IS5152 Data-Driven Decision Making

Breast Cancer Diagnostic Dataset Analysis

Submitted on 26th April 2020

Group Members

Name: Amaratunga Vithanage Akila Ravihansa Perera **Student ID:** A0212216X **Email:** e0503495@u.nus.edu	**Name:** Vadakkencherry Govindan Sreelakshmy **Student ID:** A0212240A **Email:** e0503519@u.nus.edu
Name: Low Zi Jie **Student ID:** A0149304J **Email:** e0014190@u.nus.edu	**Name:** Tay Kong Yong **Student ID:** A0194930Y **Email:** e0383342@u.nus.edu

Table of Contents

1. Executive Summary

The existing problems of inaccurate screening results and scarce medical resources continue to affect the health of millions of women worldwide and strain the countries' medical infrastructure and resources. It is the most common type of all cancers and is the key cause of women's death worldwide. These problems have accentuated the importance and need to improve the accuracy of improving the diagnostic results.

This report aims to provide an analysis and evaluation of how machine learning models could be leveraged on to make data-driven decisions, helping to increase the accuracy of the diagnostic results of the mammogram screening tests. A total of four algorithm methods are used for our analysis, namely, Naive Bayes Theorem, Support Vector Machines, Decision Tree Model and Logistic Regression. Furthermore, we build an ensemble model by combining all of four of the base models.

The approach involves the generation of multiple models using 70% of the raw data for the various algorithm methods mentioned. We seek to optimize the models by adopting the k-fold cross-validation technique as an evaluation method. The eventual optimized model for each algorithm is analysed further using various metrics after the test data which comprises the remaining 30% of the raw data is fed into the models. All the experimented models are executed within a simulation environment and conducted in the WEKA data mining tool and Python.

2. Business Problem

According to the World Health Organization (WHO), breast cancer remains one of the top ten issues plaguing women's health worldwide. A majority of the deaths occur in low and middle-income countries where access to screening and treatment are scarce[1]. To date, even the best screening tests, mammograms, are not 100% accurate. About 20% of the mammograms result in a false-negative wherein the cancer present was not revealed from the result of the mammogram[2]. Such high rates of false-negative could be worrying as the cancer was not treated early and left to spread in the body.

Likewise, the problem of false-positives can also be perilous as unnecessary treatments are done. Such treatments include chemotherapy, radiation therapy, and in more serious cases, mastectomy. These treatment methods administered on patients who are diagnosed wrongly based on the false-positive results may lead to unintended complications which can affect their health. In addition, the unnecessary treatment puts a strain on the cost and availability of the healthcare resources which are in shortage in most parts of the world.

Data mining and Classification models are an effective way to assist in predicting whether the presence of a tumour is benign or malignant. This could be very useful in the medical field where classification methods can leverage data to help in medical diagnosis and analytics to make decisions. To improve the assessment and accuracy of mammogram results, we explore the use of various machine learning models (Naive Bayes, Decision Trees, Support Vector Machine, Logistics Regression) to achieve more accurate results and reduce the false-negatives.

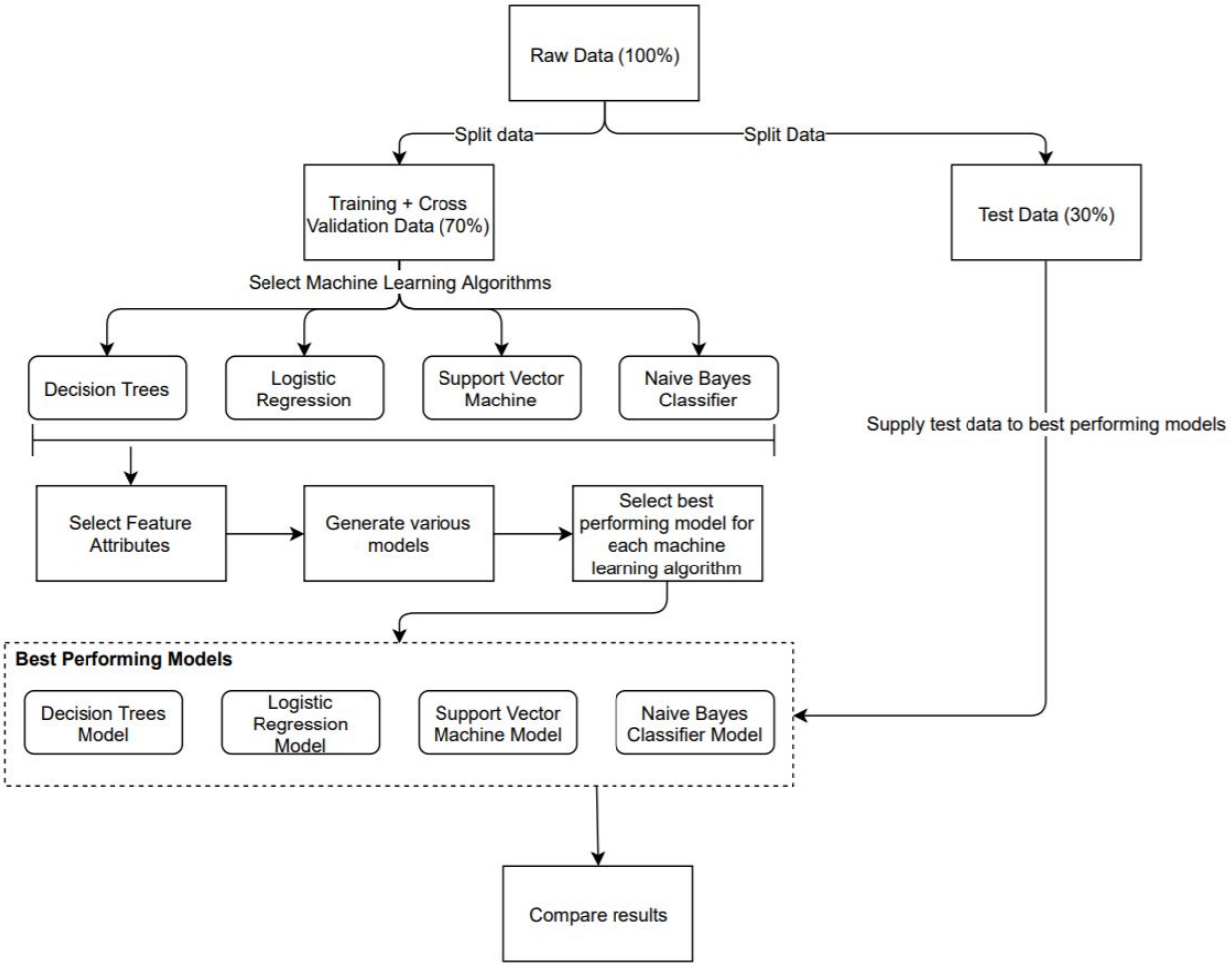
3. Data Attributes

The dataset used for our models is retrieved publicly from the UCI Machine Learning Repository[3]. The data attributes are results from a program used by Dr.William H.Wolberg that analyses the cytological features based on the digital scans of the solid breast masses from the patients. A summary table of the attributes in the dataset can be found in the Appendix section.

4. Methodology

The raw data is split using a resampling method without replacement into two datasets; "training + cross-validation data set" with records comprising 70% of the raw data sample and "test data set" with records comprising 30% of the raw data sample.

For the purpose of our analysis, we decided to employ four machine learning algorithms, namely, Decision Trees, Logistic Regression, Support Vector Machine and Naive Bayes Classifier. Within each of the algorithms identified, multiple models are generated and optimized using the k-fold cross-validation method. After the best-optimized model built using each of the machine learning algorithms is identified, we feed the model with the records from our test data set. The returned results are then analyzed through the use of various metrics that we have identified.



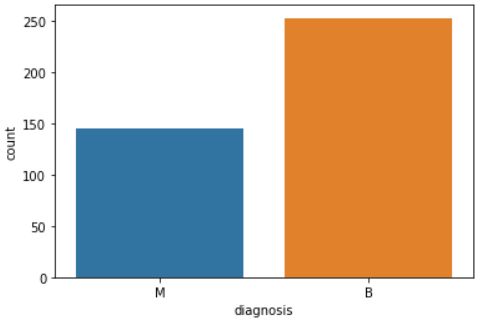
4.1 Searching for the best hyperparameters

A model hyperparameter is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter must be set before the learning process begins. For example, C (regularization parameter) in Support Vector Machines, K in k-Nearest Neighbours, and the number of hidden layers in Neural Networks. Conversely, the parameter is an internal state of the model that can be learned from the training data. For example, coefficients of logistic regression or support vectors in SVM. We used *Grid-Search* [9] technique to find the optimal hyperparameters of each model which results in the best 'recall' score. This is because in cancer prediction false negatives could mean that a patient with malignant cancer cells could be sent home which could be deadly. We use Python sklearn library's GridSearchCV class and specify the 'recall' as the metric to evaluate the models on.

5. Data Exploration

5.1 Target Attributes Distribution

We first took a look at our target variable, diagnosis. The target variable shows if the result is Malignant or Benign.



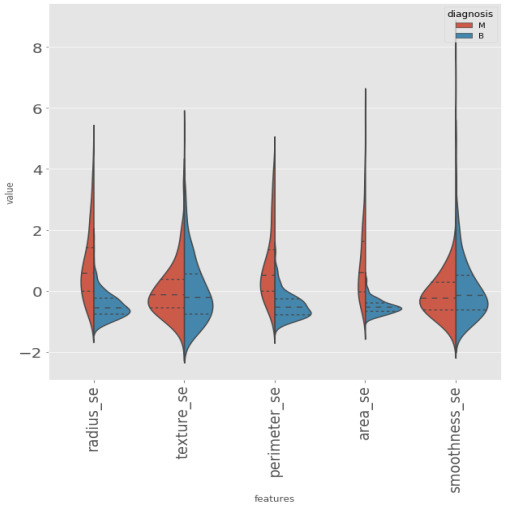
Out of the 398 observations, 36.4% (or 145 observations) are Malignant while the remaining 63.6% (or 253 observations) are Benign.

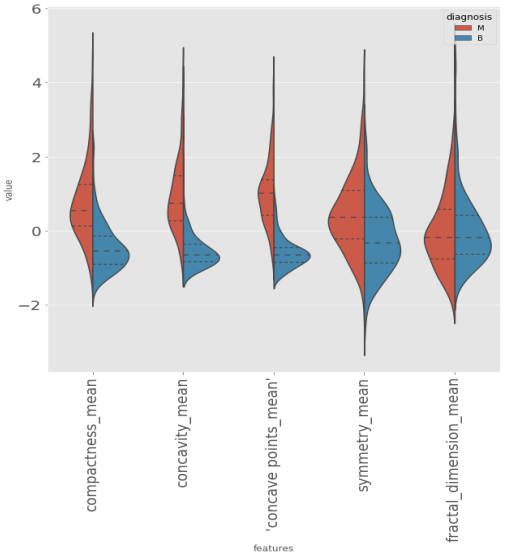
5.2 Feature Attributes Distribution

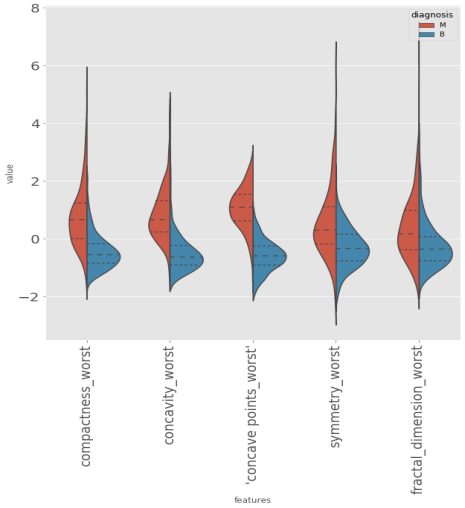
We looked into the distribution of our feature attributes. The summary table of key statistics for the distribution of the feature attributes can be found in the Appendix.

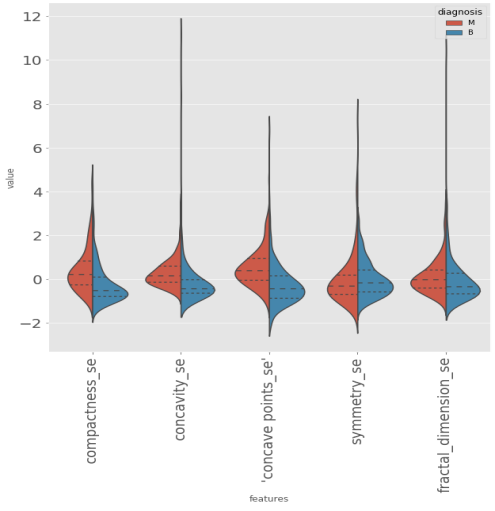
The data are then normalized based on the z-index as the scales of the data are different and hence, can be difficult to visualize. The features attributes are plotted in 3 different groups as below:

From the plot below, we notice that symmetry_mean, fractual_distribution_mean, symmetry_mean, fractual_dimension_mean, texture_se, smoothness_se, symmetry worst, fractual_dimension_worst does not look separated. Based on this preliminary study, they may not be very helpful for classification.



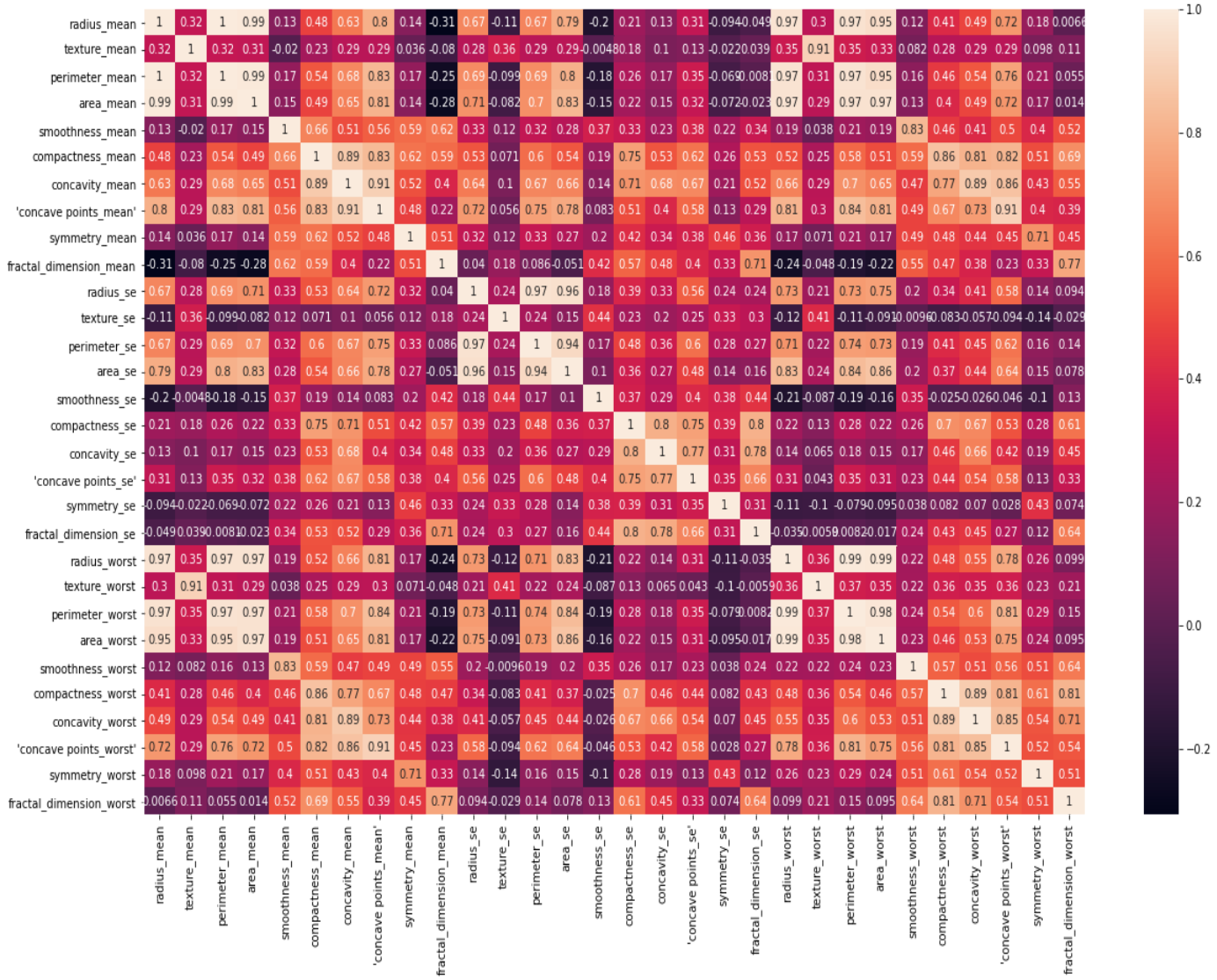






Correlation Heat Map

Below is the correlation heatmap across all the feature attributes. There seem to be a number of feature attributes that are highly correlated as displayed by the light orange colour used to indicate values greater than 0.8.



Pairplot

A pairplot among the attributes has also been prepared and is appended in the Appendix.

6. Data Cleaning

Using the pandas function, we found no missing or null data points in our data set.

```
id                         0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
diagnosis                  0
dtype: int64
```

7. Model Creation

7.1 Naive Bayes Theorem

7.1.1 Feature selection

We have identified the data column ‘id’ to be a non-predictive variable. Hence, the data column ‘id’ shall be excluded when generating the different Naïve Bayes Theorem models by varying the list of feature attributes used.

Further, we explored the use of three different feature attribute selection methods. Through these methods, we identify and use the selected feature attributes when generating three different Naïve Bayes Theorem models.

7.1.1.1 Correlation-Based Feature Selection

We used the correlation-based feature selection method to determine the collinearity between each of the feature attributes and the output variable, diagnosis. The returned results allow us to evaluate the worth of each feature attribute.

We utilized WEKA’s inbuilt functionality that employs the CorrelationAttributeEval technique with the use of the Ranker search method.

Based on the returned results found in the Appendix, we decided to use 0.4 as the cut-off to determine the feature attributes used in generating our first Naïve Bayes Theorem model (NBT-01). As such, we excluded the following feature attributes in our NBT-01 model; concave points_se, smoothness_mean, symmetry_mean, fractal_dimension_worst, compactness_se, concavity_se, fractal_dimension_se, smoothness_se, symmetry_se, texture_se, fractal_dimension_mean.

7.1.1.2 Information Gain Based Feature Selection

We employed the information gain based feature selection method to determine the entropy of each feature attribute for the output variable, diagnosis. Through this method, we can identify the feature attributes that contribute more information towards the prediction of the output class. This method allows us to retain only feature attributes identified to be useful and to remove attributes which do not add more information.

We used WEKA's feature selection function that utilized the InfoGainAttributeEval technique and the Ranker search method.

Based on the returned results found in the Appendix, we decided to use 0.3 as the cut-off to determine the feature attributes used for the generation of our second Naïve Bayes Theorem model (NBT-02). As such, we excluded the following feature attributes in our NBT-02 model; concavity_se, texture_worst, texture_mean, symmetry_worst, concave points_se, smoothness_worst, compactness_se, smoothness_mean, symmetry_mean, fractal_dimension_worst, fractal_dimension_se, fractal_dimension_mean, symmetry_se, texture_se, smoothness_se.

7.1.1.3 Learner Based Feature Selection

In this selection method, we aim to identify a subset of feature attributes that results in the best performance of the algorithm. We do this in WEKA by using the WrapperSubsetEval technique and BestFirst Search method. We selected the Best First Search method over the GreedyStepwise method as our dataset is of moderate size; hence we do not need to be concerned with the compute time. Navigate here for the parameters and configuration used in WEKA.

Based on the returned results found in the Appendix, the selected seven feature attributes that should be used in the generation of our third Naïve Bayes Theorem model (NBT-03) are compactness_se, radius_worst, texture_worst, perimeter_worst, smoothness_worst, concave points_worst, symmetry_worst.

7.1.2 Model Generation

Based on the different feature attributes identified via the three methods in section 1.1 Feature selection, the training data is cleaned further and is used to generate our three Naïve Bayes Theorem models. The models are trained based on the methodology explained in the section: Methodology. The parameters used in WEKA when generating each of the three Naïve Bayes Theorem models can be found in the following Appendix sections; NBT-01, NBT-02, NBT-03.

The summary of the weighted average results for the three NBT models is as follows:

	NBT-01	**NBT-02**	**NBT-03**
TP Rate	0.942	0.932	0.957
FP Rate	0.074	0.080	0.060

Precision	0.942	0.932	0.957
Recall	0.942	0.932	0.957
F-measure	0.942	0.932	0.957
MCC	0.875	0.853	0.907
ROC Area	0.988	0.984	0.986
PRC Area	0.987	0.984	0.987

Based on the table above, we can conclude that the model NBT-03, in general, is the best in predicting our output variable: diagnosis.

7.2 Decision Tree

Decision Tree is a supervised learning method which is used for classification and regression. Here, the model learns simple decision rules deduced from the data features and predicts the value of a target variable. We have used 5 types of models for classifying the data as malignant or benign. All five were implemented using the Weka tool. The classifiers used were weka.classifiers.trees.J48, weka.classifiers.trees.J48 Consolidated, and RandomForest, Bagging and boosting using J48. (Refer Appendix for further details).

