machine-formulation-and-derivation-b146ce89f28

[2] Aragón-Royón F., Jiménez-Vílchez A., Arauzo-Azofra A., Benitez J. (2020). "FSinR: an exhaustive package for feature selection." arXiv e-prints, arXiv:2002.10330. 2002.10330, https://arxiv.org/abs/2002.10330

[3] Breast Cancer Wisconsin (Diagnostic) Data Set. (2016, September 25). Retrieved July 13, 2020, from https://www.kaggle.com/uciml/breast-cancer-wisconsin-data#data.csv

[4] Brownlee, J. (2019, August 12). naïve Bayes for Machine Learning. Retrieved July 13, 2020, from https://machinelearningmastery.com/na\unhbox\voidb@x\bgroup\accent127\OT1\i\protect\penalty\@M\hskip\z@skip\egroupve-bayes-for-machine-learning/

[5] Chapelle, O. "Training a support vector machine in the primal." Neural Computation 19.5 (2007): 1155-1178.

[6] Cordón I, García S, Fernández A, Herrera F (2018). "Imbalance: Oversampling algorithms for imbalanced classification in R", Knowledge-Based Systems, volume 161, pages 329-341

[7] Dubey, A. (2018, December 14). Feature Selection Using Random forest. Retrieved July 13, 2020, from https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f

[8] Fayed, L. (2020, April 8). When Was Cancer First Discovered? Retrieved August 1, 2020, from https://www.verywellhealth.com/the-history-of-cancer-514101

[9] Foot, R. (2020, March 2). Canadian Charter of Rights and Freedoms. Retrieved July 13, 2020, from https://www.thecanadianencyclopedia.ca/en/article/canadian-charter-of-rights-and-freedoms

[10] Genuer, R. (2019, July 18). Package 'VSURF.' Retrieved August 1, 2020, from https://cran.r-project.org/web/packages/VSURF/VSURF.pdf

[11] Genuer, R., Poggi, JM. and Tuleau-Malot, C. (2019). VSURF: Variable Selection Using Random Forests. R package version 1.1.0. https://CRAN.R-project.org/package=VSURF

[12] H. Lei and B. Sun (2007), "A Study on the Dynamic Time Warping in Kernel Machines" 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, pp. 839-845

[13] Liaw A. and Wiener M. (2002). Classification and Regression by randomForest. R News 2(3), 18–22.

[14] Harrad, W. (2020, February 18). A Top Machine Learning Algorithm Explained: Support Vector Machines (SVMs). Retrieved July 13, 2020, from https://www.vebuso.com/2020/02/a-top-machine-learning-algorithm-explained-support-vector-machines-svms/

[15] Khairunnahar, L. (2019, January 1). Classification of malignant and benign tissue with logistic regression. Retrieved July 13, 2020, from https://www.sciencedirect.com/science/article/pii/S2352914818301497

[16] Kuhn, M (2020). caret: Classification and Regression Training. R package version 6.0-86. https://CRAN.R-project.org/package=caret

[17] Lee, J. (2018, August 22). Support Vector Machines in R. Retrieved July 13, 2020, from https://www.datacamp.com/community/tutorials/support-vector-machines-r

[18] Majka, M (2019). naïvebayes: High Performance Implementation of the naïve Bayes Algorithm in R. R package version 0.9.7, https://CRAN.R-project.org/package=naivebayes

[19] Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F.(2019). e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.7-3. https://CRAN.R-project.org/package=e1071

[20] Mitchell, T. (2020, April 9). Generative and Discriminative Classifiers. Retrieved August 1, 2020, from https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

[21] Mitchell, T. M. (2011, January 25). Machine Learning. Retrieved August 1, 2020, from https://www.cs.cmu.edu/~tom/10701_sp11/slides/GNB_1-25-2011.pdf

[22] Patel, S. (2018a, June 21). Chapter 1 : Supervised Learning and naïve Bayes Classification — Part 1 (Theory). Retrieved August 1, 2020, from https://medium.com/machine-learning-101/chapter-1-supervised-learning-and-naive-bayes-classification-part-1-theory-8b9e361897d5

[23] Patel, S. (2018, November 10). Theory - Machine Learning 101. Retrieved July 13, 2020, from https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

