A Classification Method for the Wisconsin Breast Cancer Dataset Using Ensemble Learning

Derek Lankeaux
Rochester Institute of Technology
MS Applied Statistics

November 6, 2023

Contents

Abstract

Breast cancer is the most common type of cancer among women in the United States and represents almost a third of all cancer diagnoses among women. The objective of this study is to predict whether a breast cancer tumor is benign or malignant based on 30 characteristics. The Wisconsin Breast Cancer Dataset, available through the Kaggle and UCI Machine Learning Repository, is used to determine the associations that exist between the variables and their groupings. The results of the WBCD analysis pave the way for the incorporation of additional data, the implementation of Confirmatory Factor Analysis (CFA), and the integration with machine learning or causal inference. The experimental results show that the proposed ensemble method records an accuracy of 99%, along with a precision of 100% to classify breast cancer data, outperforming existing methods. Most of the scholarly work has assessed the capabilities of classifiers based on accuracy, which is a value that is higher when the frequencies of true positives (TPs) and true negatives (TNs) are greater than those of false positives (FPs) and false negatives (FNs). It was found that the highest ensemble performance was based on the original WBCD, and not with the subset of features driven by feature selection, nor the principal components added to the data from Exploratory Factor Analysis methods. This implies and confirms the highly nonlinear nature of the FNA characteristics of breast cancer tumors.

1 Introduction

1.1 Data Preprocessing

The Wisconsin Breast Cancer data set contains 569 biopsied breast cells, each with 30 characteristics. The features are calculated from a digital image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image. The data set contains two classes, benign (represented by the integer 0) and malignant (represented by the integer 1). Through careful analysis and interpretation of multivariate data, it is possible to narrow down the number of variables by employing various methods. This experimental study focuses on predicting whether a breast cancer tumor is benign or malignant based on 30 characteristics. Before a tumor is classified as benign or malignant, it is useful to perform exploratory data analysis before analyzing the data using univariate, bivariate, and finally multivariate methods. Data preprocessing involves cleaning the data, normalizing the data, and selecting features. Data cleaning involves removing missing values, outliers, or incorrect values. Normalizing the data involves scaling the data so that all features have the same range of values. Feature selection involves selecting the most relevant features from the data set that will be used for the model. This can be done using methods such as correlation analysis, recursive feature elimination, or principal component analysis.

1.2 Machine Learning

Ensemble learning methods are a type of machine learning algorithm that combines multiple models to produce better predictive performance than could be obtained from any of the individual models alone. This is done by either combining the predictions from multiple models or by learning a model that is a combination of multiple models.

1.2.1 Decision Tree

A decision tree is a graphical representation of possible solutions to a decision based on certain conditions. It is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including the outcomes of chance events, resource costs, and utility. Each branch of the tree represents a possible decision and the branches are connected by nodes representing the outcomes or results of the decisions made. The end nodes of the tree represent the final results or decisions. Mathematically, a decision tree can be represented as a directed acyclic graph (DAG), with each node representing a decision point and the edges representing the possible outcomes of the decision.

1.3 Ensemble Methods

Ensemble methods can combine multiple individual models to create a more powerful and accurate model than traditional machine learning algorithms. The Wisconsin Breast Cancer Dataset is a collection of data that can be used to train and test ensemble methods. In this Introduction, we will discuss the dataset, the types of ensemble methods that can be used, and the advantages and disadvantages of using ensemble methods on this dataset.

1.3.1 Random Forest

Random forest is an ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

1.3.2 Gradient Boosting

Gradient boosting is an ensemble learning technique that combines multiple weak learners to create a strong learner. It works by sequentially adding weak learners to the ensemble, each one correcting the errors of the previous one. The weak learners are decision trees, and the prediction is made by taking a weighted average of the predictions of all the trees.

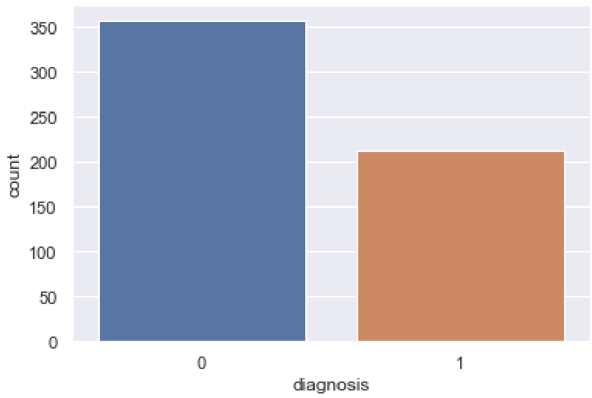


Figure 1: Dependent variable distribution

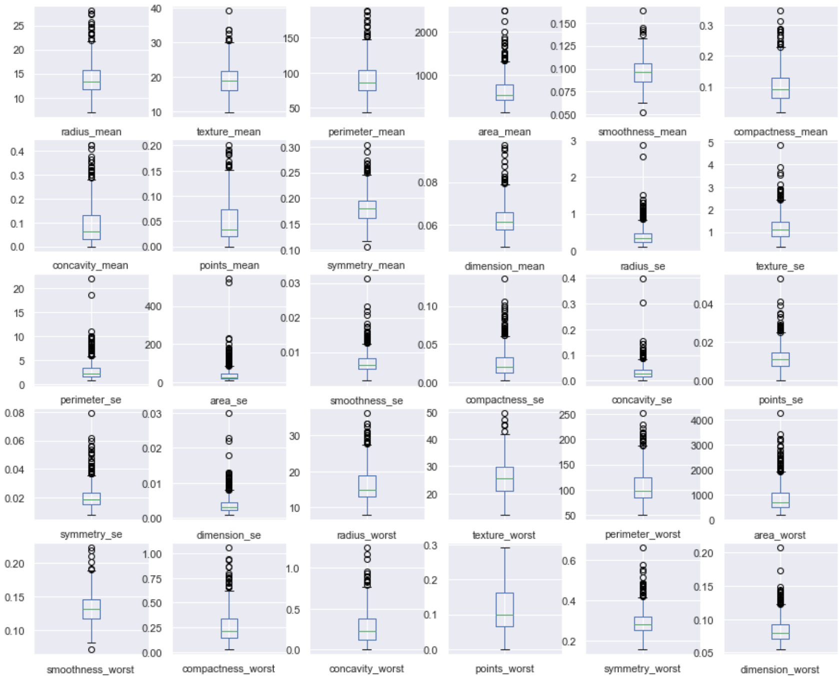


Figure 2: Boxplots of the 30 input features

1.3.3 AdaBoost

AdaBoost is an ensemble learning algorithm that combines multiple weak learners to form a strong learner. It works by assigning weights to the training examples, which are then used to build a model. The model is then used to make predictions on the test data.

1.3.4 Bagging Classifier

The Bagging classifier is an ensemble machine learning algorithm that combines multiple classifiers to improve the accuracy of the model. It works by training multiple models on different subsets of the data and then combining the results to get a more accurate prediction. The idea behind bagging is to reduce the variance of the model by combining the predictions of multiple models. This is done by training each model on a different subset of the data and then averaging the results.