7.2.1 weka.classifiers.trees.J48

This classifier is to generate a pruned or unpruned C4.5 decision tree[4]. The algorithm here has several parameters which can be configured. By default setting the confidence value is 0.25. The smaller values will incur more pruning. The decision tree formed is represented in Figure.7.2.1.

The settings in weka are given below.

Scheme: weka.classifiers.trees.J48 -C 0.25 -M 2 is the command which was given

7.2.2 weka.classifiers.trees.J48Consolidated

This classifier is to generate a pruned or unpruned C45 consolidated tree. This uses the Consolidated Tree Construction (CTC) algorithm which is developed by Pérez et al[5]. In this algorithm, a single tree is built based on a set of subsamples.

The settings in weka are given below.

Scheme: weka.classifiers.trees.J48Consolidated -C 0.25 -M 2 -Q 1 -RM-C -RM-N 99.0 -RM-B -2 -RM-D 50.0

The decision tree formed is represented in Figure.7. 2. 2

7.2.3 Bagging

Bootstrap estimation or Bagging is a statistical estimation technique where statistical quantities are estimated by random samples of data from the dataset. It is said to be useful when the dataset is limited[6].

The key parameter which has to be passed is the type of model which has to be bagged. The default is REPTree, but we pass J48 instead so that we can compare it with the usual J48 decision tree implementation.

Scheme: weka.classifiers.meta.Bagging -P 100 -S 1 -num-slots 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

7.2.4 Random Forest

Random forest is an extension of the bagging technique. This splits the tree a little differently than a usual decision tree, by selecting split points from a random subset of input attributes.

The problem with Decision trees with Bagging is that the trees which are produced look so similar to each other, which in turn reduces the variance of predictions from the bagging outputs. This affects the robustness of the decisions or predictions which are made.

Scheme: weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1

7.2.5 Boosting

The boosting algorithm used here is the Adaboost M1 method from Weka[7]. This classification model is said to improve performance drastically. But it sometimes overfits.

The first model is constructed normally, and each instance is given a weight according to the accuracy of the model and whether the data point is correctly classified or not. Then subsequent classification models are formed and evaluated until the accuracy is improved to its potential. Then all the predictions from these models are combined to increase the accuracy of prediction.

Scheme: weka.classifiers.meta.AdaBoostM1 -P 100 -S 1 -I 10 -W weka.classifiers.trees.J48 -- -C 0.25 -M 2

7.2.6 Model Evaluation

The models are trained based on the methodology explained in the section Methodology. The parameters used by WEKA when generating each of the 5 models explained above.

	J48 (DT-01)	**J48-consolidated (DT-02)**	**Random Forest (DT-03)**	**Bagging- J48 (DT-04)**	**Boosting - J48 (DT-05)**
TP Rate	0.937	0.937	0.952	0.957	0.970
FP Rate	0.080	0.071	0.060	0.051	0.041

Precision	0.937	0.937	0.952	0.957	0.970
Recall	0.937	0.937	0.952	0.957	0.970
f-measure	0.937	0.937	0.952	0.957	0.970
MCC	0.864	0.865	0.897	0.908	0.935
ROC Area	0.936	0.933	0.988	0.986	0.992
PRC Area	0.918	0.924	0.988	0.987	0.992

Table 7.2.1. Model comparison - Decision tree

From Table 7.2.1, there is a clear difference between the results of other models and boosting. The f-measure of boosting with the J48 decision tree clearly triumphs the other algorithms. Therefore, we conclude the model, DT-05, is the best Decision Tree model.

7.3 Support Vector Machine

Support vector machines (SVM) is a pattern classification technique proposed by Vapnik et al. (Boser et al., 1992). It aims to minimize the upper bound on the generalization error by maximizing the margin between a hyperplane separating the classes of data points.

7.3.1 Data scaling and model training

Next, we try to normalize the input dataset. This is required for breast cancer dataset since each feature has a different range. This will cause attributes with a higher value range to influence the result more due to its larger value. But a higher value for an attribute does not necessarily mean that it has high importance as a predictor.

Scale method (MinMax) - this method will transform the values so that the range of the values will be set to between 0 and 1 based on the minimum and maximum values in the series. It does not change the distribution of the dataset. Standard method - this method changes the values so that the standard deviation of an attribute's value distribution from the mean equals 1. We evaluated (using K-Fold, K=5) both scale and standard method for support vector-based classification and observed that both data scaling methods performed similarly.

Grid Search technique was employed to tune hyperparameters for Support Vector Machine model building. The recall scorer function was used to choose the best parameters that would yield the highest recall score. Table 7.3.1 shows the search space specified and the best hyper-parameters obtained using K-Fold cross-validation. We observed that the best hyper-parameter values generated from two sets of scaled training data (MinMax, Standard) were the same as shown in figure 7.3.1. However, standard scaling performed consistently better when fitted against the testing dataset as shown in figure 7.3.3. Therefore, it can be concluded that standard data scaling is more generalizable.

Hyper-parameter	Search Space	Best parameter value
C	[1, 2, 5, 10]	2
gamma	[0.0001, 0.01, 'scale']	scale
kernel	['rbf', 'linear']	rbf
degree	[2, 3, 5, 10]	2
decision_function_shape	['ovo', 'ovr']	ovo
tol	[1e-3, 1e-5, 1e-6]	0.001

Table 7.3.1. Hyper-parameter selection and search space for Support Vector Machine model

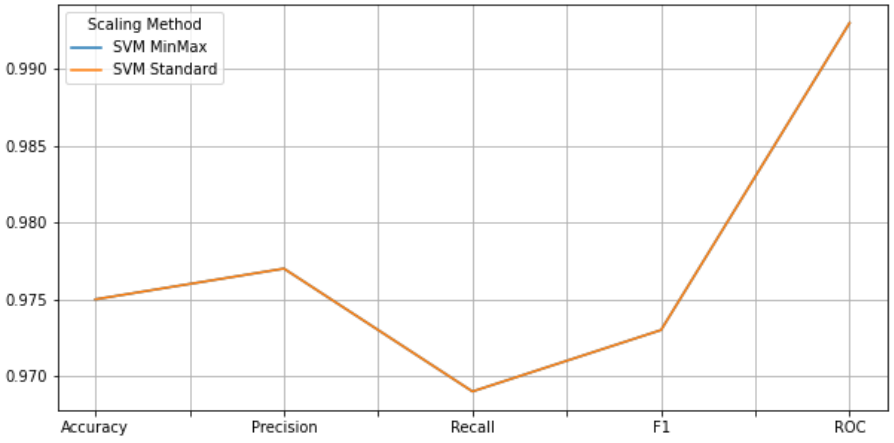


Figure 7.3.1. Performance of data scaling methods based on SVM model using K-Fold cross-validation

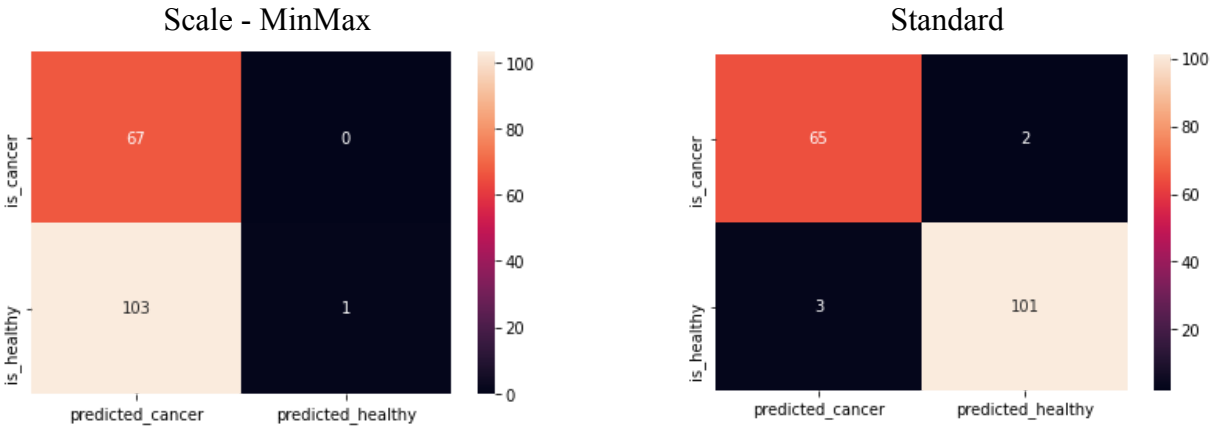


Figure 7.3.2. Confusion matrices of SVM models using different data scaling methods when fitted with the testing dataset.

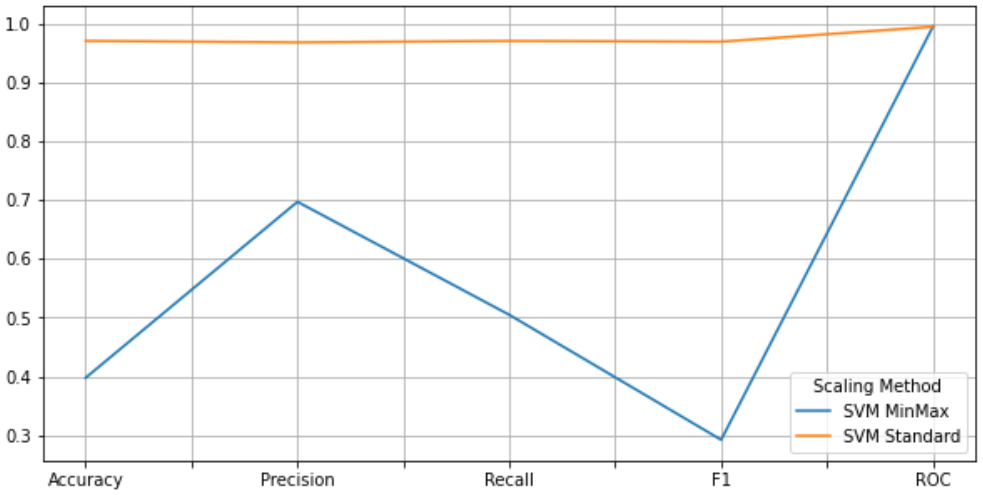


Figure 7.3.3. Performance of data scaling methods based on SVM model when fitted with the testing dataset.

7.3.2. SVM with Bagging

We attempted to improve the performance of the standard SVM model by employing the *bagging* technique which aims to build a combined model using several independent models and average their predictions in order to lower the variance. We used GridSearch to find the optimal hyper-parameters for bagging model based on SVM. Table 7.3.2 shows the search space and the best hyper-parameters obtained.

Hyper-parameter	Search Space	Best parameter value
base_estimator	All hyper-parameters listed in Table 7.3.1	C=1, decision_function_shape='ovr', degree=4, gamma=0.0001, kernel='rbf', tol=0.001
n_estimators	[20, 50, 100]	20
max_features	[0.2, 0.5, 1.0]	0.5
max_samples	[0.5, 0.8, 1.0]	1.0

Table 7.3.2. Hyper-parameter selection and search space for Support Vector Machine model with bagging technique

7.3.3. SVM with Boosting

Boosting method aims to build a combined model by iteratively fitting training dataset such that training of a particular model depends on the models fitted at the previous steps. We used AdaBoostClassifier class in Python scikit library to build a meta-estimator model. Since it behaves in an adaptive manner we expect the results to be less biased. We used GridSearch to find optimal hyper-parameters for boosting models based on SVM. Table 7.3.3 shows the search space and the best hyper-parameters obtained.

Hyper-parameter	Search Space	Best parameter value
base_estimator	All hyper-parameters listed in Table 7.3.1	C=1, decision_function_shape='ovo', degree=2, gamma=scale, kernel='rbf', tol=0.001
n_estimators	[20, 50, 100]	20
learning_rate	[0.2, 0.5, 1.0]	0.2

Table 7.3.3. Hyper-parameter selection and search space for Support Vector Machine model with boosting technique

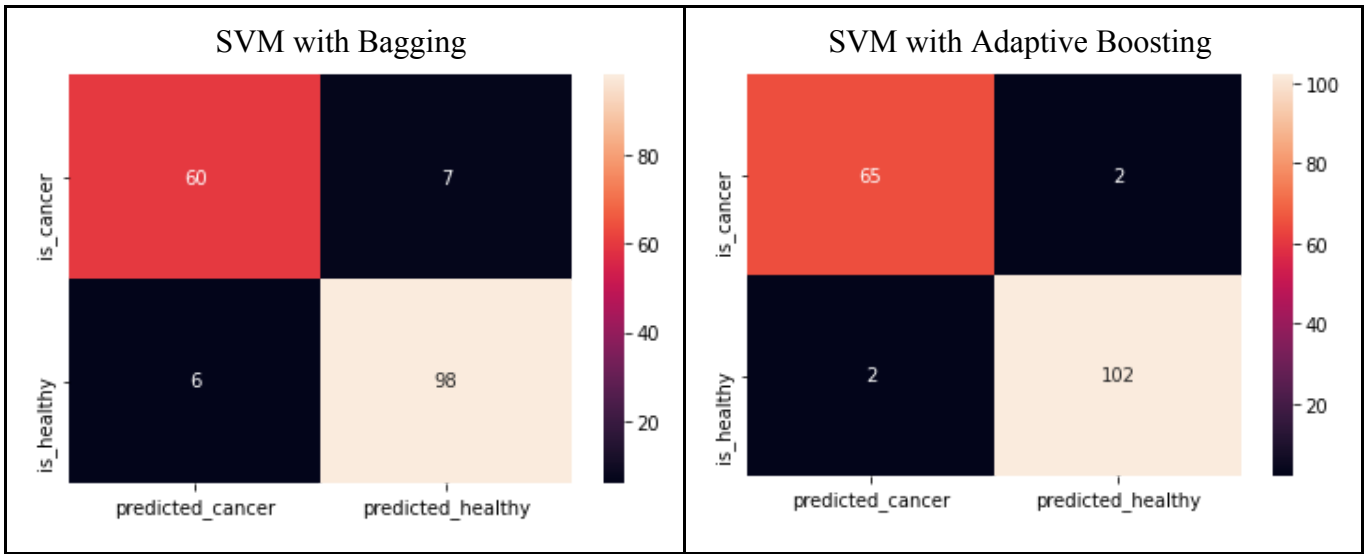


Figure 7.3.4. Confusion matrices of SVM models using bagging and boosting methods when fitted with testing dataset

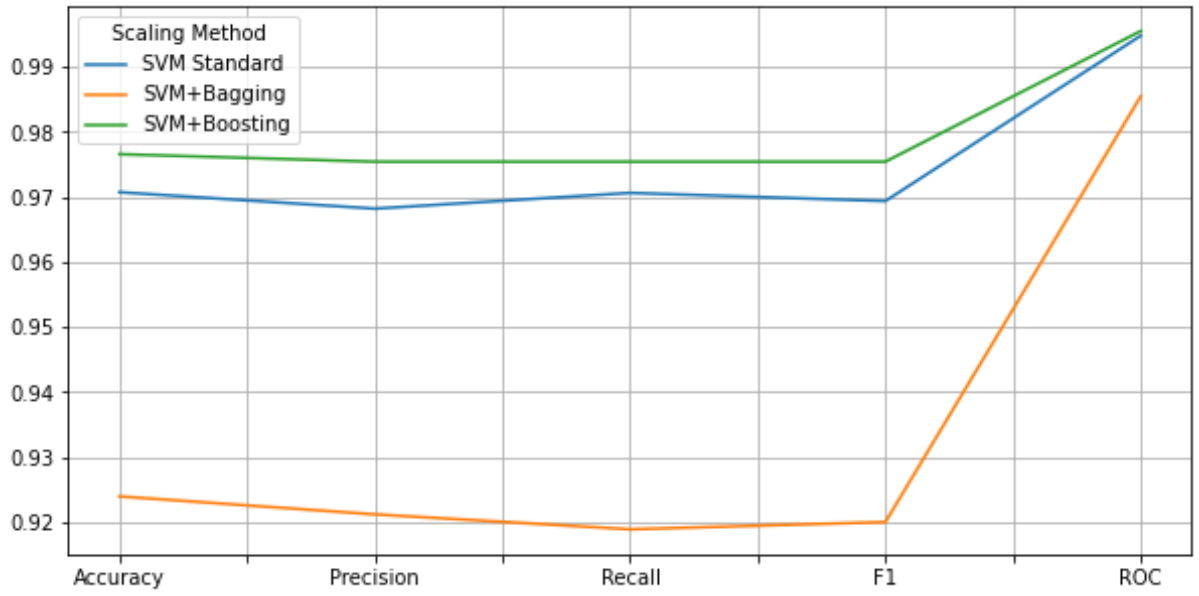


Figure 7.3.5. Performance of different SVM models when evaluated using K-Fold cross-validation method

According to figure 7.3.5, we can observe that SVM with boosting yielded better results across all evaluation metrics when compared to SVM standard and SVM with the bagging method. Interestingly, SVM with bagging model degraded the performance of the standard SVM model which could mean that the base model has a relatively low variance and higher bias. This could also be a result of the low number of input samples in the training dataset. Based on our findings we selected SVM with the adaptive boosting method to be our optimal SVM model to be compared against other models.

7.4 Logistics Regression Model

Logistic Regression is being used for classification – to classify each data point based on its feature attributes into the target attribute of whether the tumour is malignant or benign. Under Logistics Regressions, attributes are being inserted into linear regression functions which are then inputted into a logistic/sigmoid function. An iterative procedure is used to find the minimal cost function.

7.4.1 Model Coverage

We first ran a "vanilla" Logistics Regression with no regularisation and feature selection (using all attributes) as a base. Then, we run a variety of models possible to compare which models perform the best. The different models that are covered are:

1) LR1 - Logistic Regression with no regularization and feature selection
2) LR2 - Logistic Regression with L1 regularisation (Feature Selection – Embedded)
3) LR3 - Logistics Regression with Feature Selection Filter
4) LR4 - Logistics Regression with Feature Selection Wrapper - Backward Elimination
5) LR5- Logistic Regression with Feature Selection Wrapper - Recursive Feature Elimination

LR01 - Logistics Regression with no regularization

A "vanilla" Logistics Regression is being run on the data as a base. We used a python library sklearn to implement logistic regression on our data. The variables selected are:

On the other hand, we also attempt at manually running the logistic regression by our own code with gradient descent using python. (refer to appendix).

Refer to the Appendix for the list of attributes and its coefficient used for this method.

Refer to the Appendix for the result summary obtained from the Jupyter notebook.

On the other hand, we also attempt at manually running the logistic regression by our own code with gradient descent using python. (refer to the Appendix).

LR02 - Logistic Regression with L1 regularisation

Feature Selection – Embedded

In LR01, we used the maximum set of 32 attributes, which could lead to overfitting. To account for the possibility of over-fitting, we added a regularization term to the loss function. This term is an "absolute value of magnitude" of the coefficient. It acts as a penalty term to the loss function to shrink the less important feature's coefficient. L1 regularisation is used instead of L2 because we did not use coefficient polynomials above 1. This is being implemented using the python library sklearn. Adding the regularisation term has reduced the coefficients for these attributes to zero:

Perimeter_mean, area_mean, smoothness_mean, compactness_mean, concavity_mean, symmetry_mean, fractal_dimension_mean, radius_se, texture_se, perimeter_se, area_se, smoothness_se, compactness_se, concavity_se, concave points_se, symmetry_se, fractal_dimension_se, perimeter_worst, area_worst, smoothness_worst, compactness_worst concavity_worst, fractal_dimension_worst

Refer to the Appendix for the list of attributes and its coefficient used for this method, as well as the attributes that are being dropped.

Refer to the Appendix for the result summary obtained from the Jupyter notebook.

LR03 - Logistics Regression

Feature Selection - Filter

In this model, we apply the Filter method for our feature selection before implementing the logistics regression. Relevant attributes are selected before moving on to the learning phase. Pearson correlation of each attribute is being run against the target attribute. Attributes with correlations below a threshold of 50% are excluded from the learning phase. This is implemented manually through python code. Logistics regression is then being performed on the set of selected attributes. In this model, Feature selection is independent of the method used.

The remaining attributes that are used are : radius_mean, perimeter_mean, area_mean, compactness_mean, concavity_mean, concave points_mean, radius_se, perimeter_se, area_se, radius_worst, perimeter_worst, area_worst, compactness_worst, concavity_worst, concave points_worst.

Refer to the Appendix for the list of attributes and its coefficient used for this method, as well as the attributes that are being dropped.

Refer to the Appendix for the result summary obtained from the Jupyter notebook.

LR04 - Logistics Regression

Feature Selection Wrapper - Backward Elimination

In this model, we apply the Wrapper - Backward Elimination method for our feature selection before implementing the logistics regression. All possible features are being fed to the model initially. The performance of each attribute of the model is then checked. The attributes with the highest p-value and whose p-value is above 0.05 are being removed from the model. This process is performed iteratively

until the overall performance is in the acceptable range. This is implemented manually through python code. Logistics regression is then being performed with the set of selected attributes.