[24] R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[25] Ray, S. (2020a, April 1). 6 Easy Steps to Learn Naïve Bayes Algorithm with codes in Python and R. Retrieved July 13, 2020, from https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[26] Ray, S. (2020, April 15). Understanding Support Vector Machine (SVM) algorithm from examples. Retrieved July 13, 2020, from https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[27] Robin, X., Turck, N., Hainard, A., Tiberti, N., Lisacek F., Sanchez, JC., and Müller M.(2011). pROC: an open-source package for R and S+ to analyze and compare ROC curves. BMC Bioinformatics, 12, p. 77. http://www.biomedcentral.com/1471-2105/12/77/

[28] Saabas, A. Selecting good features – Part III: random forests — Diving into data. (2014, December 1). Retrieved August 1, 2020, from https://blog.datadive.net/selecting-good-features-part-iii-random-forests/

[29] Saito, T. and Rehmsmeier, M. (2017). Precrec: fast and accurate precision-recall and ROC curve calculations in R. Bioinformatics (2017) 33 (1): 145-147

[30] Saitta, L. (n.d.). Support-Vector Networks. Retrieved August 1, 2020, from https://link.springer.com/content/pdf/10.1007/BF00994018.pdf

[31] Wang, L. (2005). Support Vector Machines: Theory and Applications (Studies in Fuzziness and Soft Computing) (2005th ed.). Auckland, New Zealand: Springer.

[32] W.H.C. (2019, November 20). Leading Causes of Death-Females-All races/origins. Retrieved August 1, 2020, from https://www.cdc.gov/women/lcod/2017/all-races-origins/index.htm

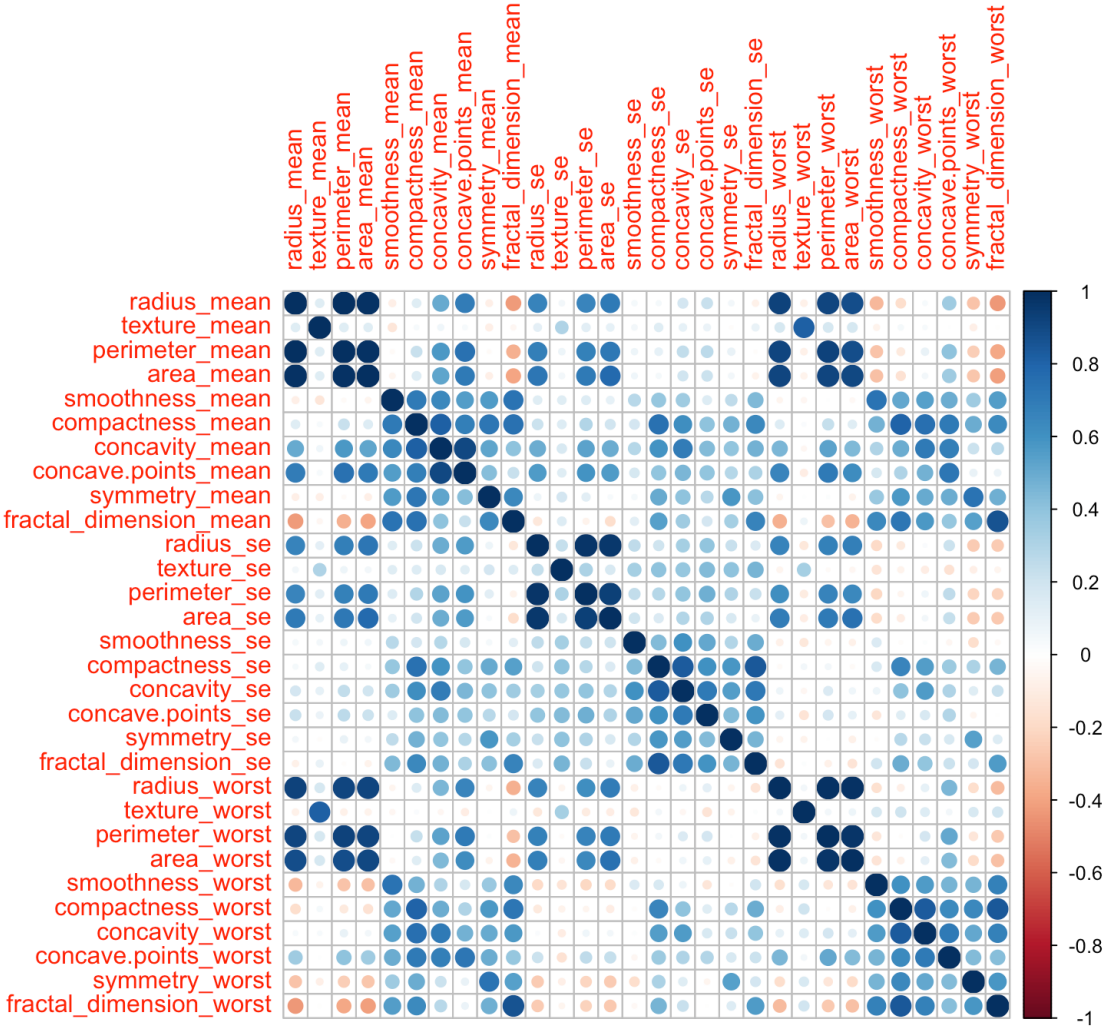
[33] Wickham, H., François, R., Henry, L. and Müller, K (2020). dplyr: A Grammar of Data Manipulation. R package version 0.8.5. https://CRAN.R-project.org/package=dplyr