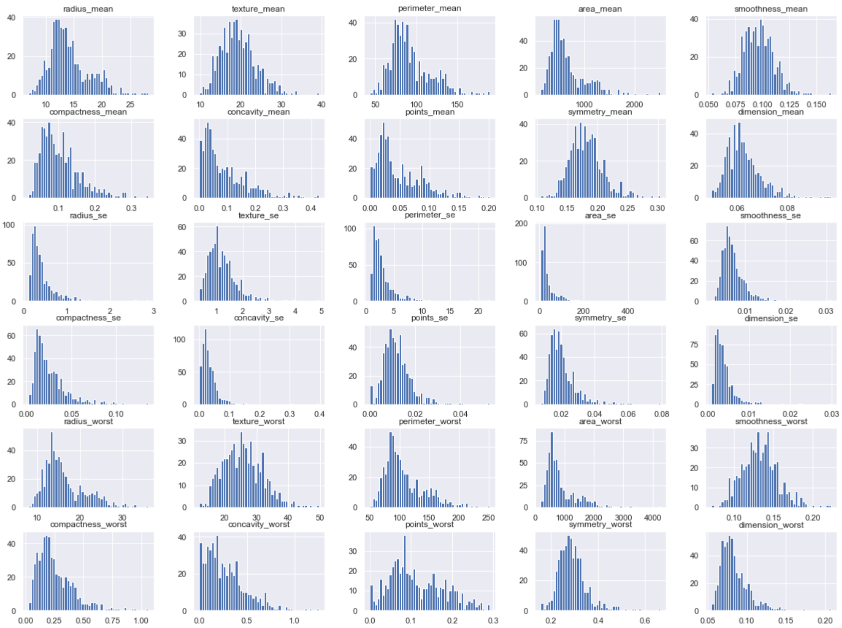


Figure 3: Histograms of the 30 input features

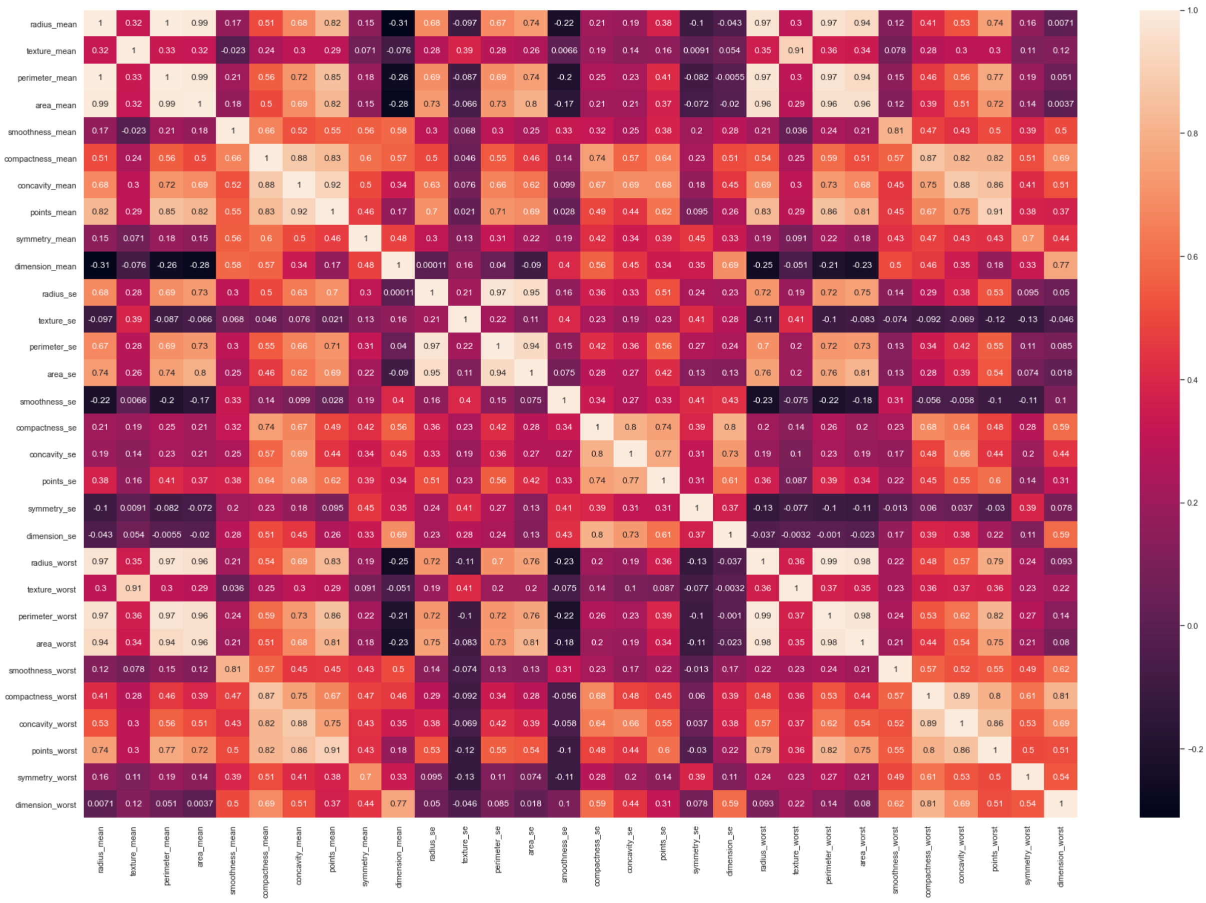


Figure 4: Correlogram of the 30 input features

2 Methods

2.1 Data Preprocessing

For WBCD ensemble learning, we can use a variety of data visualization strategies such as box plots, scatter plots, and heat maps. Additionally, we may use dimensionality reduction techniques, such as identifying multicollinearity, or principal component analysis (PCA), to reduce the number of features and create more interpretable visualizations. The primary concern between the initial variables is the high number of instances of multicollinearity that exist between the input variables. High instances of correlation between the dependent variables can adversely affect factor analysis and should be accounted for early on. For example, the mean and worst values for some of the variables were correlated with each other, as well as with other variables, which can be seen by looking at the correlation matrix. It is practical to then remove the highly correlated variables from the dataset for further analysis. For the given data, redundant features were dropped and exploratory factor analysis was used using principal component analysis to gain a sense of the correlation structure. After the first two principal components, volume complexity and texture characteristics, the features were added to the data set. However, after attempting to add them to the refined data set and the original, this resulted in a decrease in the overall accuracy of the four models.

2.2 Machine Learning

Following exploratory data analysis procedures, factor extraction can be determined by Principal Component Analysis. Determining the number of factors will be considered based on the three most common criteria, the Kaiser criterion, a scree plot, and the cumulative variance approach. Varimax rotation, an orthogonal rotation commonly used in EFA, was used to simplify the structure so that an accurate interpretation can be provided by factor loading analysis, indicating associations between the selected variables and factors. After evaluating the performance of the Random Forest algorithm with and without feature selection, it was concluded that the algorithms performed better in the absence of feature engineering. Therefore, we can assume that the structure of the WBCD is non-linear in nature, which calls for more advanced algorithms suited to the classification of the given data.

2.2.1 Decision trees

- Decision Tree function (data, features)
- Create a root node.

  If all examples in the data are of the same class, return the single-node tree root; with the label corresponding to this class.

  If the list of features is empty, return the single-node tree root with a label corresponding to the most common class in the data.

  Otherwise, try partitioning the data by each of the characteristics.
- Choose the feature that produces the most homogeneous subsets.
- Split the node into two children nodes.
- Recur on the sublists obtained by splitting the node.
- Return the root node of the tree.

2.3 Ensemble Methods

Ensemble learning is a machine learning technique that combines multiple models to create a more powerful model. The most common methods used in ensemble learning are bagging, boosting, and stacking. Bagging is a technique that uses multiple models to reduce the variance of a single model. Boosting is a technique that uses multiple models to reduce the bias of a single model. Stacking is a technique that combines multiple models to create a more powerful model. In addition, ensemble learning can also be used to improve the accuracy of a single model by combining multiple models with different parameters. The ensemble methods used in this project were as follows:

2.3.1 Random Forest

The random forest algorithm works by constructing M decision trees on bootstrap samples from training data. The prediction of the random forest is given by

$$\hat{Y} = \frac{1}{M}\sum_{m=1}^{M} f_m(X)$$

- For each tree in the forest:
  Select random data samples with replacement
  Build a Decision Tree based on samples
- Output the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

2.3.2 Gradient Boosting

If $\hat{Y}$ is the predicted output label.

$$F(x) = \sum_{m=1}^{M} h_m(x) = h_0(x) + \sum_{m=1}^{M} \gamma_m h_m(x)$$

- for i in the range:
  Fit a model to the current residuals. Make predictions on the training set.
  Compute the new residuals.
  Add the model to the ensemble.
- make predictions on the test set.

2.3.3 AdaBoost

The weights are updated in each iteration based on the accuracy of the predictions.

$$\hat{y} = \sum_{t=1}^{T} \alpha_t h_t(x)$$

$$\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$$

where T is the number of weak learners, h_t is the t-th weak learner, α_t is the weight assigned to the t-th weak learner, and ϵ_t is the error rate of the t-th weak learner

- Initialize the weights of each training example to 1/N, where N is the number of training examples
- For each iteration $t = 1, 2, ..., T$:
  Fit a classifier to the training data using weights
  Compute the weighted error rate ϵ_t of h_t
  Compute the coefficient $\alpha_t = 0.5 \times ln((1-\epsilon_t)/\epsilon_t)$
- Update the weights for each training example: $w_{t+1}(i) = w_t(i) \cdot \exp(-\alpha_t \cdot y_i \cdot h_t(x_i))$
- Output the final classifier: $F(x) = sign[\sum_{m=1}^{M} \alpha_m h_m(x)]$

2.3.4 Bagging Classifier

- Let $f_1, f_2, ..., f_n$ be the classifiers trained on different subsets of data. The bagging classifier is then given by the following:

$$F(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

  where x is the input data.