The remaining attributes are used: radius_mean, texture_mean, compactness_mean, concave points_mean, smoothness_se, concave points_se, fractal_dimension_se, radius_worst, area_worst, symmetry_worst, fractal_dimension_worst

Refer to the Appendix for the list of attributes and its coefficient used for this method, as well as the attributes that are being dropped.

Refer to the Appendix for the result summary obtained from the Jupyter notebook.

LR05 Logistics Regression

Feature Selection Recursive Feature Elimination(RFE)

In this model, we apply the Wrapper - Recursive Feature Elimination method for our feature selection before implementing the logistics regression. All features are being fed into the model initially. Recursive Feature elimination will remove attributes and run the logistic regression model on the smaller set of attributes. It will then rank the variables with 1 being the most importance. The least important attributes are then pruned from the set of attributes. This procedure is recursively repeated until the desired set of the number of attributes is reached. RFE is implemented by Python library sklearn. Additionally, we run a loop of size 1 to the full size of our 32 attributes with RFE manual implementation on Python to find the optimal number of features for the RFE. (refer to appendix) Logistics regression are then being performed with the set of selected attributes.

The optimal size of the attribute size is 23 and the following attributes are selected: radius_mean, texture_mean, perimeter_mean, area_mean, concavity_mean, concave points_mean, fractal_dimension_mean, radius_se, texture_se, perimeter_se, area_se, compactness_se, symmetry_se, fractal_dimension_se, radius_worst, texture_worst, perimeter_worst, area_worst, smoothness_worst, compactness_worst, concavity_worst, concave points_worst, symmetry_worst.

Refer to the Appendix for the list of attributes and its coefficient used for this method, as well as the attributes that are being dropped.

Refer to the Appendix for the result summary obtained from the Jupyter notebook.

Results:

We ran two sets of logistic regression for each of the models above, one set with a test split of 15% and 85% and the other set with 10-fold cross-validations. Below are the results:

Test-Splitting	**Accuracy**	**Precision**	**Recall**	**F1**
LR-01	0.975	0.98	0.97	0.97

LR-02	0.958	0.96	0.95	0.95
LR-03	0.95	0.95	0.95	0.95
LR-04	0.925	0.93	0.91	0.92
LR-05	0.95	0.95	0.94	0.95

10 Fold Cross Validations	**Accuracy**	**Precision**	**Recall**	**F1**	**ROC**
LR-01	0.96	0.959	0.955	0.956	0.986
LR-02	0.970	0.972	0.962	0.966	0.989
LR-03	0.95	0.948	0.942	0.944	0.988
LR-04	0.882	0.882	0.86	0.864	0.942
LR-05	0.952	0.95	0.944	0.946	0.989

Under Test-splitting, the best performing model is LR-01 Logistics Regression with no regularisation and feature selection. This high performance can be due to LR-01 overfitting to the training data and hence the model may not have the same accuracy when it is generalized across a new set of data.

Under 10 Fold Cross Validations, the best performing model is LR-02 Logistic Regression with L1 regularisation (Feature Selection-Embedded). All of the 5 metrics (Accuracy, Precision, Recall, F1 and ROC) have a higher performance under LR-02 than the rest of the model. There is also a drop in performance for LR-01 which performs the best in test-splitting. These suggest that the LR-01 experienced over-fitting as it cannot generalise across the "10-Folds". The regularisation term L1 in LR-02 has helped to control for the issue of overfitting.

We also tried to implement Bagging to the K-Fold results to see if there are any improvements in the models but we couldnt derive any conclusion from it.

The full summary of the result has been prepared in the Appendix.

7.5 Stacking - Meta-classifier Model

Stacking is an ensemble learning technique that aims to build a strong classifier by combining multiple heterogeneous weak classifiers via a meta-classifier. We use the StackingCVClassifier class in Python mlxtend library to build the ensemble model. We define our weak learners as individual models built previously (LR, SVM, NB, DT) and learn a multi-layer neural network as the meta-model. The outputs from our four weak learners will be fed into the neural network as inputs and it will learn to return final predictions based on it. We devised the following framework adapted from [8] to fit a stacking ensemble model composed of these four weak models as depicted in figure 7.5.2.

- Split the training data into five folds (K-Fold technique)
- Fit four weak learners with data from the 1st to 4th fold
- Each weak learner make predictions for observations in the 1st to 4th fold
- Fit the meta-model on the 5th fold, using predictions made by the weak learners

By using this *K-fold cross-training* approach we can use all the observations in the training dataset to train the meta-model. Therefore we can produce relevant predictions for each observation of our dataset and train our meta-model thus maximizing the utilization of limited training dataset.

Hyper-parameter	Search Space	Best parameter value
learning_rate	["constant", "adaptive"]	adaptive
hidden_layer_sizes	[(50,50,50), (50,100,50), (100,)]	(50,50,50)
solver	['sgd', 'adam']	sgd
activation	["logistic", "relu", "tanh"]	relu

Table 7.3.3. Hyper-parameter selection and search space for the meta-classifier model with stacking technique

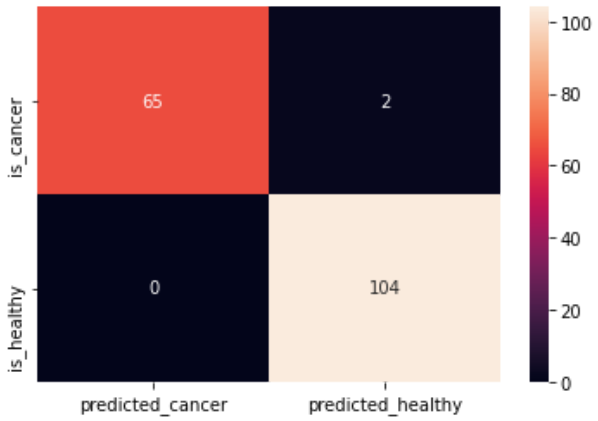


Figure 7.5.1. Confusion matrix of the meta-classifier model when fitted with the testing dataset.

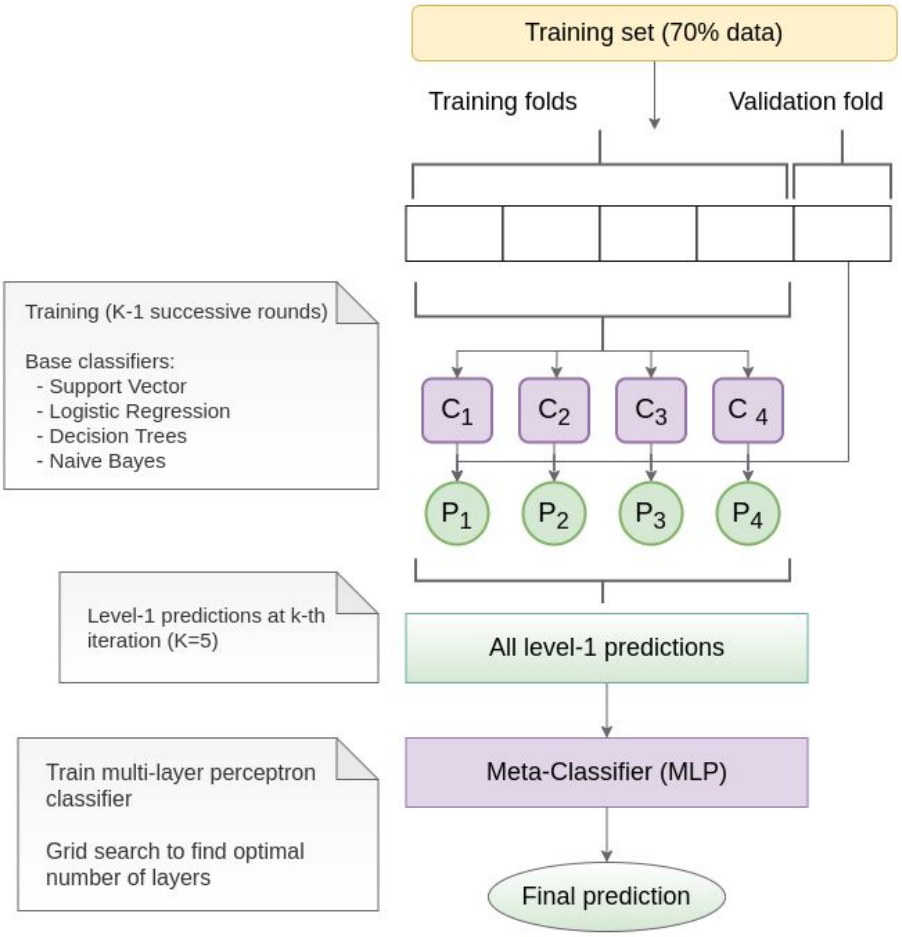


Figure 7.5.2. Cross-training framework for building meta-classifier based on stacking technique. This framework was adopted from [8]

8. Comparison of results between different algorithms

One of the important metrics we considered for comparison here is Recall. In a specific problem like cancer classification, minimizing false negatives become important. Because we do not want to classify malignant as benign, as it incurs a higher penalty. When a person who is not having breast cancer is classified as cancerous, there is minimal risk. However, this can be considered less dangerous than classifying a cancerous person as non-cancerous. So tracking a false negative is more important here than tracking false positive. Hence, more than precision, we have to concentrate on keeping the recall measure closer to 100%.

We evaluated the models that were built previously (illustrated in section 7) using K-Fold cross-validation technique and results are shown in table 8.1 in Appendix and figure 8.1. This evaluation was performed using only the training dataset (70% of the total dataset). It can be observed that SVM with Adaptive boosting method performs better in terms of accuracy, precision, recall, F1 and ROC compared to other models.

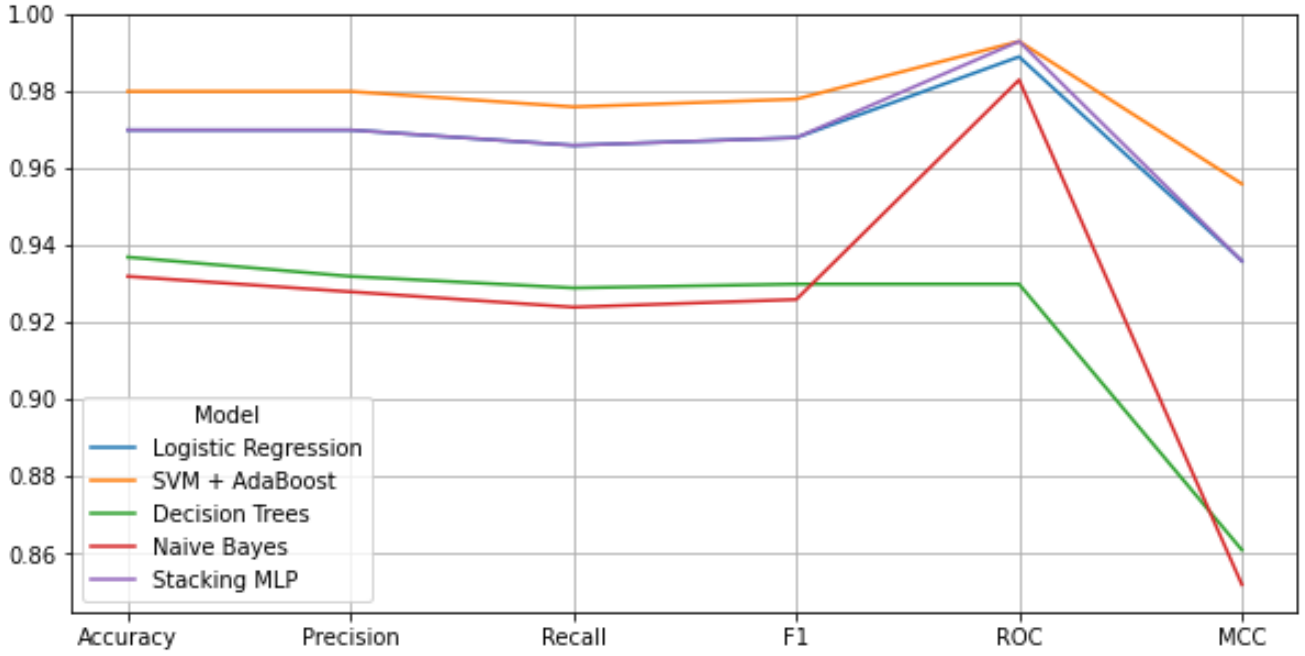


Figure 8.1. Performance of different models when fitted with training data using K-Fold cross-validation technique

Finally, we fitted the testing dataset (30% of overall dataset) on the best-optimized models from each learning algorithm. The evaluation scores of the predictions are shown in table 8.2 in Appendix and trend is depicted in figure 8.2. Interestingly, MLP with stacking model performs better than others across all metrics when fitted against the test dataset. This shows that ensemble methods can be utilized to build learning models that are generalized and provides predictions with low variance when tested with unseen real-world data.

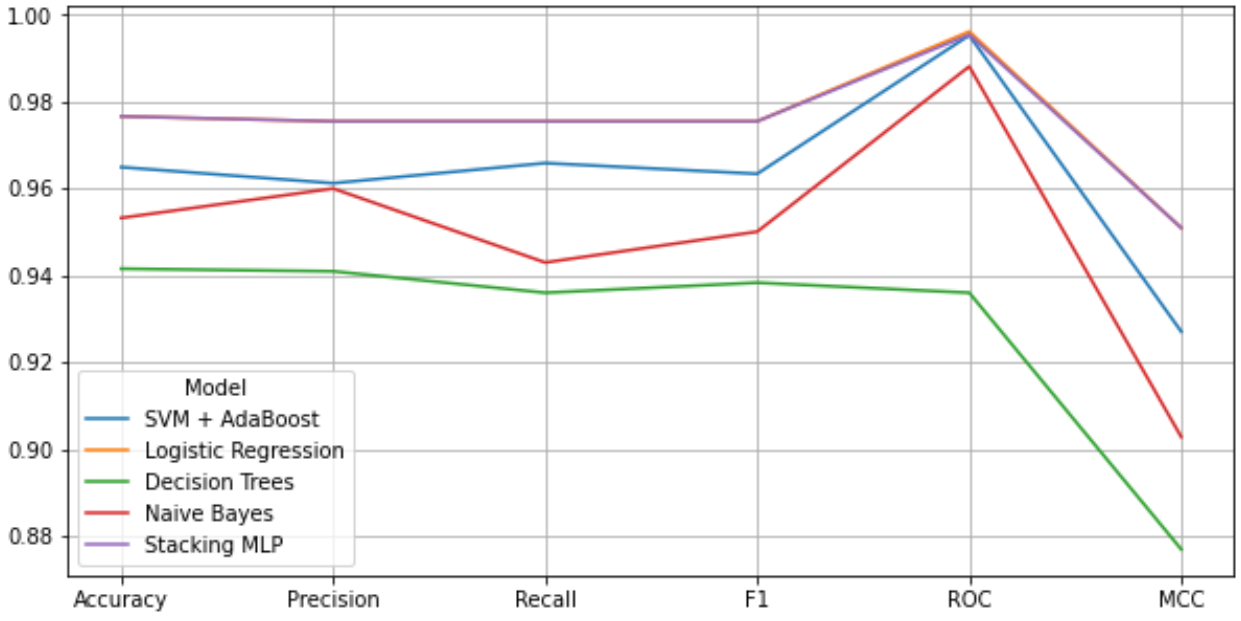


Figure 8.2. Performance of different models when fitted with test dataset

9. Conclusion

In this paper, we have provided an in-depth analysis and explanation of our approach towards the generation of various machine learning algorithms for breast cancer prediction based on diagnostic data. Based on the compiled results, we demonstrate that ensemble techniques like bagging and boosting can be utilized to improve the performance of the predictions of a model based on a particular learning algorithm. Furthermore, techniques like stacking can be used to build an ensemble model composed of multiple heterogeneous base models. We were able to maximize the prediction performance in terms of accuracy, precision, recall, F1, and ROC by building a meta-classifier using a multi-layer perceptron model that combined SVM, LR, NB, and DT models.

While most of our models yielded close and accurate results, we feel that further developments to improve the models are still necessary. Nevertheless, we hope that our models can serve as a basis for new screening methods that yield accurate results and are less expensive and invasive as compared to the current mammography tests. These models could potentially help to increase early detection and improve treatment outcomes for breast cancer patients globally.

10. References

[1] Ten top issues for women's health. (2020). Retrieved 25 April 2020, from https://www.who.int/life-course/news/commentaries/2015-intl-womens-day/en/

[2] Reese, H. (2020). The way we use mammograms is seriously flawed but AI could change that. Retrieved 25 April 2020, from https://qz.com/1367216/mammograms-are-seriously-flawed-the-way-we-use-them-now-ai-could-change-that/

[3] UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set. (2020). Retrieved 25 April 2020, from http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29

[4] Ross Quinlan (1993). C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

[5] Jesús M. Pérez, Javier Muguerza, Olatz Arbelaitz, Ibai Gurrutxaga, José I. Martín (2007). Combining multiple class

[6] Brownlee, J. (2020). How to Use Ensemble Machine Learning Algorithms in Weka. Retrieved 26 April 2020, from https://machinelearningmastery.com/use-ensemble-machine-learning-algorithms-weka/

[7] Yoav Freund, Robert E. Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.

[8] Sebastian Raschka, "StackingCVClassifier," . Available: http://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/. [Accessed 20 April 2020].

[9] Fabian Pedregosa, Ga¨el Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

Appendix

Table of data attributes

Data Column	Description
id	ID number
diagnosis	The diagnosis of breast tissues (M = malignant, B = benign)
radius_mean	mean of distances from the centre to points on the perimeter
texture_mean	The standard deviation of grey-scale values
perimeter_mean	mean size of the core tumour
area_mean	mean area of the core tumour
smoothness_mean	mean of local variation in radius lengths
compactness_mean	mean of perimeter^2 / area - 1.0
concavity_mean	mean of the severity of concave portions of the contour
concave points_mean	mean for the number of concave portions of the contour
symmetry_mean	mean of symmetry readings of the contour
fractal_dimension_mean	mean for "coastline approximation" - 1
radius_se	The standard error for the mean of distances from the centre to points on the perimeter
texture_se	The standard error for the standard deviation of grey-scale values
perimeter_se	The standard error for the size of the core tumour
area_se	The standard error for the area of the core tumour
smoothness_se	The standard error for local variation in radius lengths
compactness_se	The standard error for perimeter^2 / area - 1.0
concavity_se	The standard error for the severity of concave portions of the contour
concave points_se	The standard error for the number of concave portions of the contour
symmetry_se	The standard error of symmetry readings of the contour

fractal_dimension_se	standard error for "coastline approximation" - 1
radius_worst	"worst" or largest mean value for the mean of distances from the centre to points on the perimeter
texture_worst	"worst" or largest mean value for the standard deviation of grey-scale values
perimeter_worst	"worst" or largest mean value for the size of the core tumour
area_worst	"worst" or largest mean value for the area of the core tumour
smoothness_worst	"worst" or largest mean value for local variation in radius lengths
compactness_worst	"worst" or largest mean value for perimeter^2 / area - 1.0
concavity_worst	"worst" or largest mean value for the severity of concave portions of the contour
concave points_worst	"worst" or largest mean value for the number of concave portions of the contour
symmetry_worst	"worst" or largest mean value of symmetry readings of the contour
fractal_dimension_worst	"worst" or largest mean value for "coastline approximation" - 1