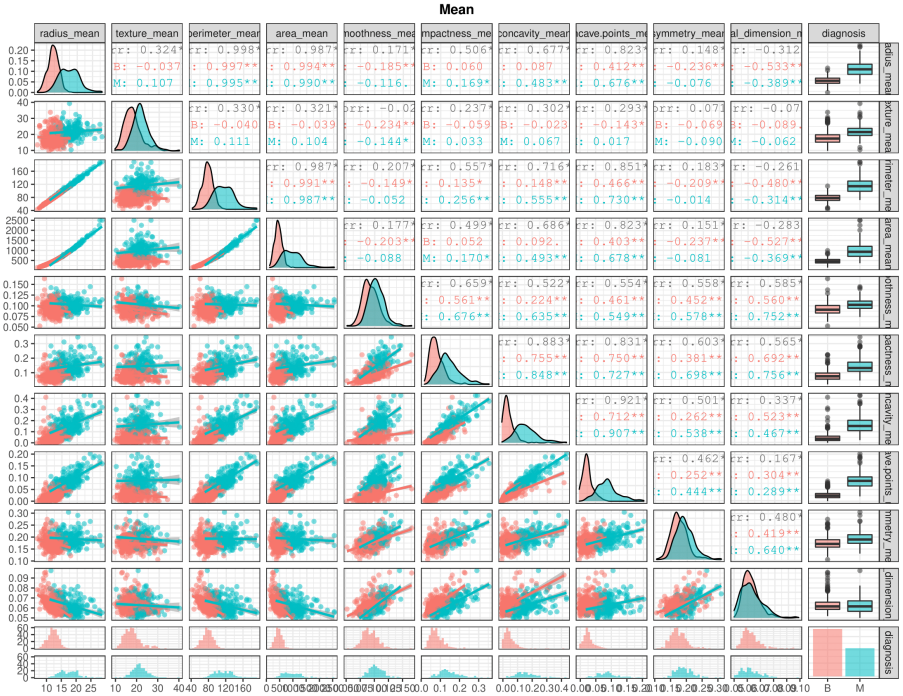
[34] Yan, Y. (2016). MLmetrics: Machine Learning Evaluation Metrics. R package version 1.1.1. https://CRAN.R-project.org/package=MLmetrics