- The bagging classifier is an effective way to reduce the variance of the model and improve the accuracy of the predictions.

- for i = 1 to n:

  Sample a subset of training data with replacement.

  Train a classifier at the end of the sampled subset

  Predict the class of a new instance by aggregating the predictions of all the classifiers.

3 Results

3.1 Data Analysis

The WBCD data analysis follows an exploratory approach, without predefined hypothesis being tested. The data was preprocessed by subtracting the mean and dividing by the standard deviation for each observation. Multicollinearity was taken into account when selecting variables, and correlation was used to measure the linear association between pairs of variables. The boxplots and the correlogram of the covariances are meant to understand potential associations amongst the input variables. Principal Component Analysis was used to determine the covariance structure of the data, and 6 factors were retained for the analysis. The limitations of this project include a limited dataset scope, a single snapshot of biopsy samples, and assumptions of linearity. Further analysis could include Confirmatory Factor Analysis, Machine Learning, and Causal Inference. Ensemble learning is used in data analysis to improve the accuracy of predictions by combining the results of multiple models.

3.2 Hyperparameter Tuning

Optimizing the hyperparameters of a set of models, as well as the hyperparameters of the set itself, is known as hyperparameter tuning. This can be done using a variety of techniques, such as grid search, random search, or Bayesian optimization. The aim is to identify the combination of hyperparameters that produces the best performance in the validation set. For this project, the grid search technique was used. This involves manually specifying a subset of the hyperparameter space of a learning algorithm and then exhaustively searching through it to find the best combination of hyperparameters for the model. Grid search works by defining a grid of hyperparameter values and then evaluating the model performance for each combination of values. The Grid Search procedure involves the following steps:

- Grid Search CV Steps:

  1. Define the hyperparameters to be tuned.

  2. Define the range of values for each hyperparameter.

  3. Create a grid of all possible combinations of hyperparameter values.

  4. Train a model for each combination of the hyperparameter values.

5. Evaluate the performance of each model.
6. Select the model with the best performance.
7. Refine the values of hyperparameters to further improve model performance.
8. Retrain the model with the refined hyperparameter values.
9. Evaluate the model performance with the refined hyperparameter values.
10. Select the model with the best performance.

The hyperparameter combination that produces the highest performance is then chosen as the optimal hyperparameter subset for the given model.

3.3 Model Evaluation

There are several methods to evaluate machine learning models, including precision, precision, recall, F1 score, ROC curve, and confusion matrix. Accuracy measures the proportion of correct predictions made by the model, while precision and recall measure the model's ability to identify true positives and true negatives. The F1 score is a combination of precision and recall, and the ROC curve is a graphical representation of the performance of the model. The confusion matrix is a table that shows the number of true positives, true negatives, false positives, and false negatives produced by the model. The following four metrics will be used to assess the performance of different Ensemble Methods:

- $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- $Precision = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $F1score = 2 \times \frac{Precision \times Recall}{Precision+Recall}$

4 Discussion

4.1 Data Analysis

Ensemble learning can be used for both classification and regression tasks. In mathematical terms, ensemble learning is a combination of multiple models that are trained on the same data set and then combined to make a prediction. Models can be of different types, such as decision trees, neural networks, support vector machines, and other methods. This project used the decision tree. Combining the models in such a way that the prediction accuracy is improved. The model that had the highest value for accuracy and precision was the adaptive boost algorithm, or AdaBoost (adaptive boost), which uses a combination of weak learners to create a strong learner. It is based on the idea of boosting, which is to combine multiple weak learners to form a strong learner. The weak learners used in AdaBoost are decision trees with a single split, also known as decision stumps. The algorithm works by weighting the observations so that observations that are misclassified gain more weight. The algorithm then fits the new weighted data and develops a new decision tree. This process is repeated for a specified number of iterations, and the observations are assigned higher weights if they are misclassified. The final model is the weighted mean of all decision trees.

4.2 Hyperparameter Tuning

Optimizing hyperparameters for ensemble methods requires adjusting the hyperparameters of each individual model in the ensemble, as well as those of the ensemble itself. This can be done using a variety of techniques, such as grid search, random search, or Bayesian optimization. The goal is to identify the combination of hyperparameters that yields the best performance in the validation set. The process method used throughout this project was grid search, a technique used to tune hyperparameters of a model by exhaustively searching through systematically exploring a manually specified subset of the hyperparameter space of a learning algorithm. It is commonly used in machine learning to find the best combination of hyperparameters for a given model. Grid search works by defining a grid of hyperparameter values and then evaluating the performance of the model for each combination of values.

4.3 Model Evaluation

Evaluation of machine learning models is an important step in the development process. There are several methods to assess the performance of machine learning models, including precision, such as precision, precision, recall, F1 score, ROC curve, and confusion matrix. Accuracy measures are the proportion of correct predictions made by the model, while precision and recall measure the model's ability to identify true positives and true negatives. The F1 score is a combination of precision and recall, and the ROC curve is a graphical representation of the performance of the model. The confusion matrix is a table that displays the number of true positives, true negatives, false positives, and false negatives generated by the model. The evaluation of machine learning models is a critical step in the development process.

Classifier	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Random Forest	.973684	.951220	.975	.962963
Gradient Boosting	.964912	.928571	.975	.951220
AdaBoost	.991228	1.0	.975	.987342
Bagging	.973684	.951220	.975	.962963

5 Conclusion

Ensemble learning is a powerful technique for enhancing the precision of machine learning models. The Wisconsin Breast Cancer Dataset is a great illustration of how ensemble learning can be utilized to boost the accuracy of a model. By combining multiple models, the accuracy of the model can be significantly improved, particularly when the models are trained on different subsets of the data. Ultimately, principal component analysis is not a viable solution for many data analysis tasks due to its limited capacity to capture intricate relationships between variables. Additionally, it is not suitable for dealing with high-dimensional data, as it is prone to overfitting and can lead to inaccurate results. Therefore, it is essential to consider other methods of data analysis when dealing with complex datasets. Future directions for research on breast cancer ensemble methods include exploring the use of different types of ensemble methods, such as boosting and bagging, to improve the accuracy of predictions; investigating the use of different types of data, such as genomic and proteomic data, to improve the accuracy of predictions; and exploring the use of different types of feature selection techniques to improve the accuracy of predictions.

References

1 https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

2 Richard Arnold Johnson, Dean W. Wichern. (2007). Applied Multivariate Statistical Analysis, 6th Edition. Pearson Prentice Hall.

3 Kunapuli, G. (2023). Ensemble Methods for Machine Learning. Simon and Schuster.

4 Dr. Josephine Prem Kumar. (2020). Identification of factors that cause breast cancer by factor analysis. International Journal of Engineering Research and Technology (IJERT), Volume 9 (issue 2), p1-4.
https://www.ijert.org/research/identification-of-factors-causing-breast-cancer-using-factor-analysis-IJERTV9IS020031.pdf

5 Khandaker Mohammad Mohi Uddin, Nitish Biswas, Sarreha Tasmin Rikta, Samrat Kumar Dey, Machine learning-based diagnosis of breast cancer utilizing feature optimization technique, Computer Methods and Programs in Biomedicine Update, Volume 3, 2023, 100098, ISSN 2666-9900.
https://www.sciencedirect.com/science/article/pii/S2666990023000071

6 Ylmaz Kaya, Fatma Kuncan, A hybrid model for classification of medical data set based on factor analysis and extreme learning machine: FA+ELM, Biomedical Signal Processing and Control, Volume 78, 2022, 104023, ISSN 1746-8094.
https://www.sciencedirect.com/science/article/pii/S1746809422005018

STAT 790: CAPSTONE PROJECT- APPENDIX

November 6, 2023

I. Data Preprocessing

Import the necessary libraries.