Summary tables for distribution of the Feature based attributes

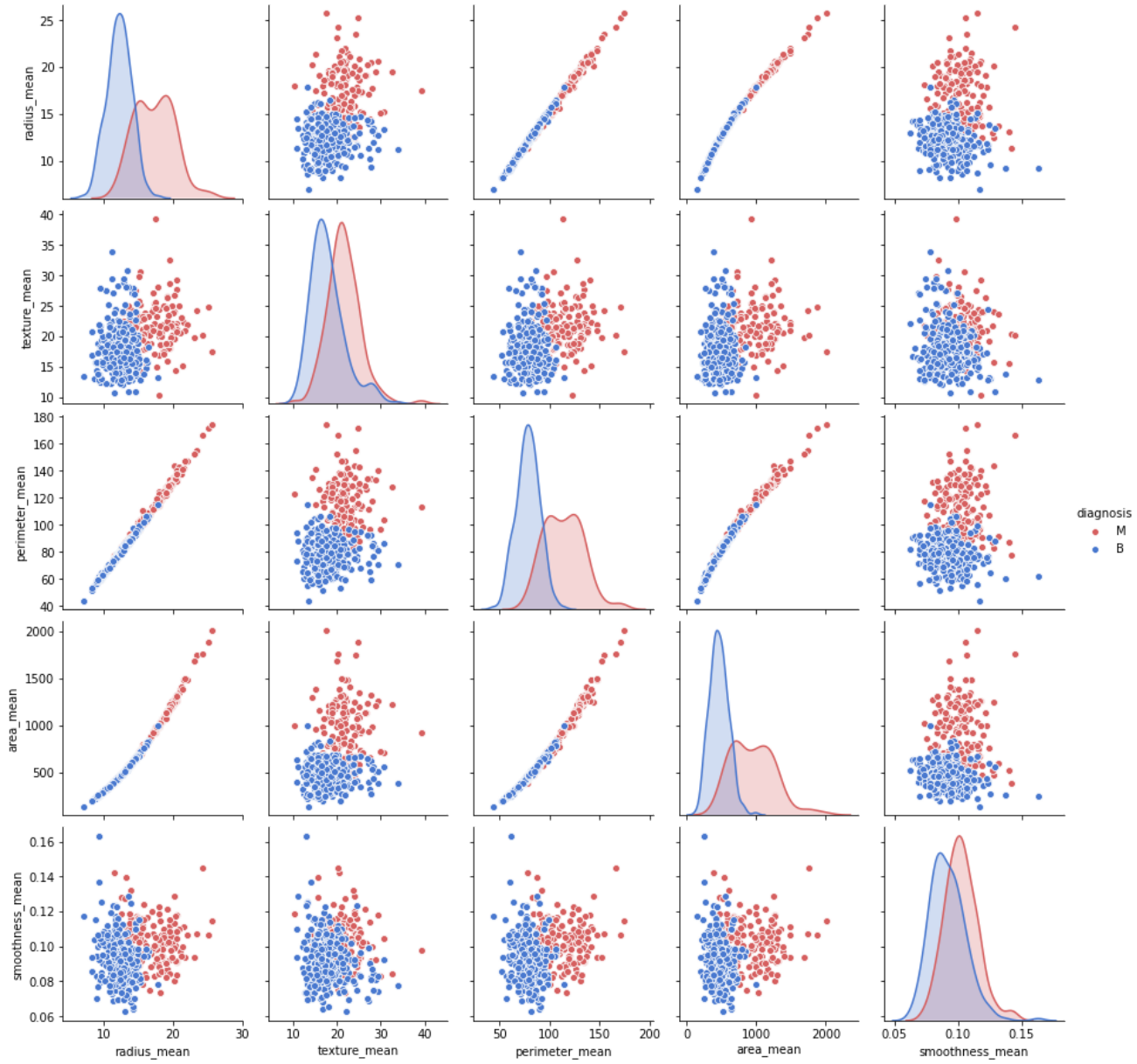
	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	'concave points_mean'	symmetry_mean	fractal_dimension_mean
mean	14.089	19.254	91.702	646.572	0.096	0.104	0.089	0.048	0.180	0.063
std	3.306	4.268	22.793	319.233	0.014	0.053	0.079	0.038	0.027	0.007
min	6.981	10.380	43.790	143.500	0.063	0.019	0.000	0.000	0.106	0.050
25%	11.718	15.990	75.605	421.100	0.085	0.067	0.030	0.021	0.160	0.058
50%	13.390	18.770	87.015	555.900	0.095	0.093	0.061	0.033	0.178	0.061
75%	15.740	21.695	103.275	771.775	0.105	0.130	0.126	0.073	0.195	0.066
max	25.730	39.280	174.200	2010.000	0.163	0.345	0.427	0.201	0.291	0.097

	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	'concave points_se'	symmetry_se	fractal_dimension_se
mean	0.393	1.209	2.793	37.960	0.007	0.025	0.033	0.012	0.020	0.004
std	0.243	0.515	1.745	33.817	0.003	0.017	0.033	0.006	0.008	0.002
min	0.112	0.362	0.757	6.802	0.003	0.002	0.000	0.000	0.008	0.001
25%	0.232	0.857	1.598	17.768	0.005	0.014	0.015	0.008	0.015	0.002
50%	0.316	1.110	2.276	24.195	0.006	0.021	0.026	0.011	0.019	0.003
75%	0.467	1.472	3.297	44.295	0.008	0.032	0.043	0.015	0.023	0.005
max	1.509	3.896	10.120	233.000	0.031	0.101	0.396	0.053	0.079	0.030

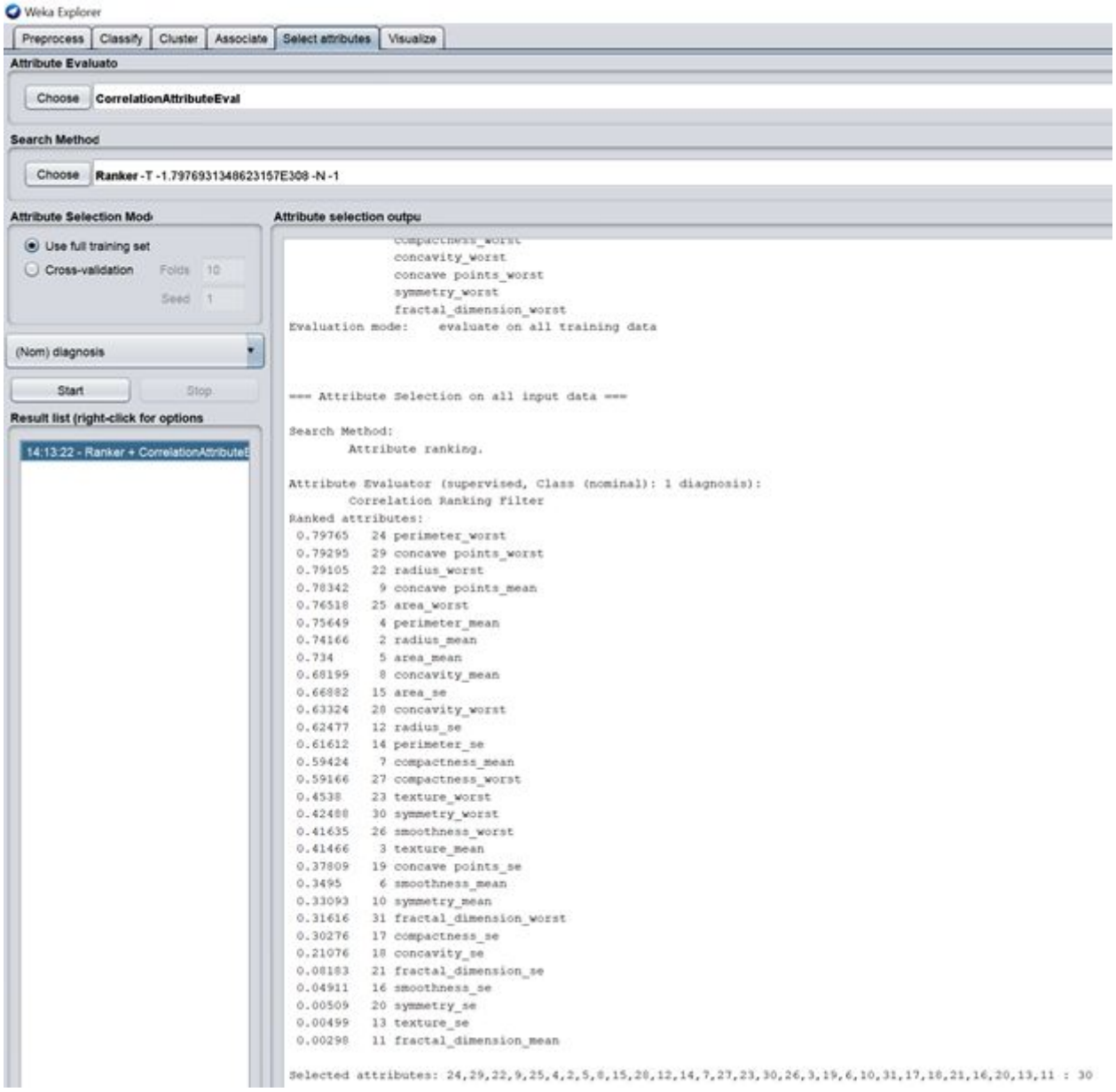
	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	'concave points_worst'	symmetry_worst	fractal_dimension_worst
mean	16.202	25.641	106.897	864.700	0.131	0.254	0.275	0.114	0.288	0.084
std	4.583	6.085	32.017	517.067	0.023	0.156	0.212	0.064	0.063	0.019
min	7.930	12.490	50.410	185.200	0.081	0.034	0.000	0.000	0.157	0.055
25%	13.070	20.890	84.543	518.700	0.114	0.145	0.114	0.065	0.248	0.071
50%	14.970	25.405	97.665	688.750	0.130	0.215	0.226	0.098	0.280	0.080
75%	18.708	29.603	125.325	1065.000	0.145	0.333	0.386	0.161	0.316	0.092
max	33.130	45.410	229.300	3234.00	0.223	1.058	1.252	0.287	0.664	0.208

Pairplot

A pairplot showing both distribution of single variables and relationships between two variables.

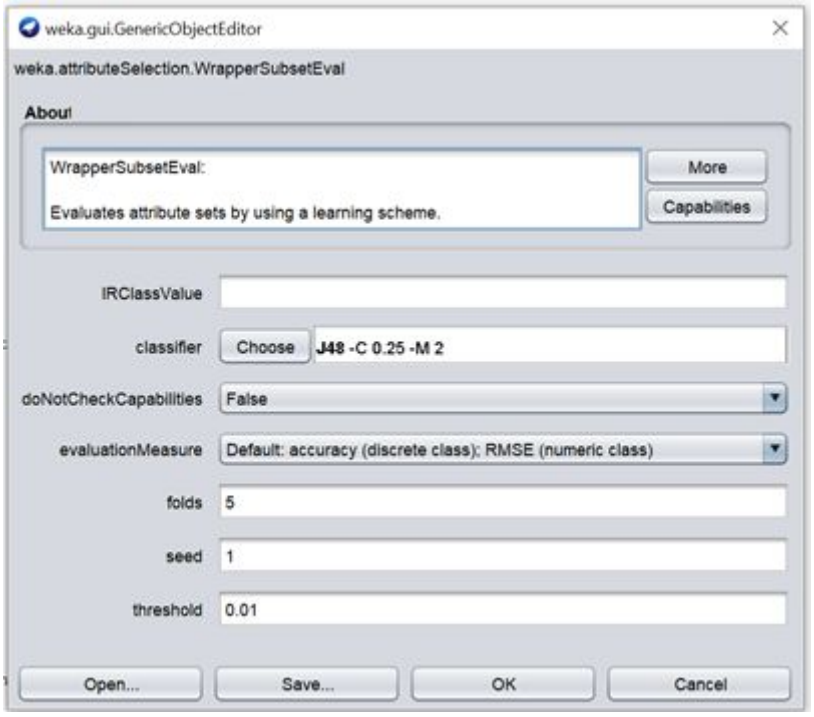


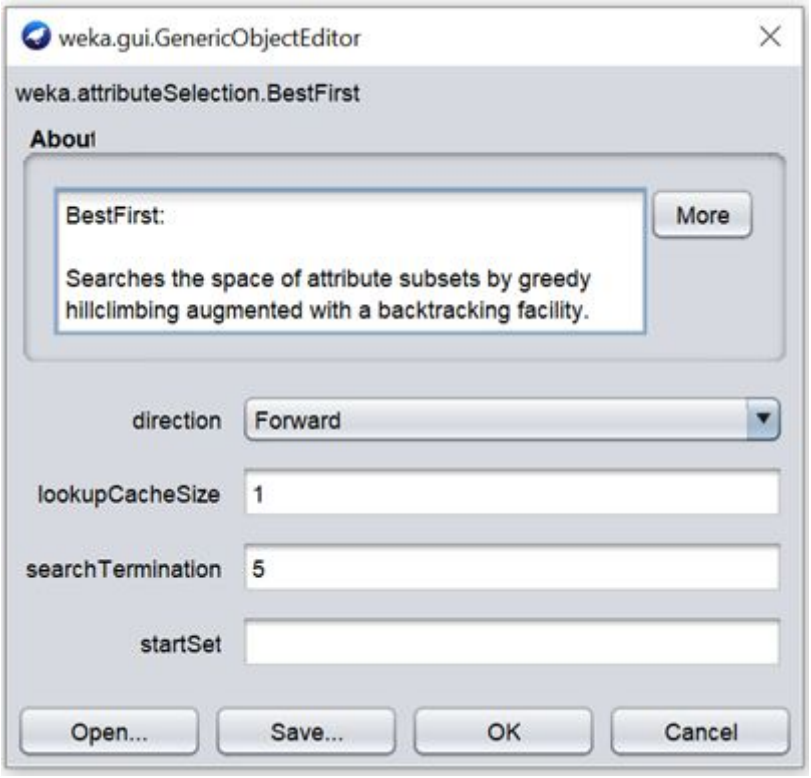
Correlation-based feature selection results

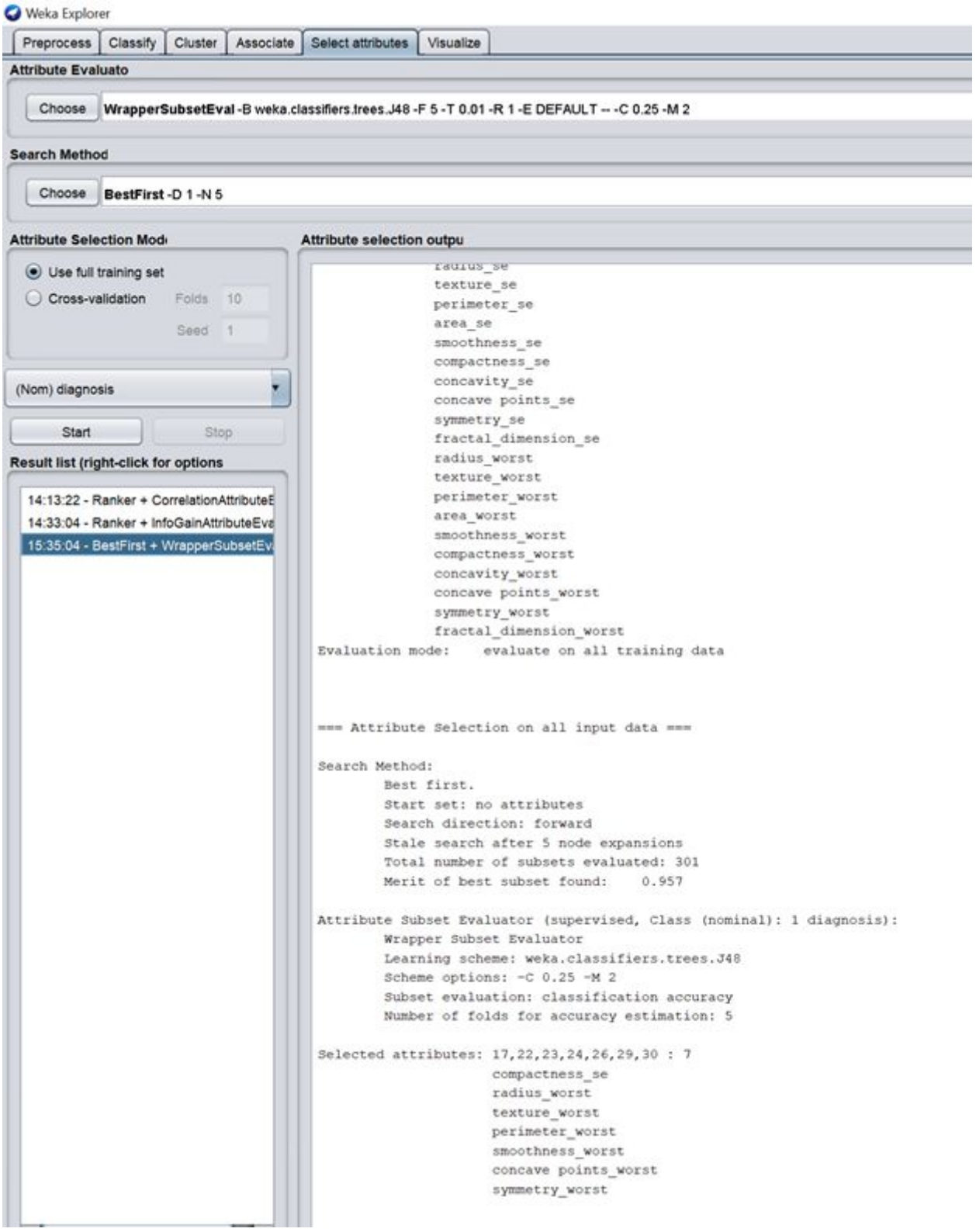


Information gain based feature selection

Learner based feature selection

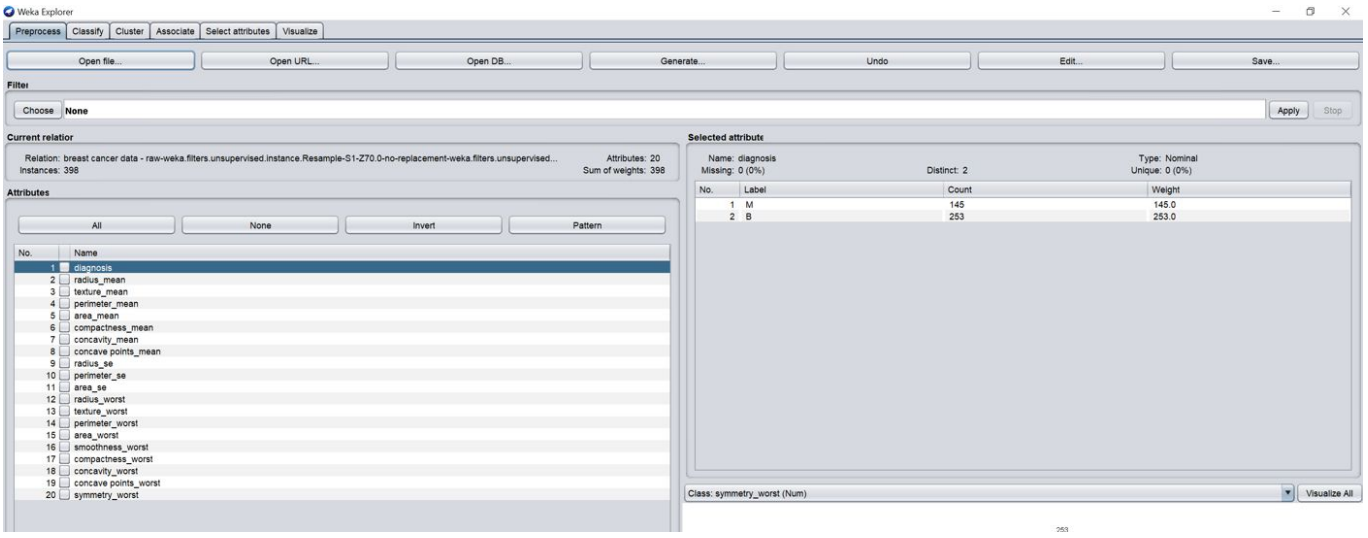


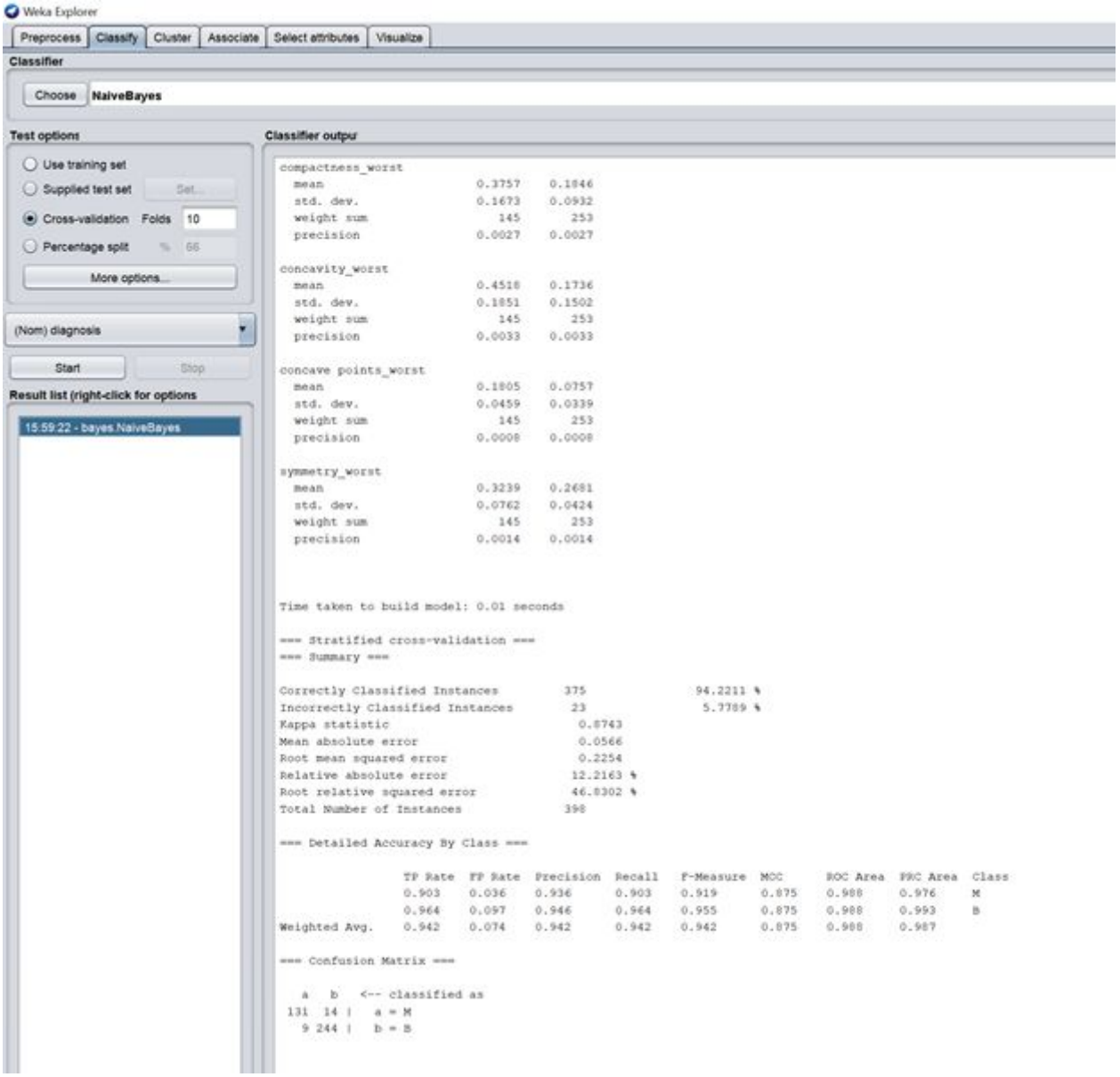



Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Attribute Evaluato
Choose
WrapperSubsetEval -B weka.classifiers.trees.J48 -F 5 -T 0.01 -R 1 -E DEFAULT -- -C 0.25 -M 2
Search Method
Choose
BestFirst -D 1 -N 5
Attribute Selection Mod
Use full training set
Cross-validation
Folds 10
Seed 1
(Nom) diagnosis
Start
Stop
Result list (right-click for options
14:13:22 - Ranker + CorrelationAttributeE
14:33:04 - Ranker + InfoGainAttributeEva
15:35:04 - BestFirst + WrapperSubsetEva
Attribute selection outpu
radius_se
texture_se
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
concave points_se
symmetry_se
fractal_dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst
compactness_worst
concavity_worst
concave points_worst
symmetry_worst
fractal_dimension_worst
Evaluation mode: evaluate on all training data
=== Attribute Selection on all input data ===
Search Method:
Best first.
Start set: no attributes
Search direction: forward
Stale search after 5 node expansions
Total number of subsets evaluated: 301
Merit of best subset found: 0.957
Attribute Subset Evaluator (supervised, Class (nominal): 1 diagnosis):
Wrapper Subset Evaluator
Learning scheme: weka.classifiers.trees.J48
Scheme options: -C 0.25 -M 2
Subset evaluation: classification accuracy
Number of folds for accuracy estimation: 5
Selected attributes: 17,22,23,24,26,29,30 : 7
compactness_se
radius_worst
texture_worst
perimeter_worst
smoothness_worst
concave points_worst
symmetry_worst