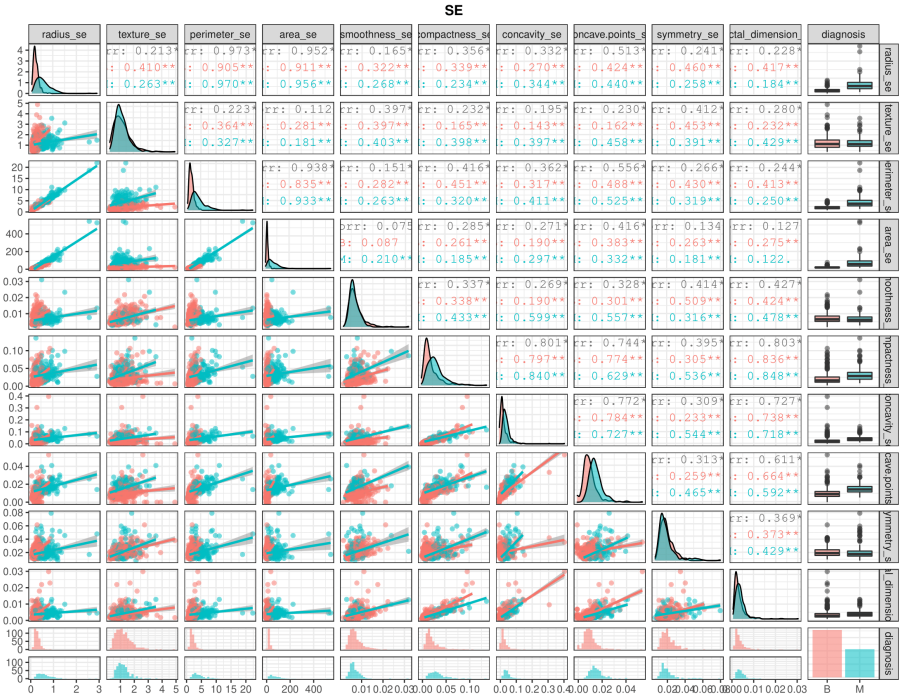
10 Appendix

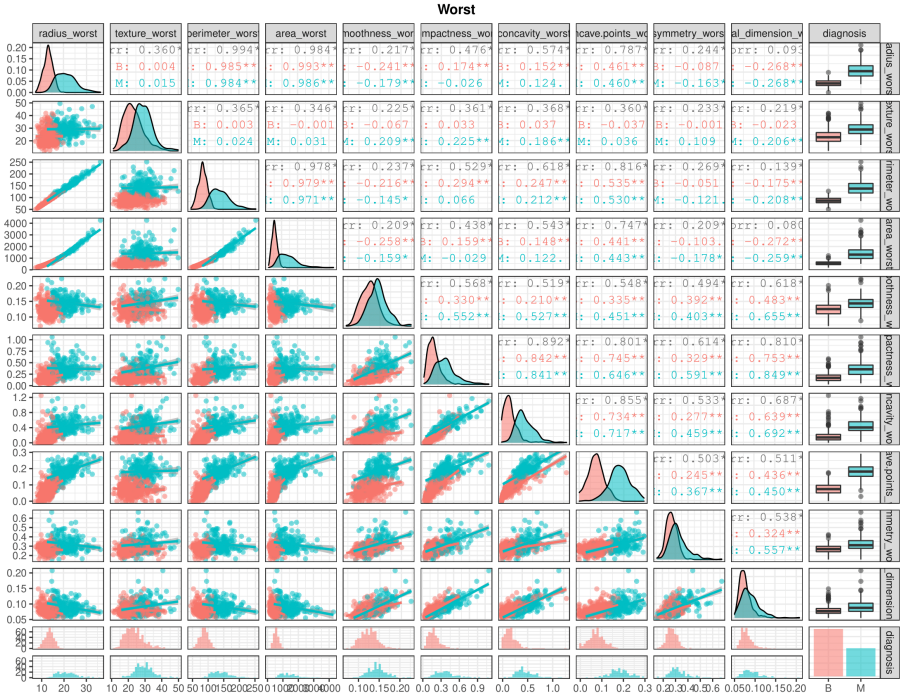
10.1 Figures

10.1.1 Correlation Plots and Box Plots

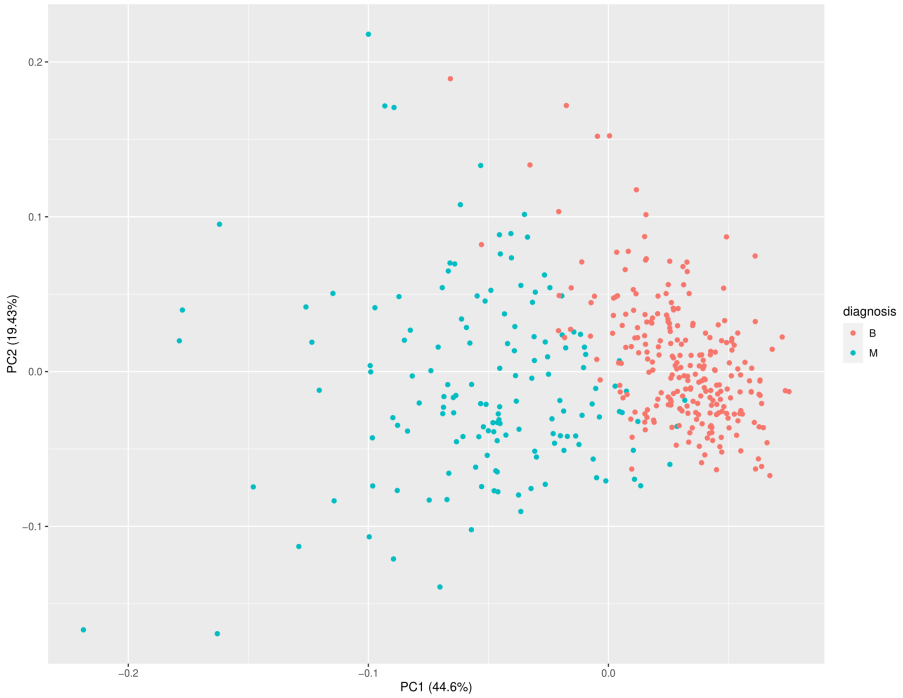



Mean
radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
concave.points_mean
symmetry_mean
fractal_dimension_mean
diagnosis
B
M


SE
radius_se
texture_se
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave.points_se
symmetry_se
fractal_dimension_se
diagnosis
B
M