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split, ParameterGrid, cross_val_score,␣
 ↪RepeatedKFold, RepeatedStratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import warnings  as ws
ws.filterwarnings("ignore")
```

[2]:
```
WBCD = pd.read_csv("Original WBCD.csv")
WBCD.head()
```

[2]:
```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0  87139402         B        12.32         12.39           78.85      464.1
1   8910251         B        10.60         18.95           69.28      346.4
2    905520         B        11.04         16.83           70.92      373.2
3    868871         B        11.28         13.39           73.00      384.8
4   9012568         B        15.19         13.21           97.65      711.8

   smoothness_mean  compactness_mean  concavity_mean  points_mean  ...  \
0          0.10280           0.06981         0.03987      0.03700  ...
1          0.09688           0.11470         0.06387      0.02642  ...
2          0.10770           0.07804         0.03046      0.02480  ...
3          0.11640           0.11360         0.04635      0.04796  ...
4          0.07963           0.06934         0.03393      0.02657  ...

   radius_worst  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0         13.50          15.64            86.97       549.1            0.1385
1         11.88          22.94            78.28       424.8            0.1213
2         12.41          26.44            79.93       471.4            0.1369
3         11.92          15.77            76.53       434.0            0.1367
4         16.20          15.73           104.50       819.1            0.1126

   compactness_worst  concavity_worst  points_worst  symmetry_worst  \
```

```
0             0.1266          0.12420        0.09391          0.2827
1             0.2515          0.19160        0.07926          0.2940
2             0.1482          0.10670        0.07431          0.2998
3             0.1822          0.08669        0.08611          0.2102
4             0.1737          0.13620        0.08178          0.2487

   dimension_worst
0          0.06771
1          0.07587
2          0.07881
3          0.06784
4          0.06766

[5 rows x 32 columns]
```

Descriptive statistics.

[3]:
```python
WBCD.describe().T
```

[3]:
```
                   count          mean           std          min  \
id                 569.0  3.037183e+07  1.250206e+08  8670.000000
radius_mean        569.0  1.412729e+01  3.524049e+00     6.981000
texture_mean       569.0  1.928965e+01  4.301036e+00     9.710000
perimeter_mean     569.0  9.196903e+01  2.429898e+01    43.790000
area_mean          569.0  6.548891e+02  3.519141e+02   143.500000
smoothness_mean    569.0  9.636028e-02  1.406413e-02     0.052630
compactness_mean   569.0  1.043410e-01  5.281276e-02     0.019380
concavity_mean     569.0  8.879932e-02  7.971981e-02     0.000000
points_mean        569.0  4.891915e-02  3.880284e-02     0.000000
symmetry_mean      569.0  1.811619e-01  2.741428e-02     0.106000
dimension_mean     569.0  6.279761e-02  7.060363e-03     0.049960
radius_se          569.0  4.051721e-01  2.773127e-01     0.111500
texture_se         569.0  1.216853e+00  5.516484e-01     0.360200
perimeter_se       569.0  2.866059e+00  2.021855e+00     0.757000
area_se            569.0  4.033708e+01  4.549101e+01     6.802000
smoothness_se      569.0  7.040979e-03  3.002518e-03     0.001713
compactness_se     569.0  2.547814e-02  1.790818e-02     0.002252
concavity_se       569.0  3.189372e-02  3.018606e-02     0.000000
points_se          569.0  1.179614e-02  6.170285e-03     0.000000
symmetry_se        569.0  2.054230e-02  8.266372e-03     0.007882
dimension_se       569.0  3.794904e-03  2.646071e-03     0.000895
radius_worst       569.0  1.626919e+01  4.833242e+00     7.930000
texture_worst      569.0  2.567722e+01  6.146258e+00    12.020000
perimeter_worst    569.0  1.072612e+02  3.360254e+01    50.410000
area_worst         569.0  8.805831e+02  5.693570e+02   185.200000
smoothness_worst   569.0  1.323686e-01  2.283243e-02     0.071170
compactness_worst  569.0  2.542650e-01  1.573365e-01     0.027290
concavity_worst    569.0  2.721885e-01  2.086243e-01     0.000000
points_worst       569.0  1.146062e-01  6.573234e-02     0.000000
symmetry_worst     569.0  2.900756e-01  6.186747e-02     0.156500
dimension_worst    569.0  8.394582e-02  1.806127e-02     0.055040
```

```
                           25% 50% 75% max id 869218.000000 906024.000000 8.813129e +␣
↪06 9.113205e + 08 radius_mean 11.700000 13.370000 1.578000e + 01 2.811000e + 01␣
↪texture_mean 16.170000 18.840000 2.180000e + 01 3.928000e + 01 perimeter_mean 75.
↪170000 86.240000 1.041000e + 02 1.885000e + 02 area_mean 420.300000 551.100000 7.
↪827000e + 02 2.501000e + 03 smoothness_mean 0.086370 0.095870 1.053000e-01 1.
↪634000e-01 compactness_mean 0.064920 0.092630 1.304000e-01 3.454000e-01␣
↪concavity_mean 0.029560 0.061540 1.307000e-01 4.268000e-01 points_mean 0.020310 0.
↪033500 7.400000e-02 2.012000e-01 symmetry_mean 0.161900 0.179200
```

Value counts.

```
[4]: WBCD['diagnosis'].value_counts()
```

```
[4]: B    357
     M    212
     Name: diagnosis, dtype: int64
```

```
[5]: WBCD['diagnosis'] = WBCD['diagnosis'].map({'B': 0, 'M': 1})
```

```
[6]: WBCD.head()
```

```
[6]:          id  diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
     0  87139402          0        12.32         12.39           78.85      464.1
     1   8910251          0        10.60         18.95           69.28      346.4
     2    905520          0        11.04         16.83           70.92      373.2
     3    868871          0        11.28         13.39           73.00      384.8
     4   9012568          0        15.19         13.21           97.65      711.8

        smoothness_mean  compactness_mean  concavity_mean  points_mean  ...  \
     0          0.10280           0.06981         0.03987      0.03700  ...
     1          0.09688           0.11470         0.06387      0.02642  ...
     2          0.10770           0.07804         0.03046      0.02480  ...
     3          0.11640           0.11360         0.04635      0.04796  ...
     4          0.07963           0.06934         0.03393      0.02657  ...

        radius_worst  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
     0         13.50          15.64            86.97       549.1            0.1385
     1         11.88          22.94            78.28       424.8            0.1213
     2         12.41          26.44            79.93       471.4            0.1369
     3         11.92          15.77            76.53       434.0            0.1367
     4         16.20          15.73           104.50       819.1            0.1126

        compactness_worst  concavity_worst  points_worst  symmetry_worst  \
     0             0.1266          0.12420       0.09391          0.2827
     1             0.2515          0.19160       0.07926          0.2940
     2             0.1482          0.10670       0.07431          0.2998
     3             0.1822          0.08669       0.08611          0.2102
     4             0.1737          0.13620       0.08178          0.2487

        dimension_worst
     0          0.06771
     1          0.07587
     2          0.07881
     3          0.06784
```

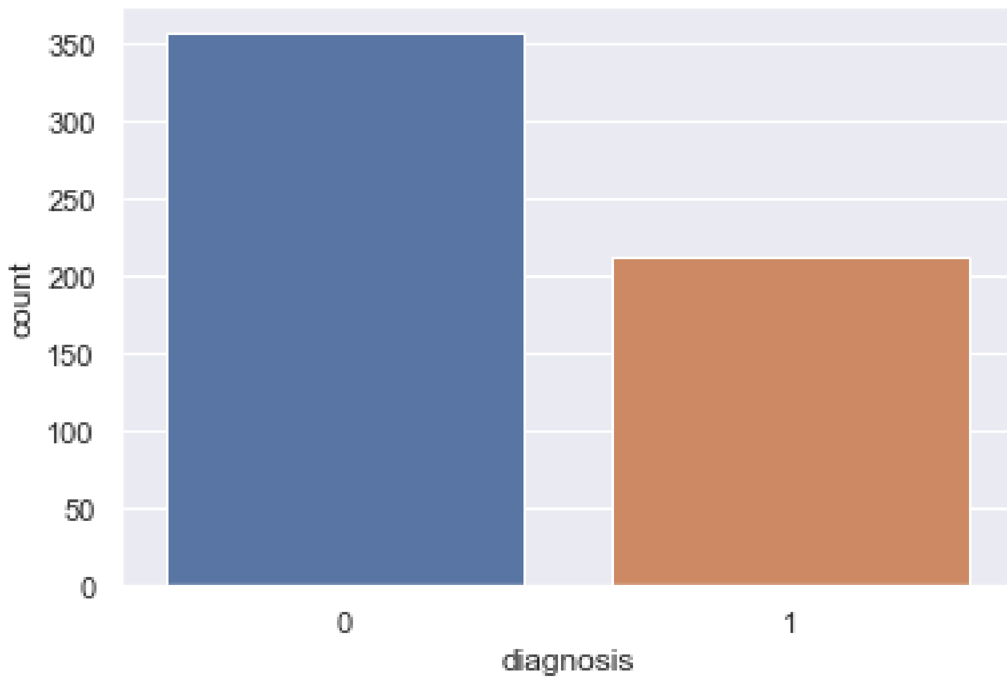
```
4          0.06766