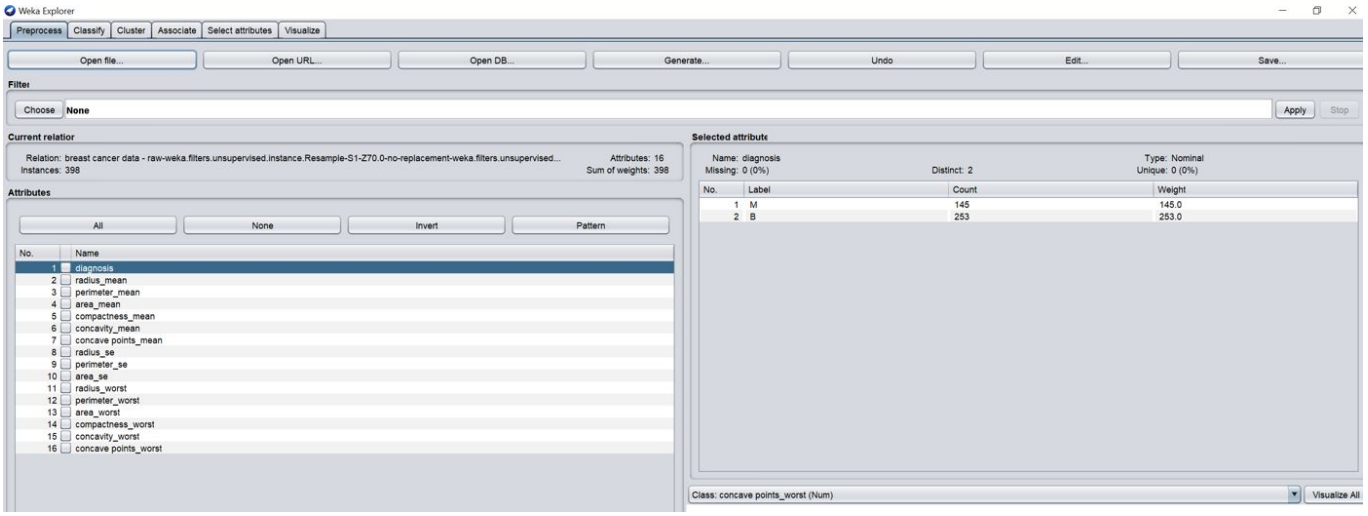
Naive Bayes Theorem attributes used and results

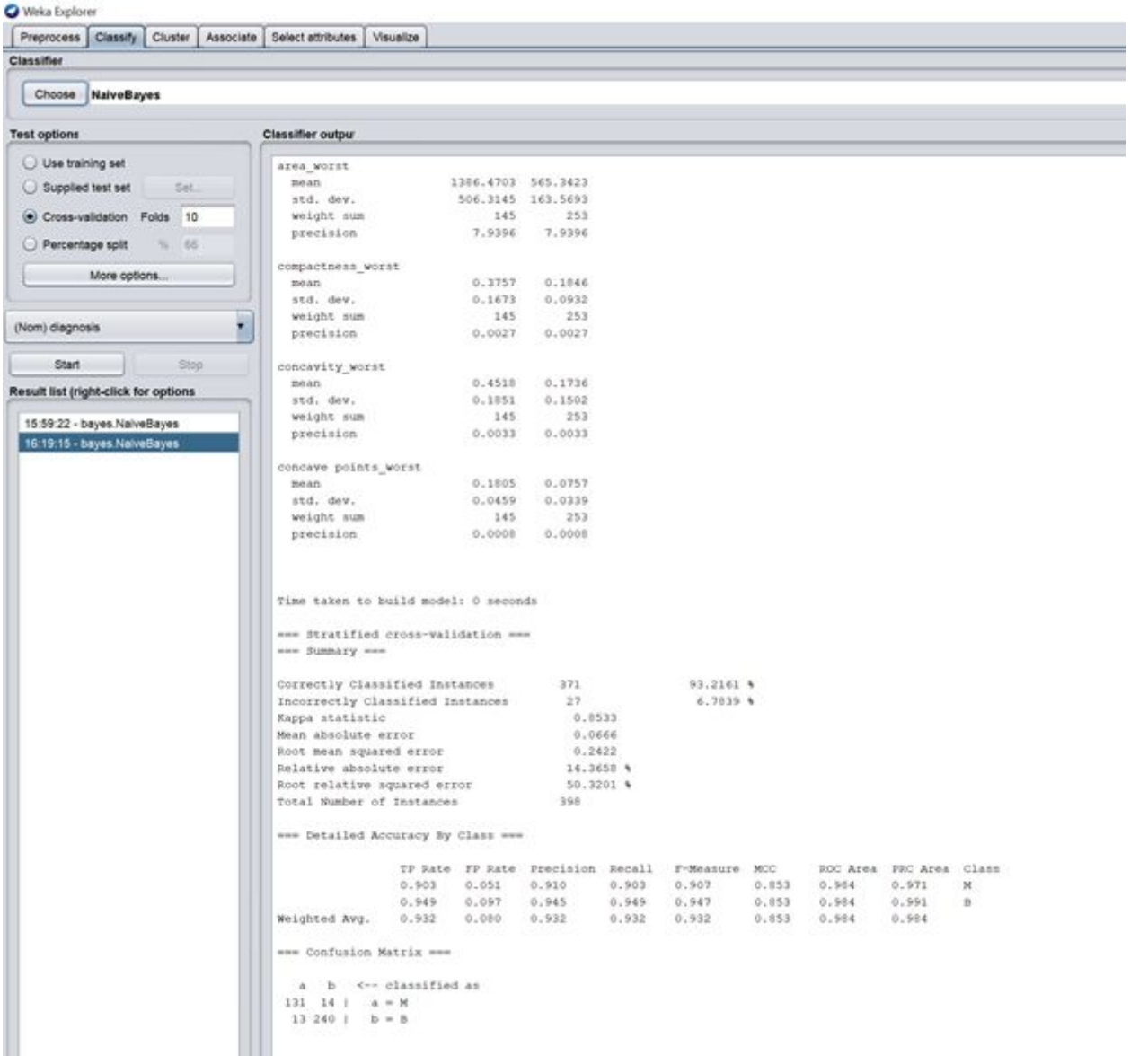
NBT-01



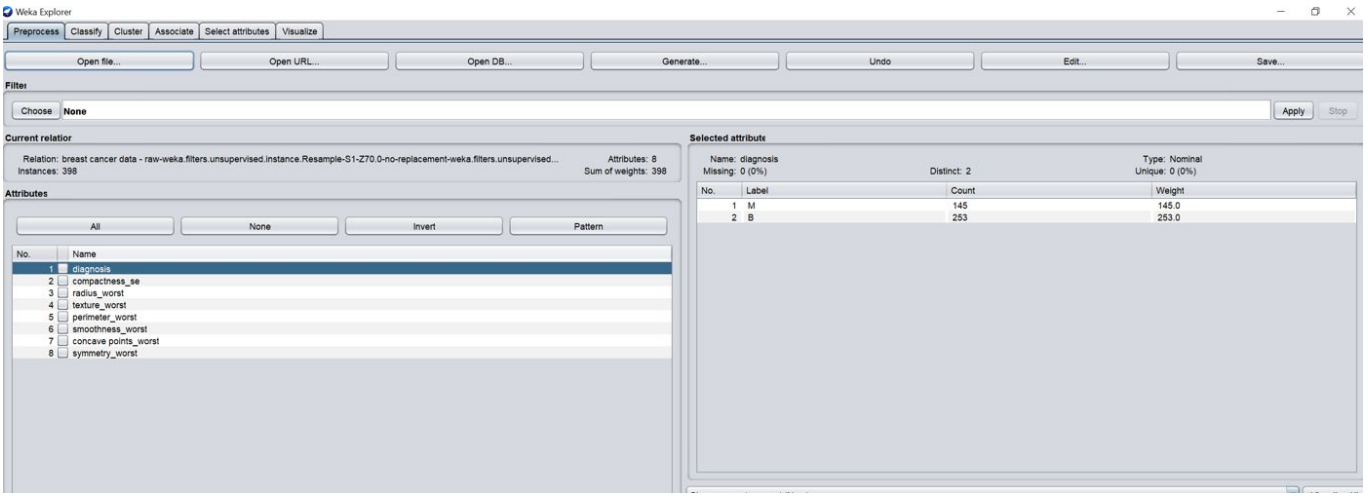

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) diagnosis
Start
Stop
Result list (right-click for options
15:59:22 - bayes.NaiveBayes
Classifier output
compactness_worst
mean 0.3757 0.1846
std. dev. 0.1673 0.0932
weight sum 145 253
precision 0.0027 0.0027
concavity_worst
mean 0.4518 0.1736
std. dev. 0.1851 0.1502
weight sum 145 253
precision 0.0033 0.0033
concave points_worst
mean 0.1805 0.0757
std. dev. 0.0459 0.0339
weight sum 145 253
precision 0.0008 0.0008
symmetry_worst
mean 0.3239 0.2681
std. dev. 0.0762 0.0424
weight sum 145 253
precision 0.0014 0.0014
Time taken to build model: 0.01 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 375 94.2211 %
Incorrectly Classified Instances 23 5.7789 %
Kappa statistic 0.8743
Mean absolute error 0.0566
Root mean squared error 0.2254
Relative absolute error 12.2163 %
Root relative squared error 46.8302 %
Total Number of Instances 398
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.903 0.036 0.936 0.903 0.919 0.875 0.988 0.976 M
0.964 0.097 0.946 0.964 0.955 0.875 0.988 0.993 B
Weighted Avg. 0.942 0.074 0.942 0.942 0.942 0.875 0.988 0.987
=== Confusion Matrix ===
a b <-- classified as
131 14 | a = M
9 244 | b = B

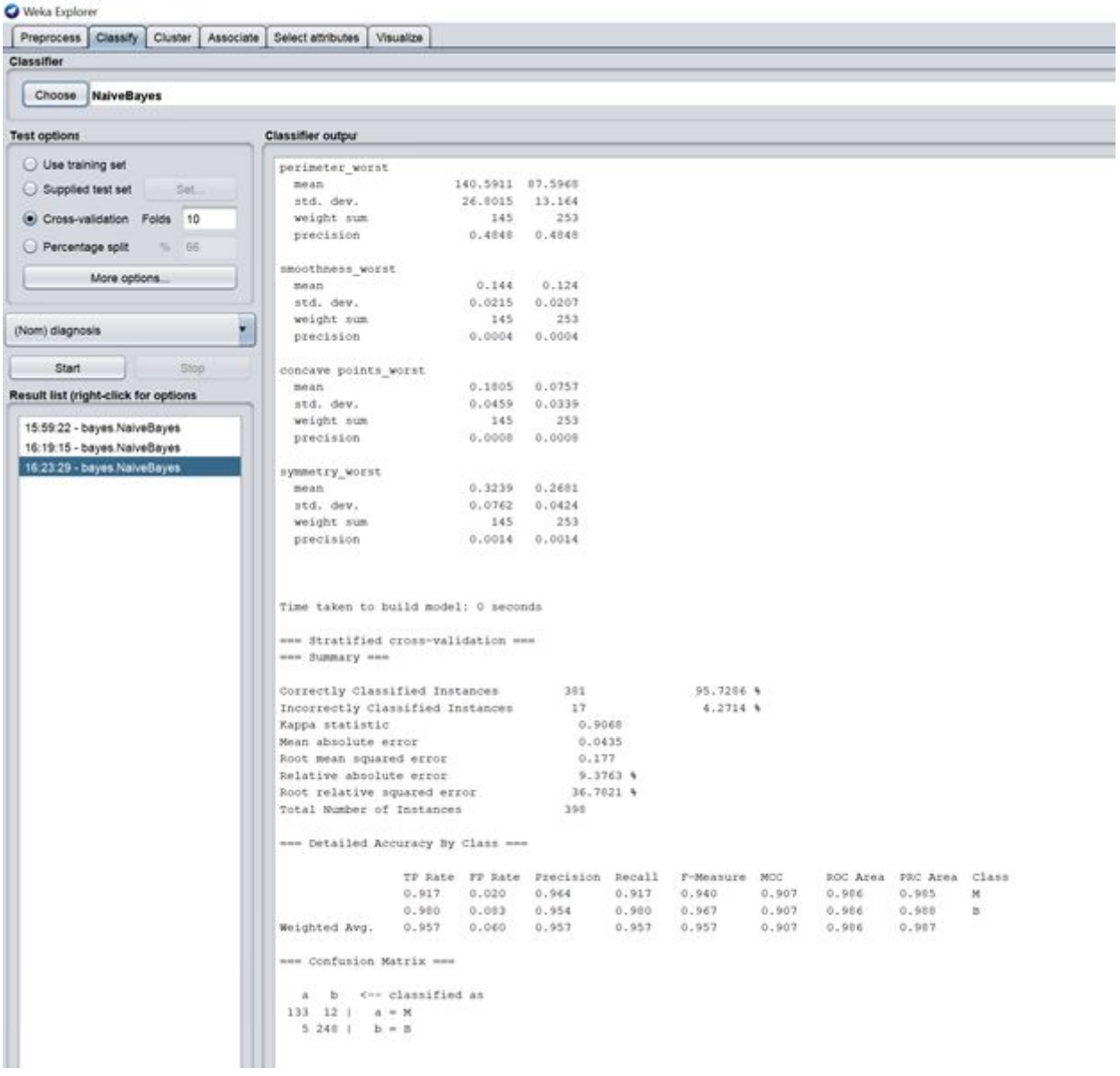
NBT-02


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Stop
Current relation
Relation: breast cancer data - raw-weka.filters.unsupervised.instance.Resample-S1-Z70.0-no-replacement-weka.filters.unsupervised...
Instances: 398
Attributes: 16
Sum of weights: 398
Attributes
All
None
Invert
Pattern
No.
Name
1 diagnosis
2 radius_mean
3 perimeter_mean
4 area_mean
5 compactness_mean
6 concavity_mean
7 concave points_mean
8 radius_se
9 perimeter_se
10 area_se
11 radius_worst
12 perimeter_worst
13 area_worst
14 compactness_worst
15 concavity_worst
16 concave points_worst
Selected attribute
Name: diagnosis
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 M 145 145.0
2 B 253 253.0
Class: concave points_worst (Num)
Visualize All

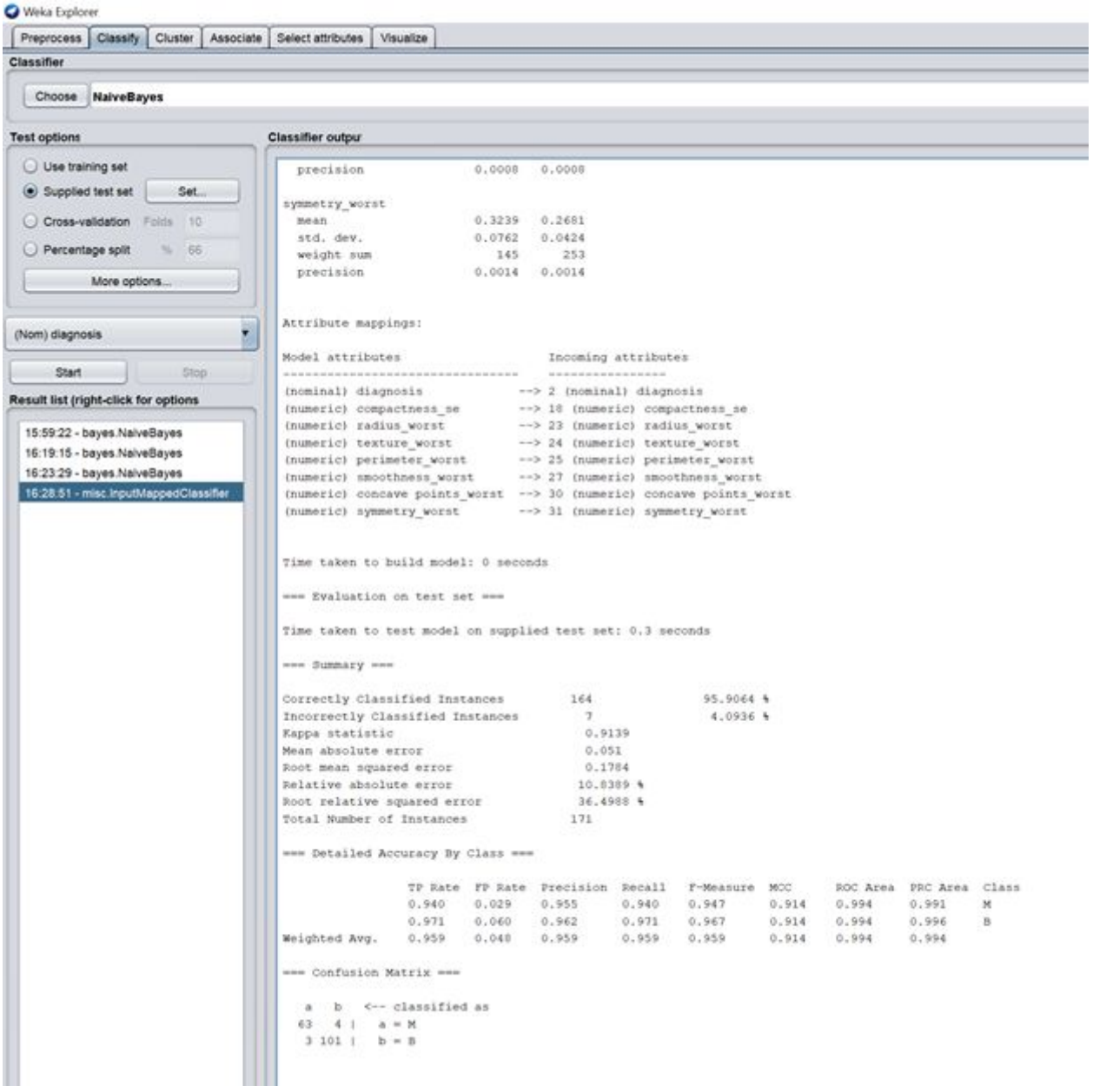

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) diagnosis
Start
Stop
Result list (right-click for options
15:59:22 - bayes.NaiveBayes
16:19:15 - bayes.NaiveBayes
Classifier output
area_worst
mean 1386.4703 565.3423
std. dev. 506.3145 163.5693
weight sum 145 253
precision 7.9396 7.9396
compactness_worst
mean 0.3757 0.1846
std. dev. 0.1673 0.0932
weight sum 145 253
precision 0.0027 0.0027
concavity_worst
mean 0.4518 0.1736
std. dev. 0.1851 0.1502
weight sum 145 253
precision 0.0033 0.0033
concave points_worst
mean 0.1805 0.0757
std. dev. 0.0459 0.0339
weight sum 145 253
precision 0.0008 0.0008
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 371 93.2161 %
Incorrectly Classified Instances 27 6.7839 %
Kappa statistic 0.8533
Mean absolute error 0.0666
Root mean squared error 0.2422
Relative absolute error 14.3658 %
Root relative squared error 50.3201 %
Total Number of Instances 398
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.903 0.051 0.910 0.903 0.907 0.853 0.984 0.971 M
0.949 0.097 0.945 0.949 0.947 0.853 0.984 0.991 B
Weighted Avg. 0.932 0.080 0.932 0.932 0.932 0.853 0.984 0.984
=== Confusion Matrix ===
a b <-- classified as
131 14 | a = M
13 240 | b = B