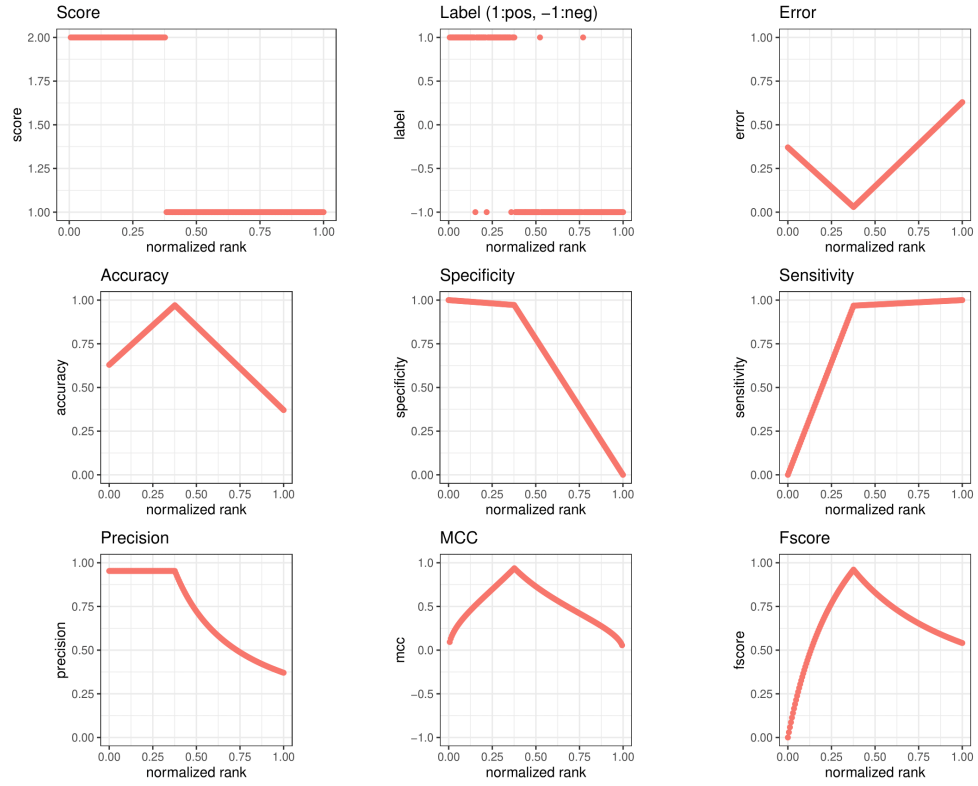


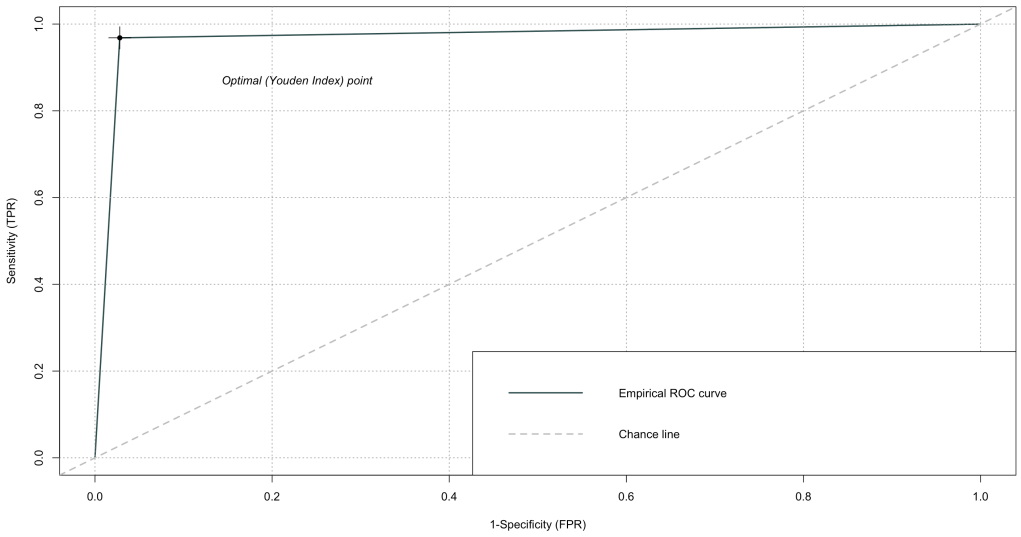
PCA Autoplot



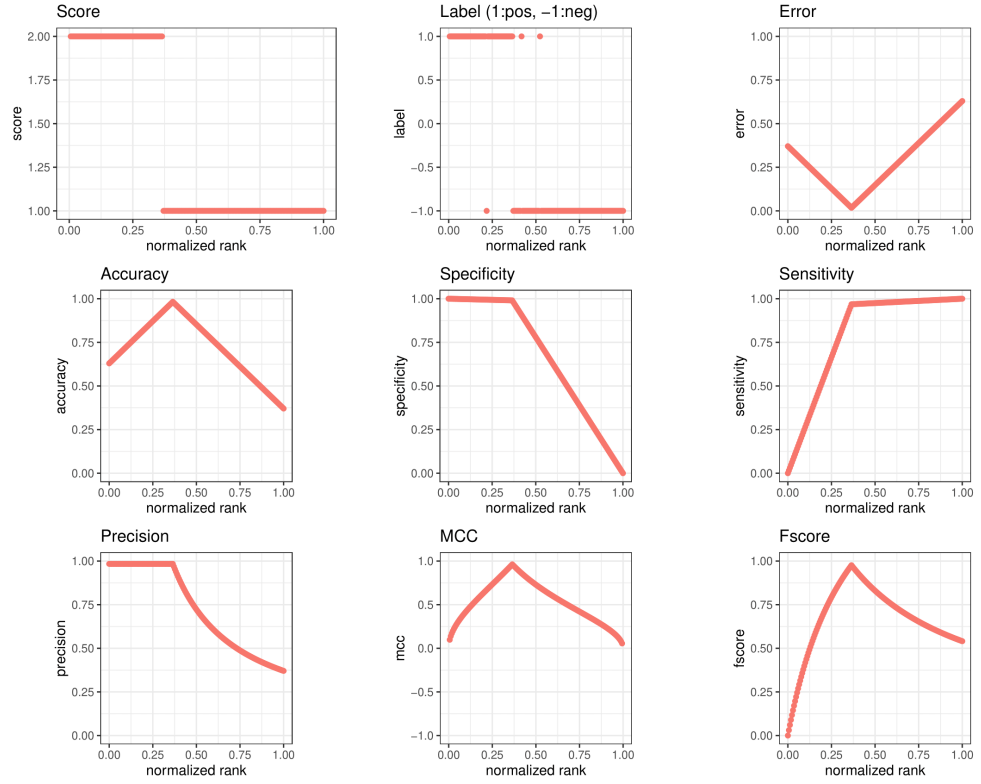
10.1.2 Evaluation Metrics

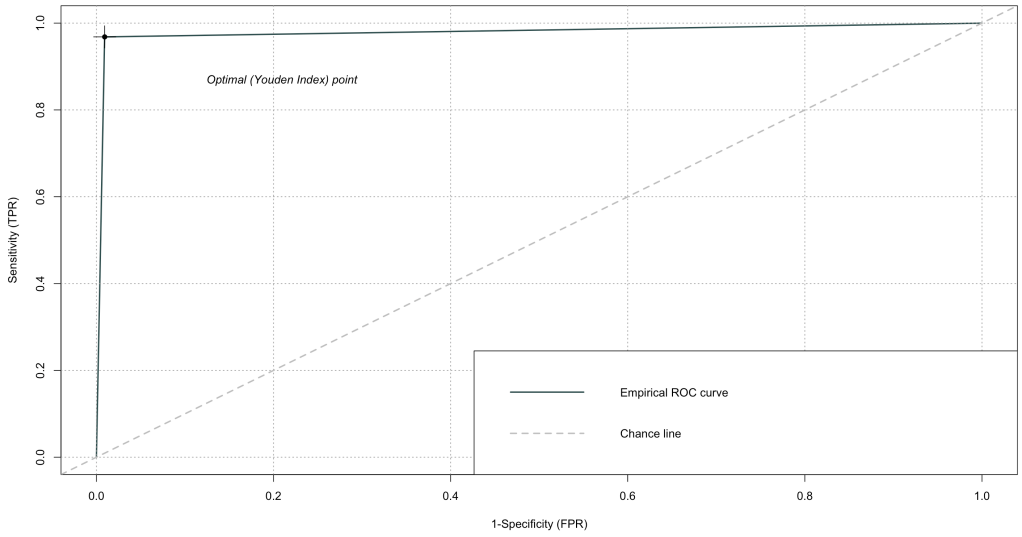
Logistic Regression



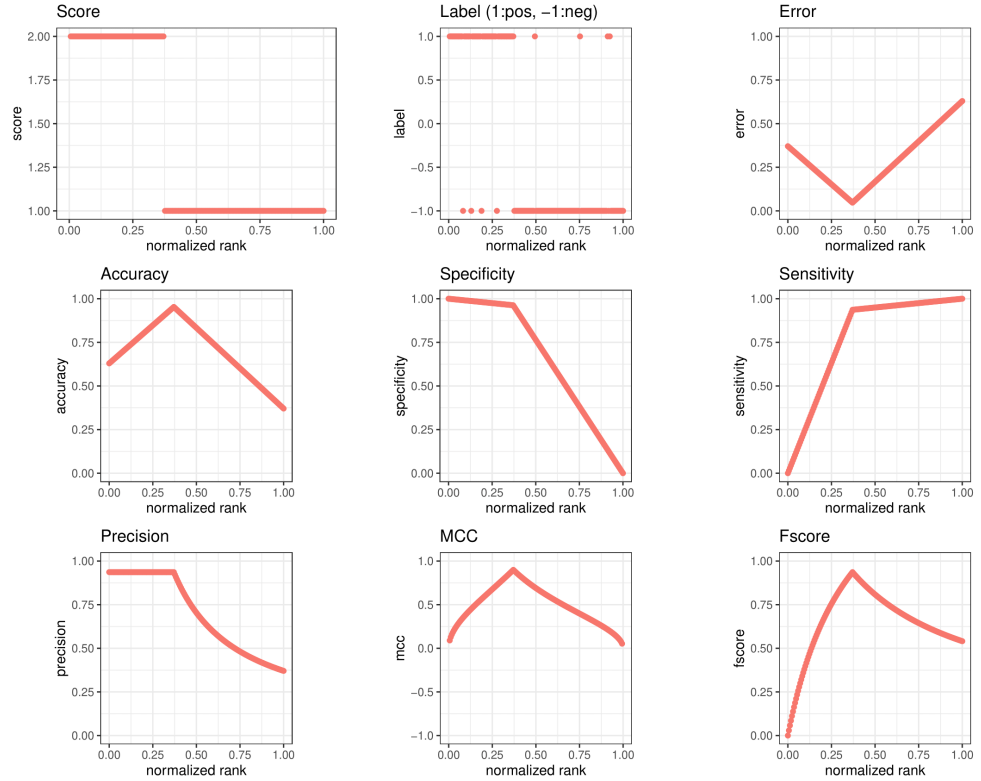


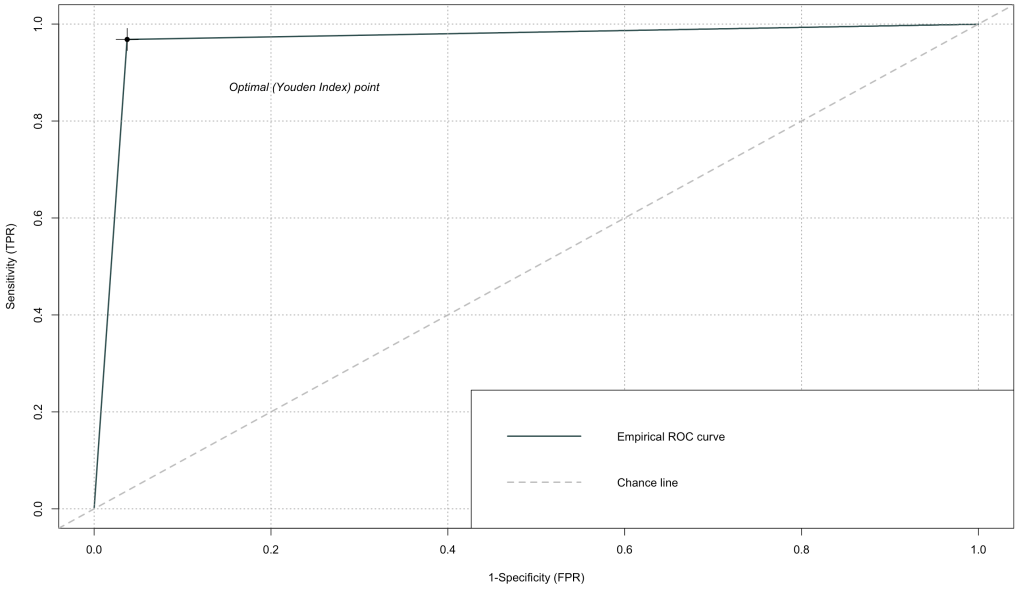
Support Vector Machine





Naïve Bayes


Score
score
normalized rank
Label (1:pos, −1:neg)
label
normalized rank
Error
error
normalized rank
Accuracy
accuracy
normalized rank
Specificity
specificity
normalized rank
Sensitivity
sensitivity
normalized rank
Precision
precision
normalized rank
MCC
mcc