[5 rows x 32 columns]
```

Data Visualization.

a. Dependent Variable distribution

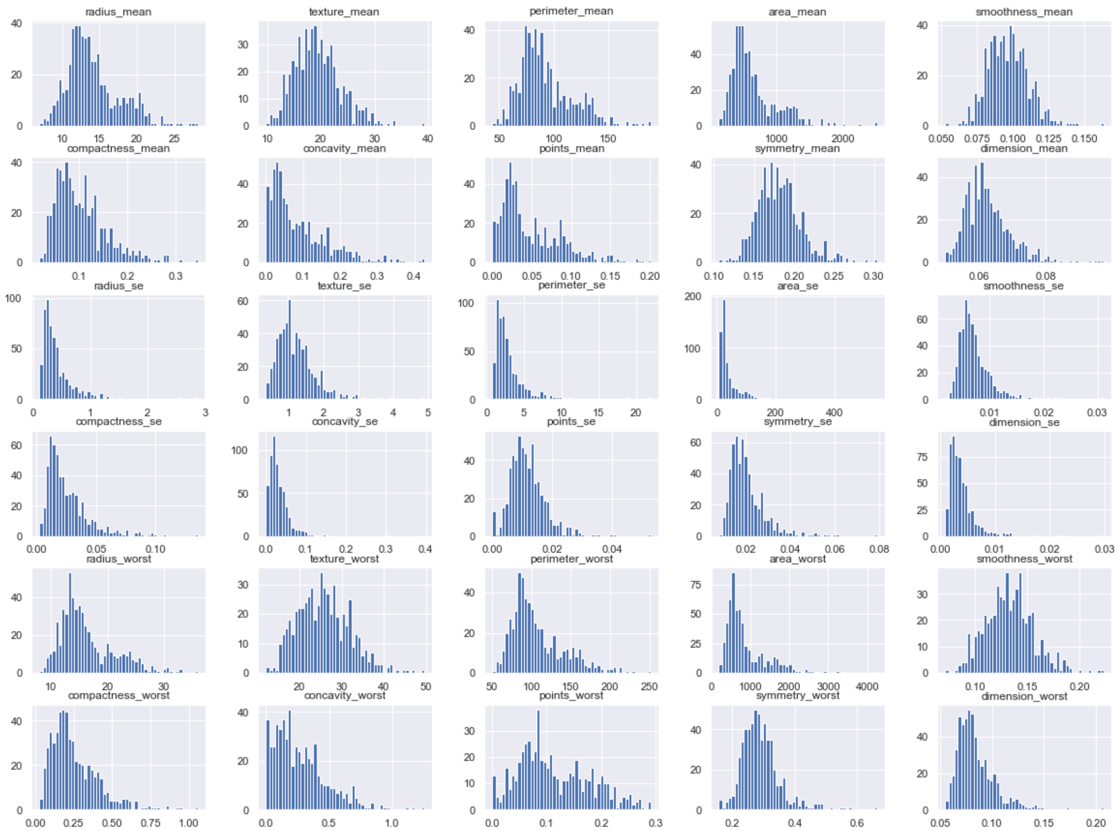
[7]:
```
sns.set()
sns.countplot(WBCD["diagnosis"])
plt.show()
```



b. Histograms of remaining variables' univariate distributions.

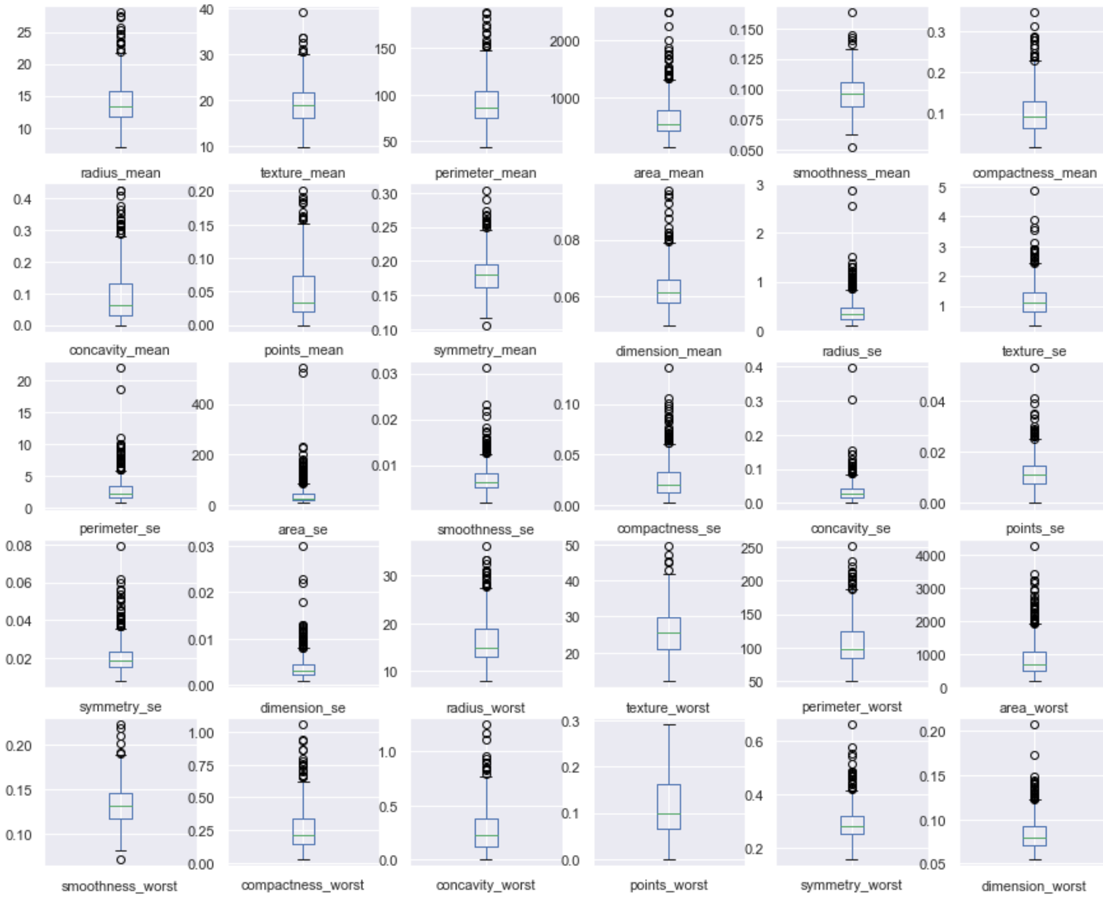
[8]:
```
X = WBCD.iloc[:,2:32]
y = WBCD.iloc[:,1]
```

[9]:
```
WBCD.iloc[:,2:32].hist(bins=50, figsize=(20, 15))
plt.show()
```

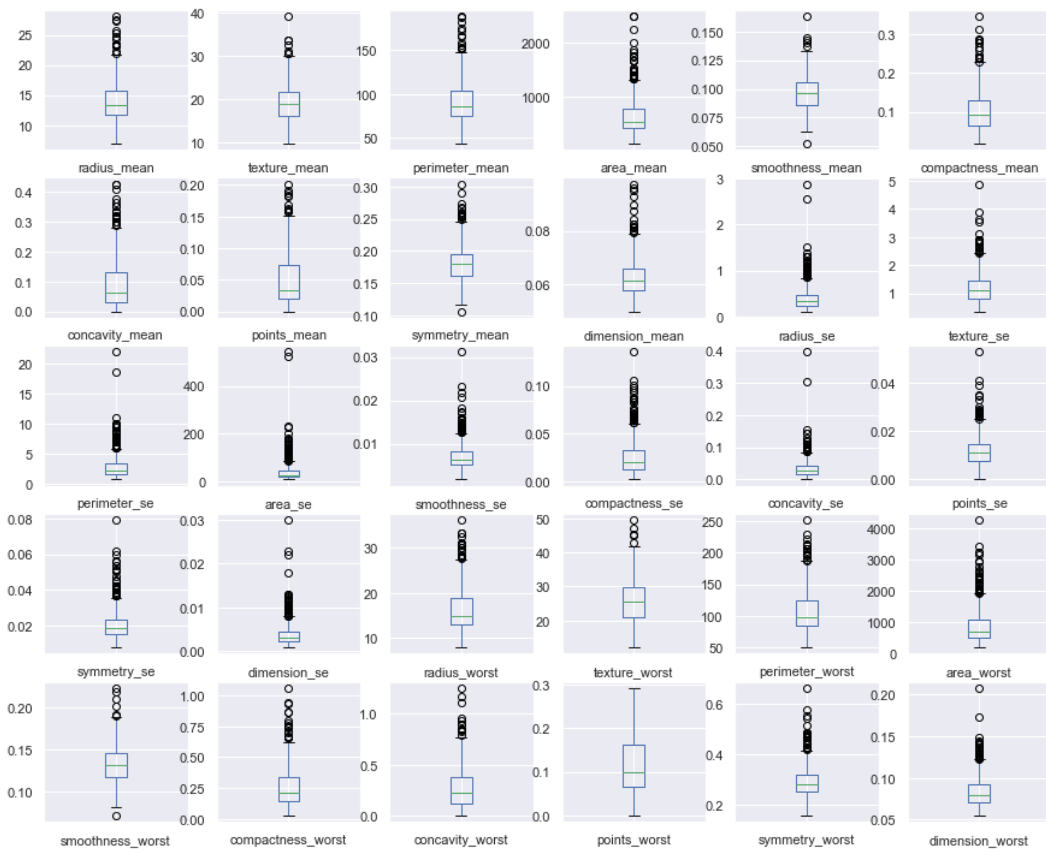


c. Boxplots of remaining variables' univariate distributions.

[10]:
```
WBCD.iloc[:,2:32].plot(kind='box', subplots=True, layout=(6, 6), sharex=False,␣
↪sharey=False, figsize=(15, 15))
plt.show()
```



radius_mean
texture_mean
perimeter_mean
area_mean
smoothness_mean
compactness_mean
concavity_mean
points_mean
symmetry_mean
dimension_mean
radius_se
texture_se
perimeter_se
area_se
smoothness_se
compactness_se
concavity_se
points_se
symmetry_se
dimension_se
radius_worst
texture_worst
perimeter_worst
area_worst
smoothness_worst
compactness_worst
concavity_worst
points_worst
symmetry_worst
dimension_worst



New correlation structure.

[11]:
```
WBCD.iloc[:,2:32].corr()
```

[11]:
```
                    radius_mean  texture_mean  perimeter_mean  area_mean  \
radius_mean            1.000000      0.323782        0.997855   0.987357
texture_mean           0.323782      1.000000        0.329533   0.321086
perimeter_mean         0.997855      0.329533        1.000000   0.986507
area_mean              0.987357      0.321086        0.986507   1.000000
smoothness_mean        0.170581     -0.023389        0.207278   0.177028
compactness_mean       0.506124      0.236702        0.556936   0.498502
concavity_mean         0.676764      0.302418        0.716136   0.685983
points_mean            0.822529      0.293464        0.850977   0.823269
symmetry_mean          0.147741      0.071401        0.183027   0.151293
dimension_mean        -0.311631     -0.076437       -0.261477  -0.283110
radius_se              0.679090      0.275869        0.691765   0.732562
texture_se            -0.097317      0.386358       -0.086761  -0.066280
perimeter_se           0.674172      0.281673        0.693135   0.726628
area_se                0.735864      0.259845        0.744983   0.800086
smoothness_se         -0.222600      0.006614       -0.202694  -0.166777
compactness_se         0.206000      0.191975        0.250744   0.212583
concavity_se           0.194204      0.143293        0.228082   0.207660
points_se              0.376169      0.163851        0.407217   0.372320
```