NBT-03


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose None
Apply
Stop
Current relation
Relation: breast cancer data - raw-weka.filters.unsupervised.instance.Resample-S1-Z70.0-no-replacement-weka.filters.unsupervised...
Instances: 398
Attributes: 8
Sum of weights: 398
Attributes
All
None
Invert
Pattern
No. Name
1 diagnosis
2 compactness_se
3 radius_worst
4 texture_worst
5 perimeter_worst
6 smoothness_worst
7 concave points_worst
8 symmetry_worst
Selected attribute
Name: diagnosis
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)
No. Label Count Weight
1 M 145 145.0
2 B 253 253.0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose NaiveBayes
Test options
Use training set
Supplied test set
Set...
Cross-validation Folds 10
Percentage split % 66
More options...
(Nom) diagnosis
Start
Stop
Result list (right-click for options
15:59:22 - bayes.NaiveBayes
16:19:15 - bayes.NaiveBayes
16:23:29 - bayes.NaiveBayes
Classifier output
perimeter_worst
mean 140.5911 87.5968
std. dev. 26.8015 13.164
weight sum 145 253
precision 0.4848 0.4848
smoothness_worst
mean 0.144 0.124
std. dev. 0.0215 0.0207
weight sum 145 253
precision 0.0004 0.0004
concave points_worst
mean 0.1805 0.0757
std. dev. 0.0459 0.0339
weight sum 145 253
precision 0.0008 0.0008
symmetry_worst
mean 0.3239 0.2681
std. dev. 0.0762 0.0424
weight sum 145 253
precision 0.0014 0.0014
Time taken to build model: 0 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 381 95.7286 %
Incorrectly Classified Instances 17 4.2714 %
Kappa statistic 0.9068
Mean absolute error 0.0435
Root mean squared error 0.177
Relative absolute error 9.3763 %
Root relative squared error 36.7621 %
Total Number of Instances 398
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure MCC ROC Area PRC Area Class
0.917 0.020 0.964 0.917 0.940 0.907 0.986 0.985 M
0.980 0.083 0.954 0.980 0.967 0.907 0.986 0.988 B
Weighted Avg. 0.957 0.060 0.957 0.957 0.957 0.907 0.986 0.987
=== Confusion Matrix ===
a b <-- classified as
133 12 | a = M
5 248 | b = B

Results of Test data on NBT-03 model



Decision Tree

Weka logs : for weka.classifiers.trees.J48 -C 0.25 -M 2

=== Run information ===

-: weka.classifiers.trees.J48 -C 0.25 -M 2
Relation: breast cancer data - 70 training
Instances: 398
Attributes: 32

Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

J48 pruned tree
---

perimeter_worst <= 105.9
| concave points_worst <= 0.1342: B (228.0/3.0)
| concave points_worst > 0.1342
| | id <= 8712853: B (10.0/1.0)
| | id > 8712853: M (5.0)
perimeter_worst > 105.9
| concave points_mean <= 0.04846
| | texture_worst <= 32.94
| | | area_worst <= 947.9: B (13.0)
| | | area_worst > 947.9
| | | | symmetry_mean <= 0.1521: B (2.0)
| | | | symmetry_mean > 0.1521: M (4.0)
| | texture_worst > 32.94: M (5.0)
| concave points_mean > 0.04846
| | perimeter_worst <= 114.3
| | | texture_worst <= 20.65: B (3.0)
| | | texture_worst > 20.65: M (16.0/1.0)
| | perimeter_worst > 114.3: M (112.0)

Number of Leaves : 10

Size of the tree : 19

Time taken to build model: 0.08 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances 373 93.7186 %

Incorrectly Classified Instances 25 6.2814 %
Kappa statistic 0.8634
K&B Relative Info Score 84.8021 %
K&B Information Score 319.3896 bits 0.8025 bits/instance
Class complexity | order 0 376.6293 bits 0.9463 bits/instance
Class complexity | scheme 14024.0456 bits 35.2363 bits/instance
Complexity improvement (Sf) -13647.4163 bits -34.29 bits/instance
Mean absolute error 0.0706
Root mean squared error 0.2443
Relative absolute error 15.2379 %
Root relative squared error 50.7539 %
Total Number of Instances 398

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.897	0.040	0.929	0.897	0.912	0.864	0.936	0.903	M
	0.960	0.103	0.942	0.960	0.951	0.864	0.936	0.927	B
Weighted Avg.	0.937	0.080	0.937	0.937	0.937	0.864	0.936	0.918	

=== Confusion Matrix ===

a	b	<-- classified as
130	15	a = M
10	243	b = B

Test

=== Run information ===

Scheme: weka.classifiers.misc.InputMappedClassifier -I -trim -W weka.classifiers.trees.J48 -- -C 0.25 -M 2
Relation: breast cancer data - 70 training
Instances: 398
Attributes: 32

=== Evaluation on test set ===

Time taken to test model on supplied test set: 0.01 seconds

=== Summary ===

Correctly Classified Instances 161 94.152 %
Incorrectly Classified Instances 10 5.848 %
Kappa statistic 0.876
K&B Relative Info Score 86.4553 %
K&B Information Score 143.1337 bits 0.837 bits/instance

Class complexity | order 0 165.558 bits 0.9682 bits/instance
Class complexity | scheme 3250.4428 bits 19.0084 bits/instance
Complexity improvement (Sf) -3084.8849 bits -18.0403 bits/instance
Mean absolute error 0.0639
Root mean squared error 0.2278
Relative absolute error 13.582 %
Root relative squared error 46.5993 %
Total Number of Instances 171

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.896	0.029	0.952	0.896	0.923	0.877	0.976	0.941	M
	0.971	0.104	0.935	0.971	0.953	0.877	0.976	0.979	B
Weighted Avg.	0.942	0.075	0.942	0.942	0.941	0.877	0.976	0.964	

=== Confusion Matrix ===

a	b	<-- classified as
60	7	a = M
3	101	b = B

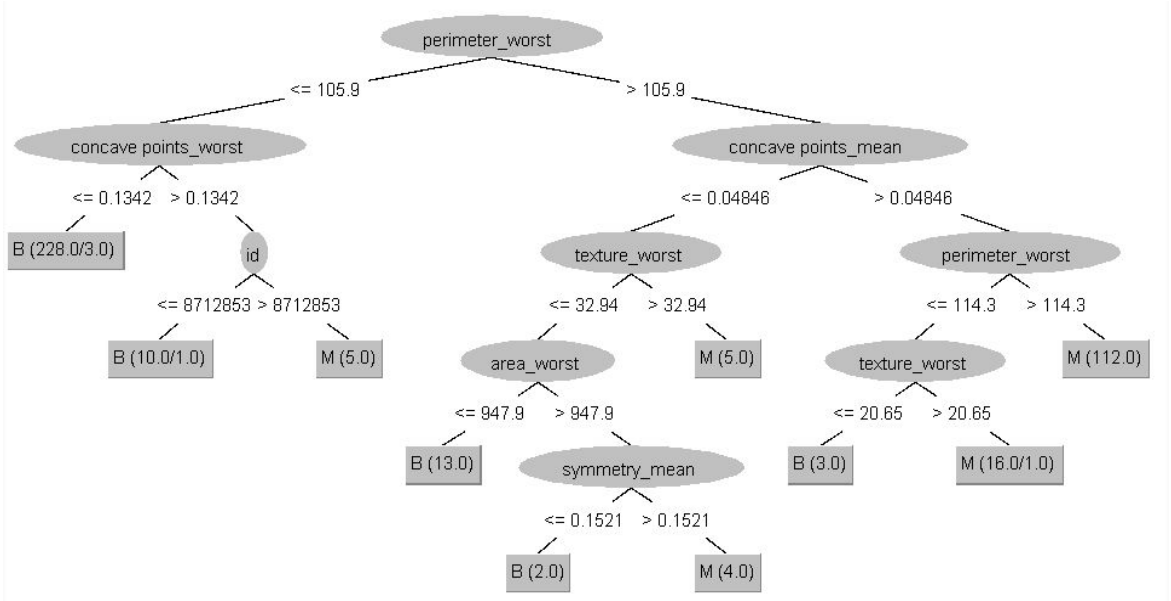


Figure 7. 2.1. Decision tree by using weka.classifiers.trees.J48

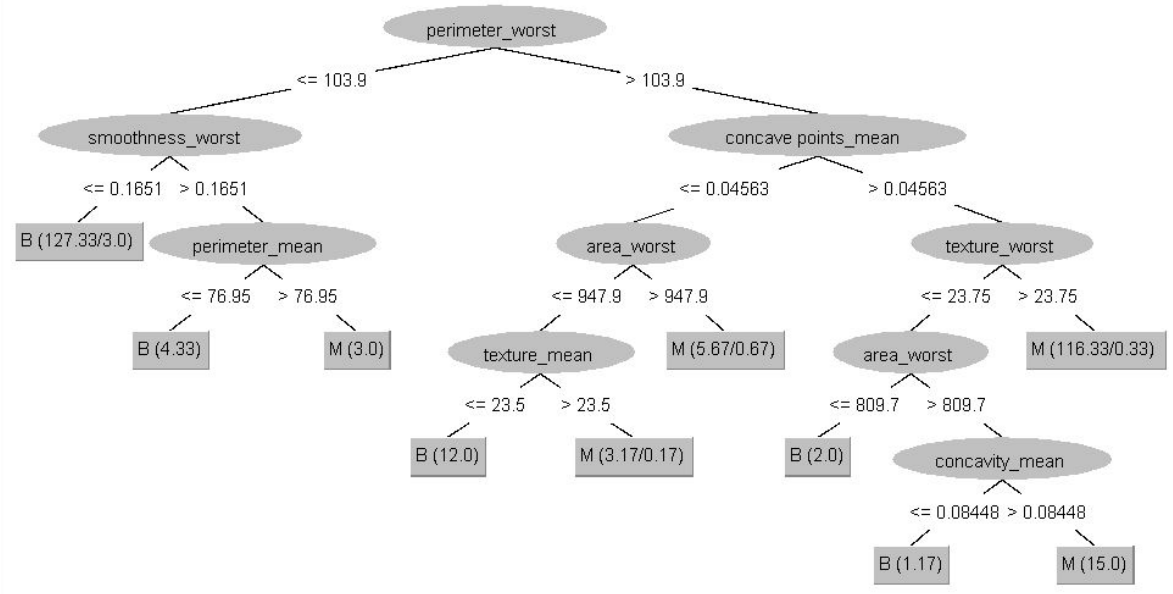


Figure 7. 2. 2. Decision tree by using weka.classifiers.trees.J48Consolidated

Model Training

• “y_train” variable represents the feature we are trying to predict (Output). In this case we are trying to predict if our “target” is cancerous (Malignant) or not (Benign). i.e. we are going to use the “target” feature here.

• “X_train” variable represents the predictors which are the remaining columns (mean radius, mean texture, mean perimeter, mean area, mean smoothness, etc.)

```
In [286]: X_train = df_cancer_train.drop(['diagnosis', 'id'], axis = 1)
          X_train.head()
```

Out[286]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	'concave points_mean'	symmetry_mean	fractal
0	18.82	21.97	123.70	1110.0	0.10180	0.13890	0.15940	0.08744	0.1943	
1	17.29	22.13	114.40	947.8	0.08999	0.12730	0.09697	0.07507	0.2108	
2	14.50	10.89	94.28	640.7	0.11010	0.10990	0.08842	0.05778	0.1856	
3	17.27	25.42	112.40	928.8	0.08331	0.11090	0.12040	0.05736	0.1467	
4	12.67	17.30	81.25	489.9	0.10280	0.07664	0.03193	0.02107	0.1707	

5 rows × 30 columns

```
In [287]: y_train = df_cancer_train['diagnosis']
          y_train.head()

Out[287]: 0    M
          1    M
          2    B
          3    M
          4    B
          Name: diagnosis, dtype: object
```

```
In [60]: param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf', 'linear']}
         grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=1)
         grid.fit(X_train_scaled,y_train)

         Fitting 5 folds for each of 32 candidates, totalling 160 fits

         [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
         [Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed:    1.2s finished

Out[60]: GridSearchCV(cv=None, error_score=nan,
                      estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                                    class_weight=None, coef0=0.0,
                                    decision_function_shape='ovr', degree=3,
                                    gamma='scale', kernel='rbf', max_iter=-1,
                                    probability=False, random_state=None, shrinking=True,
                                    tol=0.001, verbose=False),
                      iid='deprecated', n_jobs=None,
                      param_grid={'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001],
                                  'kernel': ['rbf', 'linear']},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                      scoring=None, verbose=1)

In [61]: print (grid.best_params_)
         print ('\n')
         print (grid.best_estimator_)

         {'C': 10, 'gamma': 1, 'kernel': 'rbf'}


         SVC(C=10, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
             probability=False, random_state=None, shrinking=True, tol=0.001,
             verbose=False)
```

Logistic Regression - Attributes Selected and their coefficients for each model.

Attributes	LR without Regularisation	LR with Regularisation	LR with FS Filter	LR with FS Backward Elimination	LR with FS Recursive Feature Elimination
radius_mean	5.61	0.00	-18.22	-0.157498	-1.67
texture_mean	2.61	0.00	0.00	-0.137060	-0.03
perimeter_mean	5.31	0.00	-5.40	0.000000	-0.18
area_mean	9.62	0.00	-8.04	0.000000	0.02
smoothness_mean	-3.04	0.00	0.00	0.000000	0.00
compactness_mean	-13.46	0.00	-19.83	0.000366	0.00
concavity_mean	4.29	0.00	7.44	0.000000	0.42
concave points_mean	15.38	1.64	23.49	0.000812	0.21
symmetry_mean	-2.47	0.00	0.00	0.000000	0.00
fractal_dimension_mean	-9.23	0.00	0.00	0.000000	0.02
radius_se	6.07	0.00	-20.76	0.000000	-0.08
texture_se	-9.26	0.00	0.00	0.000000	-0.58
perimeter_se	6.19	0.00	-4.42	0.000000	-0.26
area_se	14.82	0.00	49.09	0.000000	0.05
smoothness_se	9.36	0.00	0.00	-0.000104	0.00
compactness_se	-5.77	0.00	0.00	0.000000	0.05
concavity_se	2.50	0.00	0.00	0.000000	0.00
concave points_se	-4.24	0.00	0.00	-0.000063	0.00
symmetry_se	0.20	0.00	0.00	0.000000	0.03
fractal_dimension_se	-7.30	0.00	0.00	-0.000032	0.01
radius_worst	12.49	10.66	19.28	-0.141876	-1.71
texture_worst	24.59	3.67	0.00	0.000000	0.28
perimeter_worst	8.70	0.00	15.88	0.000000	0.29
area_worst	15.85	0.00	28.20	0.008278	0.02

smoothness_worst	23.18	0.00	0.00	0.000000	0.18
compactness_worst	-1.24	0.00	11.90	0.000000	0.77
concavity_worst	12.03	0.00	0.08	0.000000	1.06
concave points_worst	13.97	6.89	2.88	0.000000	0.37
symmetry_worst	18.86	0.05	0.00	-0.002883	0.38
fractal_dimension_worst	3.77	0.00	0.00	-0.000756	0.00

Logistics Regression Result Summary

		LR without Regularisation	LR with Regularisation	LR with FS Filter	LR with FS Backward Elimination	LR with FS Recursive Feature Elimination
Ksplit	**Test Accuracy**	0.975	0.958	0.95	0.925	0.95
	Precision	0.98	0.96	0.95	0.93	0.95
	Recall	0.97	0.95	0.95	0.91	0.94
	F-score	0.97	0.95	0.95	0.92	0.95
K-Fold	**Mean**					
	Test Accuracy	0.96	0.97	0.95	0.882	0.92
	Precision	0.959	0.972	0.948	0.882	0.918
	Recall	0.955	0.962	0.942	0.86	0.907
	F-score	0.956	0.966	0.944	0.864	0.908
	ROC-Area	0.986	0.989	0.988	0.942	0.967
	Stdev					
	Test Accuracy	0.033	0.027	0.039	0.05	0.048
	Precision	0.036	0.032	0.04	0.052	0.052
	Recall	0.033	0.032	0.04	0.056	0.054
	F-score	0.033	0.032	0.04	0.057	0.057
	ROC-Area	0.019	0.032	0.013	0.034	0.033
Kfold Bagging	**Mean**					
	Test Accuracy	0.975	0.965	0.947	0.885	0.92
	Precision	0.975	0.023	0.943	0.887	0.918
	Recall	0.97	0.968	0.942	0.862	0.907
	F-score	0.972	0.953	0.942	0.867	0.908

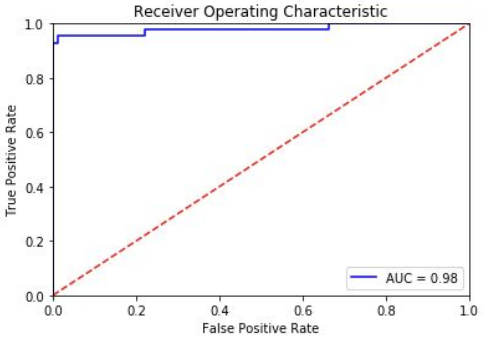
	ROC-Area	0.991	0.989	0.99	0.943	0.966
	Stdev					
	Test Accuracy	0.027	0.023	0.04	0.047	0.05
	Precision	0.032	0.03	0.045	0.051	0.055
	Recall	0.029	0.026	0.043	0.054	0.055
	F-score	0.019	0.027	0.044	0.057	0.057
	ROC-Area	0.019	0.015	0.011	0.034	0.033

Program Output

1. Logistic Regression with no regularisation and no feature selection.

    a. Summary

```
Logistic Regression with no regularisation and no feature selection
Results

 Feature Attributes:
 Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
        'smoothness_mean', 'compactness_mean', 'concavity_mean',
        ''concave points_mean'', 'symmetry_mean', 'fractal_dimension_mean',
        'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
        'compactness_se', 'concavity_se', ''concave points_se'', 'symmetry_se',
        'fractal_dimension_se', 'radius_worst', 'texture_worst',
        'perimeter_worst', 'area_worst', 'smoothness_worst',
        'compactness_worst', 'concavity_worst', ''concave points_worst'',
        'symmetry_worst', 'fractal_dimension_worst'],
       dtype='object')

 Coefficients of Feature Attributes:
 [[  5.62316916   2.57458685   5.3147093    9.60937023  -3.00105037
  -13.36664615   4.32626833  15.29171231  -2.43746772  -9.25500547
    6.07053965  -9.25681879   6.15522481  14.77794392   9.35055645
   -5.76528731   2.47390671  -4.2423435    0.21150906  -7.27206569
   12.44439283  24.56426134   8.67437464  15.78212932  23.10077009
   -1.22602062  12.0024507   13.92844816  18.79716344   3.74860572]]

 test accuracy: 0.975
train accuracy: 1.0

 Classification Report
              precision    recall  f1-score   support

           0       0.97      0.99      0.98        77
           1       0.98      0.95      0.96        43

    accuracy                           0.97       120
   macro avg       0.98      0.97      0.97       120
weighted avg       0.98      0.97      0.97       120


 Confusion Matrix
[[76  1]
 [ 2 41]]
```



b. Implementation

```
data=pd.read_csv("breast-cancer-data-training.csv")
data.drop(["id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values ## normalisation

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42) ## datasplit

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T
```

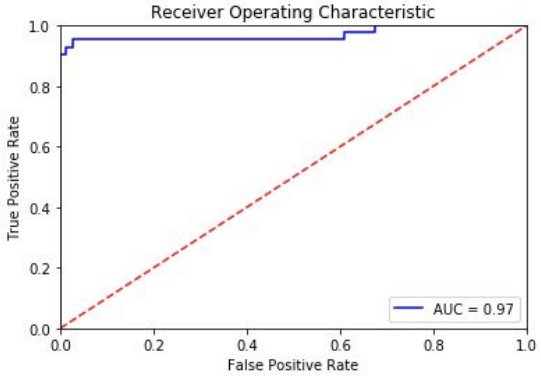
```
print(color.BOLD + 'Logistic Regression with no regularisation and no feature selection' + color.END)
print(color.BOLD + 'Results' + color.END)
logreg = linear_model.LogisticRegression(penalty='none',solver='saga',max_iter=20000)
logreg.fit(x_train.T, y_train.T)
theta=logreg.coef_
print("\n Feature Attributes:\n",x.columns)
print("\n Coefficients of Feature Attributes:\n",theta)

print("\n test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_test.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T, y_train.T)))
y_pred=logreg.predict(x_test.T)

print("\n Classification Report")
print(classification_report(y_test.T,y_pred))

#Graph Plot
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logreg.predict_proba(x_test.T)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test.T, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

#Confusion Matrix
print("\n Confusion Matrix")
cm = confusion_matrix(y_test.T,y_pred)
print(cm)