normalized rank
Fscore
fscore
normalized rank


Optimal (Youden Index) point
Sensitivity (TPR)
1-Specificity (FPR)
Empirical ROC curve
Chance line

10.2 R

10.2.1 Code

The complete R code used in the construction of this project is documented in a GitHub repository found here.

10.2.2 Partial Output

PCA Feature Frequency - R output

```
> random.max
      [,1] [,2]
 [1,]    1    1
 [2,]    1    1
 [3,]    1    1
 [4,]    2   22
 [5,]    1    1
 [6,]    1    1
 [7,]    1    1
 [8,]   12    1
 [9,]    1    1
[10,]    9    1
> var.freq<-as.data.frame(table(unlist(random.max)))
> var.freq
  Var1 Freq
1    1   16
2    2    1
3    9    1
4   12    1
5   22    1
```

VSURF - R output

```
> rf_vsruf <- VSURF(diagnosis ~ ., data=train, ntree=50, mtry=2)  #long run-time
Thresholding step
Estimated computational time (on one core): 1.9 sec.
  |=============================================================================================| 100%
Interpretation step (on 28 variables)
Estimated computational time (on one core): between 42.7 sec. and  126.7 sec.
  |=============================================================================================| 100%
Prediction step (on 14 variables)
Maximum estimated computational time (on one core): 39.9 sec.
  |=============================================================================================| 100%
```

Naïve Bayes Model Tuning - R output

```
Naive Bayes

399 samples
 17 predictor
  2 classes: 'B', 'M'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 359, 359, 360, 359, 359, 359, ...
Resampling results across tuning parameters:

  usekernel  Accuracy   Kappa
  FALSE      0.9698718  0.9361978
   TRUE      0.9924359  0.9838462

Tuning parameter 'fL' was held constant at a value of 0
Tuning parameter 'adjust' was held constant at
 a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.
```