```
symmetry_se          -0.104321      0.009127        -0.081629  -0.072497
dimension_se         -0.042641      0.054458        -0.005523  -0.019887
radius_worst          0.969539      0.352573         0.969476   0.962746
texture_worst         0.297008      0.912045         0.303038   0.287489
perimeter_worst       0.965137      0.358040         0.970387   0.959120
area_worst            0.941082      0.343546         0.941550   0.959213
smoothness_worst      0.119616      0.077503         0.150549   0.123523
compactness_worst     0.413463      0.277830         0.455774   0.390410
concavity_worst       0.526911      0.301025         0.563879   0.512606
points_worst          0.744214      0.295316         0.771241   0.722017
symmetry_worst        0.163953      0.105008         0.189115   0.143570
dimension_worst       0.007066      0.119205         0.051019   0.003738

                   smoothness_mean  compactness_mean  concavity_mean  \
radius_mean               0.170581          0.506124        0.676764
texture_mean             -0.023389          0.236702        0.302418
perimeter_mean            0.207278          0.556936        0.716136
area_mean                 0.177028          0.498502        0.685983
smoothness_mean           1.000000          0.659123        0.521984
compactness_mean          0.659123          1.000000        0.883121
concavity_mean            0.521984          0.883121        1.000000
points_mean               0.553695          0.831135        0.921391
symmetry_mean             0.557775          0.602641        0.500667
dimension_mean            0.584792          0.565369        0.336783
radius_se                 0.301467          0.497473        0.631925
texture_se                0.068406          0.046205        0.076218
perimeter_se              0.296092          0.548905        0.660391
area_se                   0.246552          0.455653        0.617427
smoothness_se             0.332375          0.135299        0.098564
compactness_se            0.318943          0.738722        0.670279
concavity_se              0.248396          0.570517        0.691270
points_se                 0.380676          0.642262        0.683260
symmetry_se               0.200774          0.229977        0.178009
dimension_se              0.283607          0.507318        0.449301
radius_worst              0.213120          0.535315        0.688236
texture_worst             0.036072          0.248133        0.299879
perimeter_worst           0.238853          0.590210        0.729565
area_worst                0.206718          0.509604        0.675987
smoothness_worst          0.805324          0.565541        0.448822
compactness_worst         0.472468          0.865809        0.754968
concavity_worst           0.434926          0.816275        0.884103
points_worst              0.503053          0.815573        0.861323
symmetry_worst            0.394309          0.510223        0.409464
dimension_worst           0.499316          0.687382        0.514930

                   points_mean  symmetry_mean  dimension_mean  ...  \
radius_mean           0.822529       0.147741       -0.311631  ...
texture_mean          0.293464       0.071401       -0.076437  ...
perimeter_mean        0.850977       0.183027       -0.261477  ...
area_mean             0.823269       0.151293       -0.283110  ...
smoothness_mean       0.553695       0.557775        0.584792  ...
compactness_mean      0.831135       0.602641        0.565369  ...
concavity_mean        0.921391       0.500667        0.336783  ...
```

```
points_mean          1.000000      0.462497        0.166917  ...
symmetry_mean        0.462497      1.000000        0.479921  ...
dimension_mean       0.166917      0.479921        1.000000  ...
radius_se            0.698050      0.303379        0.000111  ...
texture_se           0.021480      0.128053        0.164174  ...
perimeter_se         0.710650      0.313893        0.039830  ...
area_se              0.690299      0.223970       -0.090170  ...
smoothness_se        0.027653      0.187321        0.401964  ...
compactness_se       0.490424      0.421659        0.559837  ...
concavity_se         0.439167      0.342627        0.446630  ...
points_se            0.615634      0.393298        0.341198  ...
symmetry_se          0.095351      0.449137        0.345007  ...
dimension_se         0.257584      0.331786        0.688132  ...
radius_worst         0.830318      0.185728       -0.253691  ...
texture_worst        0.292752      0.090651       -0.051269  ...
perimeter_worst      0.855923      0.219169       -0.205151  ...
area_worst           0.809630      0.177193       -0.231854  ...
smoothness_worst     0.452753      0.426675        0.504942  ...
compactness_worst    0.667454      0.473200        0.458798  ...
concavity_worst      0.752399      0.433721        0.346234  ...
points_worst         0.910155      0.430297        0.175325  ...
symmetry_worst       0.375744      0.699826        0.334019  ...
dimension_worst      0.368661      0.438413        0.767297  ...