```

2. Logistic Regression with L1 Regularisation

a. Summary

Logistic Regression with L1 regularisation
Results

```
Feature Attributes:
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       ''concave points_mean'', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', ''concave points_se'', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', ''concave points_worst'',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')

Coefficients of Feature Attributes:
[[ 0.          0.          0.          0.          0.          0.
   0.          0.38130854  0.          0.          0.          0.
   0.          0.          0.          0.          0.          0.
   0.          0.         12.59305133  4.56627023  0.          0.
   0.          0.          0.          8.64764154  0.35236895  0.        ]]

test accuracy: 0.9583333333333334
train accuracy: 0.9748201438848921

Classification Report
              precision    recall  f1-score   support

           0       0.95      0.99      0.97        77
           1       0.97      0.91      0.94        43

    accuracy                           0.96       120
   macro avg       0.96      0.95      0.95       120
weighted avg       0.96      0.96      0.96       120


Confusion Matrix
[[76  1]
 [ 4 39]]
```



b. Implementation

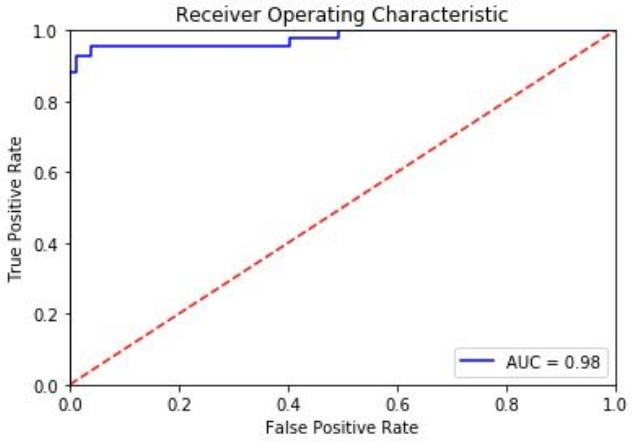
```
data=pd.read_csv("breast-cancer-data-training.csv")
data.drop(["id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values ## normalisation

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42) ## datasplit

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T
```

```
print(color.BOLD + 'Logistic Regression with L1 regularisation' + color.END)
print(color.BOLD + 'Results' + color.END)
logreg = linear_model.LogisticRegression(penalty='l1',solver='saga',max_iter=20000)
logreg.fit(x_train.T, y_train.T)
theta=logreg.coef_
print("\n Feature Attributes:\n",x.columns)
print("\n Coefficients of Feature Attributes:\n",theta)

print("\n test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_test.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T, y_train.T)))
y_pred=logreg.predict(x_test.T)

print("\n Classification Report")
print(classification_report(y_test.T,y_pred))

#Graph Plot
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logreg.predict_proba(x_test.T)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test.T, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

#Confusion Matrix
print("\n Confusion Matrix")
cm = confusion_matrix(y_test.T,y_pred)
print(cm)
```

3. Logistic Regression with Feature Selection Filter

a. Summary

```
Logistic Regression with FS Filter
Results

 Feature Attributes:
 Index(['radius_mean', 'perimeter_mean', 'area_mean', 'compactness_mean',
        'concavity_mean', ''concave points_mean'', 'radius_se', 'perimeter_se',
        'area_se', 'radius_worst', 'perimeter_worst', 'area_worst',
        'compactness_worst', 'concavity_worst', ''concave points_worst''],
       dtype='object')

 Coefficients of Feature Attributes:
 [[-18.21772245  -5.39460557  -8.04452338 -19.83112483   7.44445552
   23.49163441 -20.76736144  -4.41698495  49.0872075   19.28450526
   15.86556183  28.21506763  11.90091577   0.0758251    2.88015865]]

 test accuracy: 0.95
train accuracy: 0.9676258992805755

 Classification Report
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        77
           1       0.93      0.93      0.93        43

    accuracy                           0.95       120
   macro avg       0.95      0.95      0.95       120
weighted avg       0.95      0.95      0.95       120


 Confusion Matrix
[[74  3]
 [ 3 40]]
```



b. Implementation

In [13]:
```
data=pd.read_csv("breast-cancer-data-training.csv")
data.drop(["id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]

plt.figure(figsize=(20,20))
cor = data.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()

cor_target = abs(cor["diagnosis"])
relevant_features = cor_target[cor_target>0.5]
print(relevant_features)
irrelevant_features = cor_target[cor_target<=0.5]
print(irrelevant_features)
```

In [14]:
```
data.drop(irrelevant_features.index, axis=1, inplace=True)
y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values


x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)
```

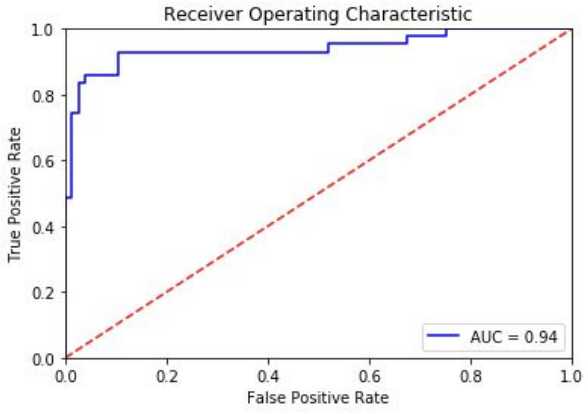
click to scroll output; double click to h

In [15]:
```
print(color.BOLD + 'Logistic Regression with FS Filter' + color.END)
print(color.BOLD + 'Results' + color.END)
logreg = linear_model.LogisticRegression(penalty='none',solver='saga',max_iter=20000)
logreg.fit(x_train.T, y_train.T)
theta=logreg.coef_
print("\n Feature Attributes:\n",x.columns)
print("\n Coefficients of Feature Attributes:\n",theta)

print("\n test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_test.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T, y_train.T)))
y_pred=logreg.predict(x_test.T)

print("\n Classification Report")
print(classification_report(y_test.T,y_pred))

#Graph Plot
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logreg.predict_proba(x_test.T)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test.T, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

#Confusion Matrix
print("\n Confusion Matrix")
cm = confusion_matrix(y_test.T,y_pred)
print(cm)

```

4. Logistic Regression with Feature Selection - Backward Elimination

a. Summary

```
Logistic Regression with FS Wrapper - Backwardward Elimination
Results

 Feature Attributes:
 Index(['radius_mean', 'texture_mean', 'compactness_mean',
       ''concave points_mean'', 'smoothness_se', ''concave points_se'',
       'fractal_dimension_se', 'radius_worst', 'area_worst', 'symmetry_worst',
       'fractal_dimension_worst'],
      dtype='object')

 Coefficients of Feature Attributes:
 [[-1.57493140e-01 -1.37063492e-01  3.66288035e-04  8.11755696e-04
  -1.04392657e-04 -6.33626480e-05 -3.17567968e-05 -1.41876383e-01
   8.27807544e-03 -2.88258291e-03 -7.55949584e-04]]

 test accuracy: 0.925
train accuracy: 0.8633093525179856

 Classification Report
              precision    recall  f1-score   support

           0       0.91      0.97      0.94        77
           1       0.95      0.84      0.89        43

    accuracy                           0.93       120
   macro avg       0.93      0.91      0.92       120
weighted avg       0.93      0.93      0.92       120


 Confusion Matrix
[[75  2]
 [ 7 36]]
```



b. Implementation

```
#Backward Elimination
data=pd.read_csv("breast-cancer-data-training.csv")
data.drop(["id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]

y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values

cols = list(x.columns)
pmax = 1
while (len(cols)>0):
    p= []
    X_1 = x[cols]
    X_1 = sm.add_constant(X_1)
    model = sm.OLS(y,X_1).fit()
    p = pd.Series(model.pvalues.values[1:],index = cols)
    pmax = max(p)
    feature_with_p_max = p.idxmax()
    if(pmax>0.05):
        cols.remove(feature_with_p_max)
    else:
        break
selected_features_BE = cols
print(selected_features_BE)

data2=data[selected_features_BE]
x = data2

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)

```

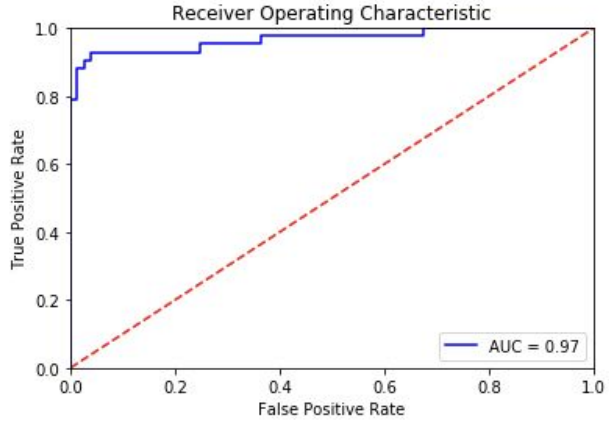
```
print(color.BOLD + 'Logistic Regression with FS Wrapper - Backwardward Elimination' + color.END)
print(color.BOLD + 'Results' + color.END)
logreg = linear_model.LogisticRegression(penalty='none',solver='saga',max_iter=20000)
logreg.fit(x_train.T, y_train.T)
theta=logreg.coef_
print("\n Feature Attributes:\n",x.columns)
print("\n Coefficients of Feature Attributes:\n",theta)

print("\n test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_test.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T, y_train.T)))
y_pred=logreg.predict(x_test.T)

print("\n Classification Report")
print(classification_report(y_test.T,y_pred))

#Graph Plot
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logreg.predict_proba(x_test.T)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test.T, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

#Confusion Matrix
print("\n Confusion Matrix")
cm = confusion_matrix(y_test.T,y_pred)
print(cm)

```

5. Logistic Regression with Feature Selection - Recursive Feature Elimination

a. Summary

Logistic Regression with FS Wrapper Recursive Feature Elimination Results

```
 Feature Attributes:
 Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'concavity_mean', ''concave points_mean'', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'compactness_se',
       'symmetry_se', 'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', ''concave points_worst'',
       'symmetry_worst'],
      dtype='object')

 Coefficients of Feature Attributes:
 [[-1.69452435e-02  2.49001768e-03 -8.26655719e-02 -1.12723335e-02
   8.42705541e-04  3.36188911e-04 -9.90709515e-05 -4.42148758e-04
  -1.08703044e-03  3.71903758e-04  5.63763796e-03  1.34674976e-04
  -1.34606953e-05  9.54002264e-06 -1.82981425e-02  7.54019376e-03
  -6.43879995e-02  2.63842436e-02 -8.88962347e-05  1.71193278e-03
   2.35246981e-03  5.59600677e-04 -1.20245333e-04]]

 test accuracy: 0.95
train accuracy: 0.9136690647482014

 Classification Report
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        77
           1       0.95      0.91      0.93        43

    accuracy                           0.95       120
   macro avg       0.95      0.94      0.95       120
weighted avg       0.95      0.95      0.95       120


 Confusion Matrix
[[75  2]
 [ 4 39]]
```



b. Implementation

In [24]:
```
data=pd.read_csv("breast-cancer-data-training.csv")
data.drop(["id"], axis=1, inplace=True)
data.diagnosis = [1 if each == "M" else 0 for each in data.diagnosis]

y = data.diagnosis.values
x_data = data.drop(['diagnosis'], axis=1)
X = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values

from sklearn.feature_selection import RFE
model = LogisticRegression(solver="liblinear")
#Initializing RFE model
rfe = RFE(model, 16)
#Transforming data using RFE
X_rfe = rfe.fit_transform(X,y)
#Fitting the data to model
model.fit(X_rfe,y)
print(rfe.support_)
print(rfe.ranking_)



```

In [25]:
```
nof_list=np.arange(1,32)
high_score=0
#Variable to store the optimum features
nof=0
score_list =[]
for n in range(len(nof_list)):
    X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, random_state = 0)
    model = LogisticRegression(solver="liblinear")
    rfe = RFE(model,nof_list[n])
    X_train_rfe = rfe.fit_transform(X_train,y_train)
    X_test_rfe = rfe.transform(X_test)
    model.fit(X_train_rfe,y_train)
    score = model.score(X_test_rfe,y_test)
    score_list.append(score)
    if(score>high_score):
        high_score = score
        nof = nof_list[n]
print("Optimum number of features: %d" %nof)
print("Score with %d features: %f" % (nof, high_score))

```

In [26]:
```
cols = list(X.columns)
model = LogisticRegression(solver="liblinear")
#Initializing RFE model
rfe = RFE(model, 23)
#Transforming data using RFE
X_rfe = rfe.fit_transform(X,y)
#Fitting the data to model
model.fit(X_rfe,y)
temp = pd.Series(rfe.support_,index = cols)
selected_features_rfe = temp[temp==True].index

x = data[selected_features_rfe]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)

```

```
x train:  (23, 278)
x test:  (23, 120)
y train:  (278,)
y test:  (120,)
```

In [27]:

```
print(color.BOLD + 'Logistic Regression with FS Wrapper Recursive Feature Elimination' + color.END)
print(color.BOLD + 'Results' + color.END)
logreg = linear_model.LogisticRegression(penalty='none',solver='saga',max_iter=20000)
logreg.fit(x_train.T, y_train.T)
theta=logreg.coef_
print("\n Feature Attributes:\n",x.columns)
print("\n Coefficients of Feature Attributes:\n",theta)

print("\n test accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_test.T, y_test.T)))
print("train accuracy: {} ".format(logreg.fit(x_train.T, y_train.T).score(x_train.T, y_train.T)))
y_pred=logreg.predict(x_test.T)

print("\n Classification Report")
print(classification_report(y_test.T,y_pred))

#Graph Plot
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = logreg.predict_proba(x_test.T)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test.T, preds)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')

#Confusion Matrix
print("\n Confusion Matrix")
cm = confusion_matrix(y_test.T,y_pred)
print(cm)

```

K-Fold Logistic Regression Results and Implementation

1. K-Fold Logistic Regression with no regularisation and no feature selection

Results

```
KFold Results

 accuracy mean score: 0.96

 accuracy stdev score: 0.033

 precision mean score: 0.959

 precision stdev score: 0.036

 recall mean score: 0.955

 recall stdev score: 0.033

 F-measure mean score: 0.956

 F-measure stdev score: 0.033

 ROC Area mean score: 0.986

 ROC Area stdev score: 0.019

 accuracy score list: [0.975, 0.925, 0.975, 0.975, 0.925, 0.95, 1.0, 0.975, 0.8974358974358975, 1.0]

 precision score list: [0.9782608695652174, 0.9045584045584045, 0.9545454545454546, 0.984375, 0.925, 0.96296296296296
3, 1.0, 0.98, 0.8986842105263158, 1.0]

 recall score list: [0.9722222222222222, 0.9226190476190477, 0.9833333333333334, 0.9444444444444444, 0.92606516290726
81, 0.9333333333333333, 1.0, 0.96875, 0.9007936507936507, 1.0]

 F-measure score list: [0.9746031746031746, 0.9127272727272726, 0.9677158999192896, 0.9626517273576096, 0.92495309568
4803, 0.945054945054945, 1.0, 0.9736668861092824, 0.8973684210526316, 1.0]

 ROC Area score list: [1.0, 0.9821428571428571, 1.0, 0.9390681003584229, 0.9949874686716792, 0.9733333333333334, 1.0,
0.9973958333333334, 0.9708994708994709, 1.0]

 Confusion metric:
 [[24.5  0.8]
 [ 0.8 13.7]]
```

Implementation:

```
print(color.BOLD + 'Logistic Regression with no regularisation and no feature selection' + color.END)
print(color.BOLD + 'KFold Results' + color.END)


confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logreg.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logreg.predict(x_test)))
        predictions = logreg.predict(x_test)
        predictions_proba = logreg.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))



print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))


print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)


```

2. K-Fold Logistic Regression with no regularisation and no feature selection with Bagging

Results

Logistic Regression with no regularisation and no feature selection Bagging
KFold Results

accuracy mean score: 0.975

accuracy stdev score: 0.025

precision mean score: 0.975

precision stdev score: 0.03

recall mean score: 0.97

recall stdev score: 0.027

F-measure mean score: 0.972

F-measure stdev score: 0.028

ROC Area mean score: 0.989

ROC Area stdev score: 0.018

accuracy score list: [1.0, 0.925, 1.0, 0.975, 0.95, 0.975, 1.0, 0.975, 0.9487179487179487, 1.0]

precision score list: [1.0, 0.9045584045584045, 1.0, 0.984375, 0.9523809523809523, 0.9807692307692308, 1.0, 0.98, 0.9484126984126984, 1.0]

recall score list: [1.0, 0.9226190476190477, 1.0, 0.9444444444444444, 0.9523809523809523, 0.9666666666666667, 1.0, 0.96875, 0.9484126984126984, 1.0]

F-measure score list: [1.0, 0.9127272727272726, 1.0, 0.9626517273576096, 0.9500000000000001, 0.9729546991210277, 1.0, 0.9736668861092824, 0.9484126984126984, 1.0]

ROC Area score list: [1.0, 0.9791666666666666, 1.0, 0.9390681003584229, 0.9924812030075187, 0.992, 1.0, 1.0, 0.9920634920634921, 1.0]

Confusion metric:
[[24.8 0.5]
[0.5 14.]]

Implementation:

```
print(color.BOLD + 'Logistic Regression with no regularisation and no feature selection Bagging' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

logregBC = BaggingClassifier(logreg)

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logregBC.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logregBC.predict(x_test)))
        predictions = logregBC.predict(x_test)
        predictions_proba = logregBC.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))


print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

3. K-Fold Logistic Regression with L1 Regularisation

Results

```
Logistic Regression with L1 regularisation
KFold Results

 accuracy mean score: 0.97

 accuracy stdev score: 0.027

 precision mean score: 0.972

 precision stdev score: 0.032

 recall mean score: 0.962

 recall stdev score: 0.032

 F-measure mean score: 0.966

 F-measure stdev score: 0.032

 ROC Area mean score: 0.989

 ROC Area stdev score: 0.032

 accuracy score list: [0.975, 0.9, 1.0, 0.975, 0.95, 0.975, 1.0, 0.975, 0.9743589743589743, 0.9743589743589743]

 precision score list: [0.9782608695652174, 0.8809523809523809, 1.0, 0.984375, 0.9565217391304348, 0.980769230769230
8, 1.0, 0.98, 0.9772727272727273, 0.9791666666666667]

 recall score list: [0.9722222222222222, 0.8809523809523809, 1.0, 0.9444444444444444, 0.9473684210526316, 0.966666666
6666667, 1.0, 0.96875, 0.9722222222222222, 0.96875]

 F-measure score list: [0.9746031746031746, 0.8809523809523809, 1.0, 0.9626517273576096, 0.9494949494949494, 0.972954
6991210277, 1.0, 0.9736668861092824, 0.9740863787375416, 0.973232669869595]

 ROC Area score list: [1.0, 0.9642857142857143, 1.0, 0.956989247311828, 0.9799498746867168, 0.9973333333333333, 1.0,
0.9973958333333334, 0.9973544973544973, 1.0]

 Confusion metric:
 [[25.1  0.2]
 [ 1.  13.5]]
```

Implementation:

```
print(color.BOLD + 'Logistic Regression with L1 regularisation' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logreg.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logreg.predict(x_test)))
        predictions = logreg.predict(x_test)
        predictions_proba = logreg.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(f1_scores), 3))

print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)



CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

4. K-Fold Logistic Regression with L1 Regularisation with Bagging

Results:

```
Logistic Regression with L1 regularisation Bagging
KFold Results

 accuracy mean score: 0.965

 accuracy stdev score: 0.023

 precision mean score: 0.968

 precision stdev score: 0.03

 recall mean score: 0.953

 recall stdev score: 0.026

 F-measure mean score: 0.96

 F-measure stdev score: 0.027

 ROC Area mean score: 0.99

 ROC Area stdev score: 0.015

 accuracy score list: [0.975, 0.9, 0.975, 0.975, 0.95, 0.975, 0.975, 0.975, 0.9743589743589743, 0.9743589743589743]

 precision score list: [0.9782608695652174, 0.8809523809523809, 0.9838709677419355, 0.984375, 0.9565217391304348, 0.9
807692307692308, 0.9827586206896552, 0.98, 0.9772727272727273, 0.9791666666666667]

 recall score list: [0.9722222222222222, 0.8809523809523809, 0.95, 0.9444444444444444, 0.9473684210526316, 0.96666666
66666667, 0.9583333333333333, 0.96875, 0.9722222222222222, 0.96875]

 F-measure score list: [0.9746031746031746, 0.8809523809523809, 0.9654874892148404, 0.9626517273576096, 0.94949494949
49494, 0.9729546991210277, 0.969488939740656, 0.9736668861092824, 0.9740863787375416, 0.973232669869595]

 ROC Area score list: [1.0, 0.9613095238095238, 1.0, 0.9605734767025089, 0.9899749373433584, 0.9973333333333333, 1.0,
0.9973958333333334, 0.9973544973544973, 1.0]

 Confusion metric:
 [[25.1  0.2]
 [ 1.2 13.3]]
```