                   radius_worst  texture_worst  perimeter_worst  area_worst  \
radius_mean            0.969539       0.297008         0.965137    0.941082
texture_mean           0.352573       0.912045         0.358040    0.343546
perimeter_mean         0.969476       0.303038         0.970387    0.941550
area_mean              0.962746       0.287489         0.959120    0.959213
smoothness_mean        0.213120       0.036072         0.238853    0.206718
compactness_mean       0.535315       0.248133         0.590210    0.509604
concavity_mean         0.688236       0.299879         0.729565    0.675987
points_mean            0.830318       0.292752         0.855923    0.809630
symmetry_mean          0.185728       0.090651         0.219169    0.177193
dimension_mean        -0.253691      -0.051269        -0.205151   -0.231854
radius_se              0.715065       0.194799         0.719684    0.751548
texture_se            -0.111690       0.409003        -0.102242   -0.083195
perimeter_se           0.697201       0.200371         0.721031    0.730713
area_se                0.757373       0.196497         0.761213    0.811408
smoothness_se         -0.230691      -0.074743        -0.217304   -0.182195
compactness_se         0.204607       0.143003         0.260516    0.199371
concavity_se           0.186904       0.100241         0.226680    0.188353
points_se              0.358127       0.086741         0.394999    0.342271
symmetry_se           -0.128121      -0.077473        -0.103753   -0.110343
dimension_se          -0.037488      -0.003195        -0.001000   -0.022736
radius_worst           1.000000       0.359921         0.993708    0.984015
texture_worst          0.359921       1.000000         0.365098    0.345842
perimeter_worst        0.993708       0.365098         1.000000    0.977578
area_worst             0.984015       0.345842         0.977578    1.000000
smoothness_worst       0.216574       0.225429         0.236775    0.209145
compactness_worst      0.475820       0.360832         0.529408    0.438296
concavity_worst        0.573975       0.368366         0.618344    0.543331
points_worst           0.787424       0.359755         0.816322    0.747419
```

symmetry_worst	0.243529	0.233027	0.269493	0.209146
dimension_worst	0.093492	0.219122	0.138957	0.079647

	smoothness_worst	compactness_worst	concavity_worst \
radius_mean	0.119616	0.413463	0.526911
texture_mean	0.077503	0.277830	0.301025
perimeter_mean	0.150549	0.455774	0.563879
area_mean	0.123523	0.390410	0.512606
smoothness_mean	0.805324	0.472468	0.434926
compactness_mean	0.565541	0.865809	0.816275
concavity_mean	0.448822	0.754968	0.884103
points_mean	0.452753	0.667454	0.752399
symmetry_mean	0.426675	0.473200	0.433721
dimension_mean	0.504942	0.458798	0.346234
radius_se	0.141919	0.287103	0.380585
texture_se	-0.073658	-0.092439	-0.068956
perimeter_se	0.130054	0.341919	0.418899
area_se	0.125389	0.283257	0.385100
smoothness_se	0.314457	-0.055558	-0.058298
compactness_se	0.227394	0.678780	0.639147
concavity_se	0.168481	0.484858	0.662564
points_se	0.215351	0.452888	0.549592
symmetry_se	-0.012662	0.060255	0.037119
dimension_se	0.170568	0.390159	0.379975
radius_worst	0.216574	0.475820	0.573975
texture_worst	0.225429	0.360832	0.368366
perimeter_worst	0.236775	0.529408	0.618344
area_worst	0.209145	0.438296	0.543331
smoothness_worst	1.000000	0.568187	0.518523
compactness_worst	0.568187	1.000000	0.892261
concavity_worst	0.518523	0.892261	1.000000
points_worst	0.547691	0.801080	0.855434
symmetry_worst	0.493838	0.614441	0.532520
dimension_worst	0.617624	0.810455	0.686511

	points_worst	symmetry_worst	dimension_worst
radius_mean	0.744214	0.163953	0.007066
texture_mean	0.295316	0.105008	0.119205
perimeter_mean	0.771241	0.189115	0.051019
area_mean	0.722017	0.143570	0.003738
smoothness_mean	0.503053	0.394309	0.499316
compactness_mean	0.815573	0.510223	0.687382
concavity_mean	0.861323	0.409464	0.514930
points_mean	0.910155	0.375744	0.368661
symmetry_mean	0.430297	0.699826	0.438413
dimension_mean	0.175325	0.334019	0.767297
radius_se	0.531062	0.094543	0.049559
texture_se	-0.119638	-0.128215	-0.045655
perimeter_se	0.554897	0.109930	0.085433
area_se	0.538166	0.074126	0.017539
smoothness_se	-0.102007	-0.107342	0.101480
compactness_se	0.483208	0.277878	0.590973
concavity_se	0.440472	0.197788	0.439329

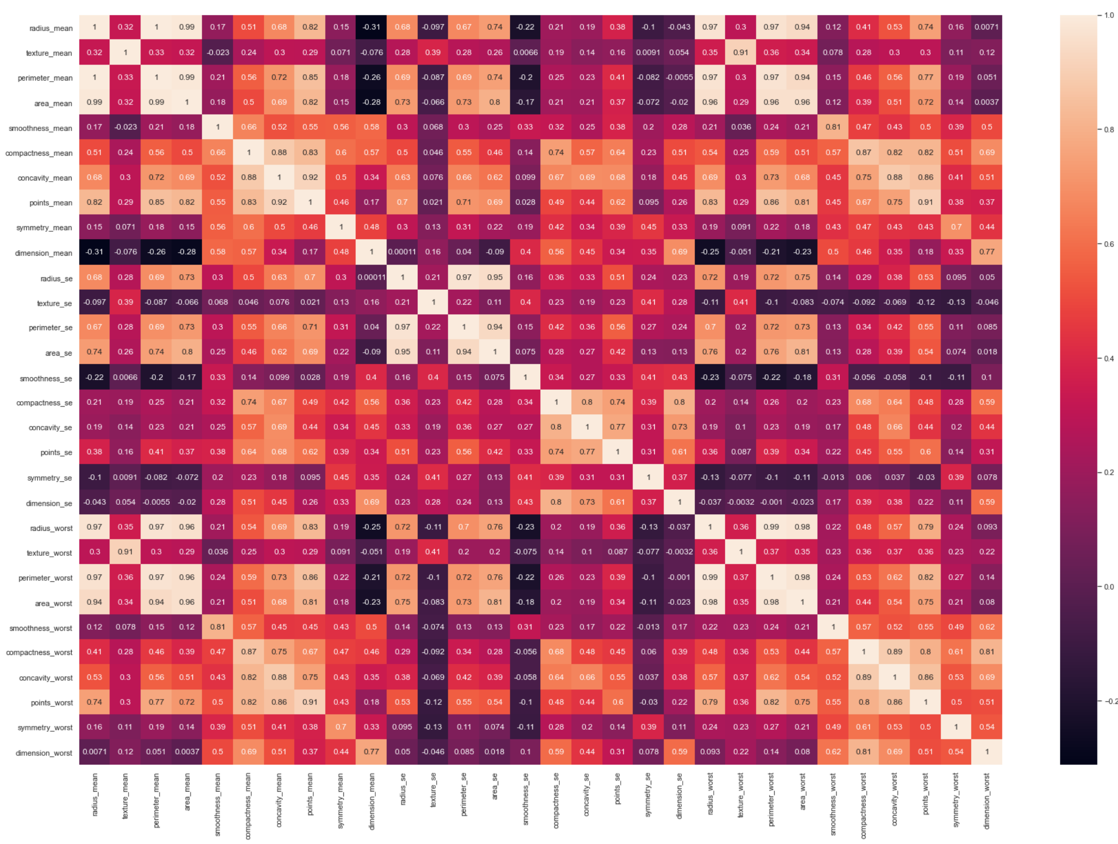
```
points_se              0.602450      0.143116      0.310655
symmetry_se           -0.030413      0.389402      0.078079
dimension_se           0.215204      0.111094      0.591328
radius_worst           0.787424      0.243529      0.093492
texture_worst          0.359755      0.233027      0.219122
perimeter_worst        0.816322      0.269493      0.138957
area_worst             0.747419      0.209146      0.079647
smoothness_worst       0.547691      0.493838      0.617624
compactness_worst      0.801080      0.614441      0.810455
concavity_worst        0.855434      0.532520      0.686511
points_worst           1.000000      0.502528      0.511114
symmetry_worst         0.502528      1.000000      0.537848
dimension_worst        0.511114      0.537848      1.000000