Implementation:

```
print(color.BOLD + 'Logistic Regression with L1 regularisation Bagging' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

logregBC = BaggingClassifier(logreg)

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logregBC.fit(x_train, y_train)
        #confusionmatrix

        confusionmatrix.append(confusion_matrix(y_test, logregBC.predict(x_test)))
        predictions = logregBC.predict(x_test)
        predictions_proba = logregBC.predict_proba(x_test)


        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))

print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

5. K-Fold Logistic Regression with Feature Selection - Filter

Results:

```
Logistic Regression with FS - Filter
KFold Results

 accuracy mean score: 0.95

 accuracy stdev score: 0.039

 precision mean score: 0.948

 precision stdev score: 0.04

 recall mean score: 0.942

 recall stdev score: 0.04

 F-measure mean score: 0.944

 F-measure stdev score: 0.04

 ROC Area mean score: 0.988

 ROC Area stdev score: 0.013

 accuracy score list: [0.975, 0.925, 0.975, 0.95, 0.875, 0.95, 1.0, 1.0, 0.8974358974358975, 0.9487179487179487]

 precision score list: [0.9782608695652174, 0.9200626959247649, 0.9838709677419355, 0.9283154121863799, 0.87626262626
26263, 0.9466666666666667, 1.0, 1.0, 0.8986842105263158, 0.9470108695652174]

 recall score list: [0.9722222222222222, 0.8988095238095238, 0.95, 0.9283154121863799, 0.8734335839598997, 0.94666666
66666667, 1.0, 1.0, 0.9007936507936507, 0.9470108695652174]

 F-measure score list: [0.9746031746031746, 0.908466819221968, 0.9654874892148404, 0.9283154121863799, 0.874292897548
7115, 0.9466666666666667, 1.0, 1.0, 0.8973684210526316, 0.9470108695652174]

 ROC Area score list: [0.9974747474747475, 0.9672619047619048, 1.0, 0.982078853046595, 0.9699248120300752, 0.99466666
66666666, 1.0, 1.0, 0.9708994708994709, 0.9972826086956522]

 Confusion metric:
 [[24.4  0.9]
 [ 1.1 13.4]]
```

Implementation

```
print(color.BOLD + 'Logistic Regression with FS - Filter' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logreg.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logreg.predict(x_test)))
        predictions = logreg.predict(x_test)
        predictions_proba = logreg.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))


print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

6. K-Fold Logistic Regression with Feature Selection - Filter - With Bagging

Results

```
Logistic Regression with FS - Filter Bagging
KFold Results

 accuracy mean score: 0.947

 accuracy stdev score: 0.04

 precision mean score: 0.942

 precision stdev score: 0.045

 recall mean score: 0.942

 recall stdev score: 0.043

 F-measure mean score: 0.941

 F-measure stdev score: 0.044

 ROC Area mean score: 0.99

 ROC Area stdev score: 0.011

 accuracy score list: [0.95, 0.9, 0.975, 0.925, 0.875, 0.975, 1.0, 1.0, 0.9230769230769231, 0.9487179487179487]

 precision score list: [0.9494949494949495, 0.8809523809523809, 0.9838709677419355, 0.8833333333333333, 0.88541666666
66667, 0.96875, 1.0, 1.0, 0.9223684210526315, 0.9470108695652174]

 recall score list: [0.9494949494949495, 0.8809523809523809, 0.95, 0.9121863799283154, 0.8709273182957393, 0.98, 1.0,
1.0, 0.9246031746031746, 0.9470108695652174]

 F-measure score list: [0.9494949494949495, 0.8809523809523809, 0.9654874892148404, 0.8964624676445212, 0.87301587301
5873, 0.9736668861092824, 1.0, 1.0, 0.9228740936058009, 0.9470108695652174]

 ROC Area score list: [0.9974747474747475, 0.9672619047619048, 1.0, 0.982078853046595, 0.9774436090225564, 0.99466666
66666666, 1.0, 1.0, 0.9814814814814815, 0.9972826086956522]

 Confusion metric:
 [[24.3  1. ]
 [ 1.1 13.4]]
```

Implementation

```
print(color.BOLD + 'Logistic Regression with FS - Filter Bagging' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

logregBC = BaggingClassifier(logreg)

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logregBC.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logregBC.predict(x_test)))
        predictions = logregBC.predict(x_test)
        predictions_proba = logregBC.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))

print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

7. K-Fold Logistic Regression with Feature Selection - Wrapper - Backward Elimination

Results:

```
Logistic Regression with FS Wrapper - Backwardward Elimination
KFold Results

 accuracy mean score: 0.882

 accuracy stdev score: 0.05

 precision mean score: 0.882

 precision stdev score: 0.052

 recall mean score: 0.86

 recall stdev score: 0.056

 F-measure mean score: 0.864

 F-measure stdev score: 0.057

 ROC Area mean score: 0.942

 ROC Area stdev score: 0.034

 accuracy score list: [0.9, 0.875, 0.95, 0.85, 0.775, 0.925, 0.9, 0.825, 0.9230769230769231, 0.8974358974358975]

 precision score list: [0.898989898989899, 0.8475783475783476, 0.9333333333333333, 0.7849462365591398, 0.815476190476
1905, 0.9464285714285714, 0.9, 0.8510971786833856, 0.9375, 0.9042857142857144]

 recall score list: [0.898989898989899, 0.8630952380952381, 0.9333333333333333, 0.7849462365591398, 0.765664160401002
5, 0.9, 0.8571428571428572, 0.7916666666666667, 0.9166666666666667, 0.8845108695652174]

 F-measure score list: [0.898989898989899, 0.8545454545454545, 0.9333333333333333, 0.7849462365591398, 0.763001974983
5418, 0.9161425576519917, 0.8746081504702194, 0.8043326345213136, 0.9212121212121211, 0.8916666666666666]

 ROC Area score list: [0.9772727272727273, 0.8928571428571428, 0.9766666666666666, 0.9247311827956989, 0.884711779448
6216, 0.9493333333333334, 0.9702380952380952, 0.9218750000000001, 0.9867724867724867, 0.9375000000000001]

 Confusion metric:
 [[24.   1.3]
 [ 3.4 11.1]]
```

Implementation:

```
print(color.BOLD + 'Logistic Regression with FS Wrapper - Backwardward Elimination' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logreg.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logreg.predict(x_test)))
        predictions = logreg.predict(x_test)
        predictions_proba = logreg.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))


print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

8. K-Fold Logistic Regression with Feature Selection - Wrapper - Backward Elimination - Bagging

Results:

```
Logistic Regression with FS Wrapper - Backwardward Elimination Bagging
KFold Results

 accuracy mean score: 0.882

 accuracy stdev score: 0.05

 precision mean score: 0.882

 precision stdev score: 0.052

 recall mean score: 0.86

 recall stdev score: 0.056

 F-measure mean score: 0.864

 F-measure stdev score: 0.057

 ROC Area mean score: 0.942

 ROC Area stdev score: 0.034

 accuracy score list: [0.9, 0.875, 0.95, 0.85, 0.775, 0.925, 0.9, 0.825, 0.9230769230769231, 0.8974358974358975]

 precision score list: [0.898989898989899, 0.8475783475783476, 0.9333333333333333, 0.7849462365591398, 0.815476190476
1905, 0.9464285714285714, 0.9, 0.8510971786833856, 0.9375, 0.9042857142857144]

 recall score list: [0.898989898989899, 0.8630952380952381, 0.9333333333333333, 0.7849462365591398, 0.765664160401002
5, 0.9, 0.8571428571428572, 0.7916666666666667, 0.9166666666666667, 0.8845108695652174]

 F-measure score list: [0.898989898989899, 0.8545454545454545, 0.9333333333333333, 0.7849462365591398, 0.763001974983
5418, 0.9161425576519917, 0.8746081504702194, 0.8043326345213136, 0.9212121212121211, 0.8916666666666666]

 ROC Area score list: [0.9772727272727273, 0.8928571428571428, 0.9766666666666666, 0.9247311827956989, 0.884711779448
6216, 0.9493333333333334, 0.9702380952380952, 0.9218750000000001, 0.9867724867724867, 0.9375000000000001]

 Confusion metric:
 [[24.   1.3]
 [ 3.4 11.1]]
```

Implementation

```
print(color.BOLD + 'Logistic Regression with FS Wrapper - Backwardward Elimination Bagging' + color.END)
print(color.BOLD + 'KFold Results' + color.END)


confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

logregBC = BaggingClassifier(logreg)

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logregBC.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logregBC.predict(x_test)))
        predictions = logregBC.predict(x_test)
        predictions_proba = logregBC.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))


print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)
```

9. K-Fold Logistic Regression with Feature Selection - Wrapper - Recursive Feature Elimination

Results:

```
Logistic Regression with FS Wrapper Recursive Feature Elimination
KFold Results

 accuracy mean score: 0.92

 accuracy stdev score: 0.048

 precision mean score: 0.918

 precision stdev score: 0.052

 recall mean score: 0.907

 recall stdev score: 0.054

 F-measure mean score: 0.908

 F-measure stdev score: 0.055

 ROC Area mean score: 0.967

 ROC Area stdev score: 0.035

 accuracy score list: [0.925, 0.925, 0.925, 0.875, 0.8, 0.925, 0.975, 0.925, 0.9743589743589743, 0.9487179487179487]

 precision score list: [0.9271099744245523, 0.9200626959247649, 0.8918495297805643, 0.8166666666666667, 0.83190883190
88319, 0.9464285714285714, 0.9615384615384616, 0.9444444444444444, 0.9772727272727273, 0.96]

 recall score list: [0.9217171717171717, 0.8988095238095238, 0.9166666666666667, 0.8405017921146953, 0.79197994987468
67, 0.9, 0.9821428571428572, 0.90625, 0.9722222222222222, 0.9375]

 F-measure score list: [0.9238095238095237, 0.908466819221968, 0.9031476997578693, 0.8274374460742018, 0.791666666666
6666, 0.9161425576519917, 0.9709090909090909, 0.9188640973630832, 0.9740863787375416, 0.9458333333333333]

 ROC Area score list: [0.9898989898989898, 0.9523809523809523, 0.99, 0.9713261648745519, 0.8822055137844611, 0.930666
6666666666, 1.0, 0.9921875, 0.9682539682539683, 0.9918478260869565]

 Confusion metric:
 [[24.4  0.9]
 [ 2.3 12.2]]
```

Implementation

```
print(color.BOLD + 'Logistic Regression with FS Wrapper Recursive Feature Elimination' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logreg.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logreg.predict(x_test)))
        predictions = logreg.predict(x_test)
        predictions_proba = logreg.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))


print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)

CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

10. K-Fold Logistic Regression with Feature Selection - Wrapper - Recursive Feature Elimination - Bagging

Result:

```
KFold Results

 accuracy mean score: 0.92

 accuracy stdev score: 0.05

 precision mean score: 0.915

 precision stdev score: 0.055

 recall mean score: 0.909

 recall stdev score: 0.055

 F-measure mean score: 0.908

 F-measure stdev score: 0.057

 ROC Area mean score: 0.969

 ROC Area stdev score: 0.033

 accuracy score list: [0.925, 0.925, 0.9, 0.875, 0.8, 0.925, 0.975, 0.95, 0.9743589743589743, 0.9487179487179487]

 precision score list: [0.9271099744245523, 0.9200626959247649, 0.8571428571428572, 0.8166666666666667, 0.8319088319088319, 0.9464285714285714, 0.9615384615384616, 0.9479166666666667, 0.9772727272727273, 0.96]

 recall score list: [0.9217171717171717, 0.8988095238095238, 0.9, 0.8405017921146953, 0.7919799498746867, 0.9, 0.9821428571428572, 0.9479166666666667, 0.9722222222222222, 0.9375]

 F-measure score list: [0.9238095238095237, 0.9084668192219680, 0.8746081504702194, 0.8274374460742018, 0.7916666666666666, 0.9161425576519917, 0.9709090909090909, 0.9479166666666667, 0.9740863787375416, 0.9458333333333333]

 ROC Area score list: [0.9898989898989898, 0.9553571428571429, 0.9866666666666667, 0.9713261648745519, 0.8897243107769424, 0.9333333333333332, 1.0, 0.9973958333333334, 0.9682539682539683, 0.9945652173913044]

 Confusion metric:
 [[24.2  1.1]
 [ 2.1 12.4]]
```

Implementation:

```
print(color.BOLD + 'Logistic Regression with FS Wrapper Recursive Feature Elimination' + color.END)
print(color.BOLD + 'KFold Results' + color.END)

confusionmatrix = []
accuracy_scores = []
precision_scores = []
recall_scores = []
f1_scores = []
roc_auc_scores = []

logregBC = BaggingClassifier(logreg)

for train_index,test_index in kf.split(x):
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y[train_index], y[test_index]
        logregBC.fit(x_train, y_train)
        #confusionmatrix
        confusionmatrix.append(confusion_matrix(y_test, logregBC.predict(x_test)))
        predictions = logregBC.predict(x_test)
        predictions_proba = logregBC.predict_proba(x_test)

        accuracy_scores.append(accuracy_score(y_test, predictions))
        precision_scores.append(precision_score(y_test, predictions, average="macro"))
        recall_scores.append(recall_score(y_test, predictions, average="macro"))
        f1_scores.append(f1_score(y_test, predictions, average="macro"))
        roc_auc_scores.append(roc_auc_score(y_test, predictions_proba[:,1]))

print("\n accuracy mean score:", np.round(np.mean(accuracy_scores), 3))
print("\n accuracy stdev score:", np.round(np.std(accuracy_scores), 3))

print("\n precision mean score:", np.round(np.mean(precision_scores), 3))
print("\n precision stdev score:", np.round(np.std(precision_scores), 3))

print("\n recall mean score:", np.round(np.mean(recall_scores), 3))
print("\n recall stdev score:", np.round(np.std(recall_scores), 3))

print("\n F-measure mean score:", np.round(np.mean(f1_scores), 3))
print("\n F-measure stdev score:", np.round(np.std(f1_scores), 3))

print("\n ROC Area mean score:", np.round(np.mean(roc_auc_scores), 3))
print("\n ROC Area stdev score:", np.round(np.std(roc_auc_scores), 3))

print("\n accuracy score list:", accuracy_scores)
print("\n precision score list:", precision_scores)
print("\n recall score list:", recall_scores)
print("\n F-measure score list:", f1_scores)
print("\n ROC Area score list:", roc_auc_scores)


CM=np.round(np.mean(confusionmatrix,axis=0),3)
print("\n Confusion metric: \n",CM)

```

Logistic Regression - Attempt at Manual Implementation of Logistic Regression

1. Setting up the Data

In [2]:
```
data2=pd.read_csv("breast-cancer-data-training.csv")
data2.drop(["id"], axis=1, inplace=True)
data2.diagnosis = [1 if each == "M" else 0 for each in data2.diagnosis]
y = data2.diagnosis.values
x_data = data2.drop(['diagnosis'], axis=1)
data2.diagnosis.head()
```

Out[2]:
```
0    1
1    1
2    0
3    1
4    0
Name: diagnosis, dtype: int64
```

In [3]:
```
x = (x_data -np.min(x_data))/(np.max(x_data)-np.min(x_data)).values
```

In [4]:
```
x.head()
```

Out[4]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	'concave points_mean'	symmetry_mean	fractal_din
0	0.631447	0.401038	0.612760	0.517814	0.389434	0.366603	0.373477	0.434592	0.478332	
1	0.549843	0.406574	0.541446	0.430913	0.272376	0.331023	0.227202	0.373111	0.567714	
2	0.401035	0.017647	0.387164	0.266381	0.471702	0.277652	0.207170	0.287177	0.431203	
3	0.548776	0.520415	0.526110	0.420734	0.206165	0.280719	0.282099	0.285089	0.220477	
4	0.303430	0.239446	0.287248	0.185588	0.399346	0.175633	0.074813	0.104722	0.350488	

5 rows × 30 columns

In [5]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.15, random_state=42)

x_train = x_train.T
x_test = x_test.T
y_train = y_train.T
y_test = y_test.T

print("x train: ",x_train.shape)
print("x test: ",x_test.shape)
print("y train: ",y_train.shape)
print("y test: ",y_test.shape)

```

```
x train:  (30, 338)
x test:  (30, 60)
y train:  (338,)
y test:  (60,)
```

2. Defining the functions

In [6]:
```
def initialize_weights_and_bias(dimension):
    w = np.full((dimension,1),0.01)
    b = 0.0
    return w, b

```

In [7]:
```
def sigmoid(z):
    y_head = 1/(1+np.exp(-z))
    return y_head

```

In [8]:
```
def forward_backward_propagation(w,b,x_train,y_train):
    z = np.dot(w.T,x_train) + b
    y_head = sigmoid(z)
    loss = -y_train*np.log(y_head)-(1-y_train)*np.log(1-y_head)
    cost = (np.sum(loss))/x_train.shape[1]

    derivative_weight = (np.dot(x_train,((y_head-y_train).T)))/x_train.shape[1]
    derivative_bias = np.sum(y_head-y_train)/x_train.shape[1]
    gradients = {"derivative_weight": derivative_weight,"derivative_bias": derivative_bias}
    return cost,gradients

```

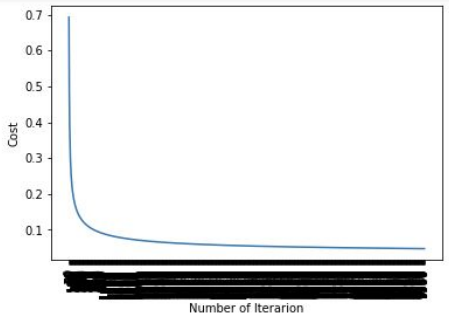
In [9]:
```
def update(w, b, x_train, y_train, learning_rate,number_of_iterarion):
    cost_list = []
    cost_list2 = []
    index = []

    for i in range(number_of_iterarion):
        cost,gradients = forward_backward_propagation(w,b,x_train,y_train)
        cost_list.append(cost)
        w = w - learning_rate * gradients["derivative_weight"]
        b = b - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_list2.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost))
    parameters = {"weight": w,"bias": b}
    plt.plot(index,cost_list2)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iterarion")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_list
```

In [10]:
```
def predict(w,b,x_test):
    z = sigmoid(np.dot(w.T,x_test)+b)
    Y_prediction = np.zeros((1,x_test.shape[1]))

    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1
    return Y_prediction
```

In [11]:
```
def logistic_regression(x_train, y_train, x_test, y_test, learning_rate ,  num_iterations):
    dimension =  x_train.shape[0]  # that is 4096
    w,b = initialize_weights_and_bias(dimension)
    parameters, gradients, cost_list = update(w, b, x_train, y_train, learning_rate,num_iterations)
    y_prediction_test = predict(parameters["weight"],parameters["bias"],x_test)
    y_prediction_train = predict(parameters["weight"],parameters["bias"],x_train)

    print("train accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_train - y_train)) * 100))
    print("test accuracy: {} %".format(100 - np.mean(np.abs(y_prediction_test - y_test)) * 100))

logistic_regression(x_train, y_train, x_test, y_test,learning_rate = 1, num_iterations = 10000)
```

```
Cost after iteration 9810: 0.047760
Cost after iteration 9820: 0.047751
Cost after iteration 9830: 0.047741
Cost after iteration 9840: 0.047731
Cost after iteration 9850: 0.047721
Cost after iteration 9860: 0.047711
Cost after iteration 9870: 0.047702
Cost after iteration 9880: 0.047692
Cost after iteration 9890: 0.047682
Cost after iteration 9900: 0.047673
Cost after iteration 9910: 0.047663
Cost after iteration 9920: 0.047653
Cost after iteration 9930: 0.047644
Cost after iteration 9940: 0.047634
Cost after iteration 9950: 0.047624
Cost after iteration 9960: 0.047615
Cost after iteration 9970: 0.047605
Cost after iteration 9980: 0.047595
Cost after iteration 9990: 0.047586
```

3. Result:




```
train accuracy: 99.40828402366864 %
test accuracy: 98.33333333333333 %
```

Final Evaluation Scores

Model	Accuracy	Precision	Recall	F1	ROC	MCC
Logistic Regression	0.970	0.970	0.966	0.968	0.989	0.936
SVM + AdaBoost	0.980	0.980	0.976	0.978	0.993	0.956
Decision Trees	0.937	0.932	0.929	0.930	0.930	0.861
Naive Bayes	0.932	0.928	0.924	0.926	0.983	0.852
Stacking MLP	0.970	0.970	0.966	0.968	0.993	0.936

Table 8.1. Evaluation scores of different models when fitted with training data using K-Fold cross-validation technique

Model	Accuracy	Precision	Recall	F1	ROC	MCC
Logistic Regression	0.976608	0.975459	0.975459	0.975459	0.996125	0.950918
SVM + AdaBoost	0.964912	0.961211	0.965844	0.963378	0.995264	0.927043
Decision Trees	0.941	0.942	0.942	0.941	0.990	0.877
Naive Bayes	0.953	0.959	0.959	0.959	0.994	0.914
Stacking MLP	0.976608	0.975459	0.975459	0.975459	0.995408	0.950918

Table 8.2. Evaluation scores of different models when fitted with test dataset