[30 rows x 30 columns]
```

Correlogram.

[12]:
```python
plt.figure(figsize=(30,20))
sns.heatmap(WBCD.iloc[:,2:32].corr(), annot = True)
```

[12]: <AxesSubplot:>



[13]:
```python
WBCD
```

[13]:
```
             id  diagnosis  radius_mean  texture_mean  perimeter_mean  \
0      87139402          0        12.32         12.39           78.85
1       8910251          0        10.60         18.95           69.28
2        905520          0        11.04         16.83           70.92
3        868871          0        11.28         13.39           73.00
4       9012568          0        15.19         13.21           97.65
..          ...        ...          ...           ...             ...
564   911320502          0        13.17         18.22           84.28
565      898677          0        10.26         14.71           66.20
566      873885          1        15.28         22.41           98.92
567      911201          0        14.53         13.98           93.86
568     9012795          1        21.37         15.10          141.30

     area_mean  smoothness_mean  compactness_mean  concavity_mean  \
0        464.1          0.10280           0.06981         0.03987
1        346.4          0.09688           0.11470         0.06387
2        373.2          0.10770           0.07804         0.03046
3        384.8          0.11640           0.11360         0.04635
4        711.8          0.07963           0.06934         0.03393
..         ...              ...               ...             ...
564      537.3          0.07466           0.05994         0.04859
565      321.6          0.09882           0.09159         0.03581
566      710.6          0.09057           0.10520         0.05375
567      644.2          0.10990           0.09242         0.06895
568     1386.0          0.10010           0.15150         0.19320

     points_mean  ...  radius_worst  texture_worst  perimeter_worst  \
0        0.03700  ...         13.50          15.64            86.97
1        0.02642  ...         11.88          22.94            78.28
2        0.02480  ...         12.41          26.44            79.93
3        0.04796  ...         11.92          15.77            76.53
4        0.02657  ...         16.20          15.73           104.50
..           ...  ...           ...            ...              ...
564      0.02870  ...         14.90          23.89            95.10
565      0.02037  ...         10.88          19.48            70.89
566      0.03263  ...         17.80          28.03           113.80
567      0.06495  ...         15.80          16.93           103.10
568      0.12550  ...         22.69          21.84           152.10

     area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0         549.1            0.1385             0.1266          0.12420
1         424.8            0.1213             0.2515          0.19160
2         471.4            0.1369             0.1482          0.10670
3         434.0            0.1367             0.1822          0.08669
4         819.1            0.1126             0.1737          0.13620
..          ...               ...                ...              ...
564       687.6            0.1282             0.1965          0.18760
565       357.1            0.1360             0.1636          0.07162
566       973.1            0.1301             0.3299          0.36300
567       749.9            0.1347             0.1478          0.13730
568      1535.0            0.1192             0.2840          0.40240

     points_worst  symmetry_worst  dimension_worst
```

```
0         0.09391          0.2827          0.06771
1         0.07926          0.2940          0.07587
2         0.07431          0.2998          0.07881
3         0.08611          0.2102          0.06784
4         0.08178          0.2487          0.06766
..            ...             ...              ...
564       0.10450          0.2235          0.06925
565       0.04074          0.2434          0.08488
566       0.12260          0.3175          0.09772
567       0.10690          0.2606          0.07810
568       0.19660          0.2730          0.08666

[569 rows x 32 columns]
```

II. Machine Learning

[14]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

a. Random Forest classifier

[15]:
```python
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,␣
 ↪classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt'],
}

# Create a RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

# Perform grid search with cross-validation
grid_search = GridSearchCV(rf_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Get the best hyperparameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'max_depth': 10, 'max_features': 'auto',
'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

[16]:
```
rf_classifier = RandomForestClassifier(max_depth = 10, max_features = 'auto',␣
 ↪min_samples_leaf = 1, min_samples_split = 2, n_estimators = 200)
```

[17]:
```
rf_classifier
```

[17]:
```
RandomForestClassifier(max_depth=10, n_estimators=200)
```

b. Gradient Boosting

[18]:
```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt'],
}

# Create a GradientBoostingClassifier
gb_classifier = GradientBoostingClassifier(random_state=42)

# Perform grid search with cross-validation
grid_search = GridSearchCV(gb_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Get the best hyperparameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'max_depth': 3, 'max_features':
'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}
```

[19]:
```
GBC = GradientBoostingClassifier(learning_rate = 0.1, max_depth =  3, max_features =␣
 ↪'sqrt', min_samples_leaf = 2, min_samples_split = 2, n_estimators = 200)
```

[20]:
```
GBC
```

[20]:
```
GradientBoostingClassifier(max_features='sqrt', min_samples_leaf=2,
                           n_estimators=200)
```

c. AdaBoost Classifier

[21]:
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV


# Define the parameter grid for hyperparameter tuning
```

```
param_grid = {
    'base_estimator': [DecisionTreeClassifier(max_depth=1),␣
↪DecisionTreeClassifier(max_depth=2)],
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 1.0],
}

# Create an AdaBoostClassifier
adaboost_classifier = AdaBoostClassifier(random_state=42)

# Perform grid search with cross-validation
grid_search = GridSearchCV(adaboost_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)

# Get the best hyperparameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'base_estimator': DecisionTreeClassifier(max_depth=1),
'learning_rate': 1.0, 'n_estimators': 200}
```

[22]:
```
AB = AdaBoostClassifier(base_estimator = DecisionTreeClassifier(max_depth = 1),␣
↪learning_rate = 1.0, n_estimators = 200)
```

[23]:
```
AB
```

[23]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   n_estimators=200)
```

d. Bagging Classifier

[24]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid for the base estimator
param_grid = {
    'base_estimator__max_depth': [None, 10, 20],
    'n_estimators': [50, 100, 200],
    'max_samples': [0.5, 0.7, 0.9],
}

# Create the base estimator with default hyperparameters
base_estimator = DecisionTreeClassifier(random_state=42)

# Create a Bagging Classifier
bagging_classifier = BaggingClassifier(base_estimator=base_estimator, random_state=42)

# Perform grid search with cross-validation
grid_search = GridSearchCV(bagging_classifier, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)
```

```
# Get the best hyperparameters and the best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

# Print the best hyperparameters
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'base_estimator__max_depth': None, 'max_samples': 0.9,
'n_estimators': 200}
```

[25]:
```
BC = BaggingClassifier(max_samples = 0.9, n_estimators = 200)
```

[26]:
```
BC
```

[26]:
```
BaggingClassifier(max_samples=0.9, n_estimators=200)
```

III. Evaluation Metrics.

[27]:
```
rf_classifier.fit(X_train, y_train)

# Make predictions on the test data
y_pred = rf_classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.2f}")

recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.2f}")

f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.2f}")

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Print a classification report (includes precision, recall, and F1 score)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.96
Precision: 0.94
Recall: 0.97
F1 Score: 0.95
Confusion Matrix:
[[106   4]
 [  2  59]]
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.98      0.96      0.97       110
           1       0.94      0.97      0.95        61

    accuracy                           0.96       171
   macro avg       0.96      0.97      0.96       171
weighted avg       0.97      0.96      0.97       171
```

[28]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,␣
 ↪classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create and train a classifier (Random Forest, for example)
GBC.fit(X_train, y_train)

# Make predictions on the test data
y_pred = GBC.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.2f}")

recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.2f}")

f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.2f}")

# Calculate and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Print a classification report (includes precision, recall, and F1 score)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.97
Precision: 0.95
Recall: 0.97
F1 Score: 0.96
Confusion Matrix:
[[72  2]
 [ 1 39]]
Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.99      0.97      0.98        74
           1       0.95      0.97      0.96        40

    accuracy                           0.97       114
   macro avg       0.97      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114
```

[29]:

```python
AB.fit(X_train, y_train)

# Make predictions on the test data
y_pred = AB.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.2f}")

recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.2f}")

f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.2f}")

# Calculate and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Print a classification report (includes precision, recall, and F1 score)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.99
Precision: 1.00
Recall: 0.97
F1 Score: 0.99
Confusion Matrix:
[[74  0]
 [ 1 39]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        74
           1       1.00      0.97      0.99        40

    accuracy                           0.99       114
   macro avg       0.99      0.99      0.99       114
weighted avg       0.99      0.99      0.99       114
```

[30]:
```
BC.fit(X_train, y_train)

# Make predictions on the test data
y_pred = BC.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.2f}")

recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.2f}")

f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.2f}")

# Calculate and print the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Print a classification report (includes precision, recall, and F1 score)
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.96
Precision: 0.93
Recall: 0.97
F1 Score: 0.95
Confusion Matrix:
[[71  3]
 [ 1 39]]
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.97        74
           1       0.93      0.97      0.95        40

    accuracy                           0.96       114
   macro avg       0.96      0.97      0.96       114
weighted avg       0.97      0.96      0.97       114
```

[31]:
```
import pandas as pd
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,␣
↪AdaBoostClassifier, BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Create and train different classifiers
classifiers = {
    'Random Forest': rf_classifier,
    'Gradient Boosting': GBC,
    'AdaBoost': AB,
    'Bagging': BC
}

metrics = []

# Calculate and consolidate metrics for each classifier
for name, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    metrics.append({
        'Classifier': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1
    })

# Create a DataFrame to consolidate the metrics
metrics_df = pd.DataFrame(metrics)

# Print the consolidated metrics
print("Consolidated Metrics:")
print(metrics_df)
```

```
Consolidated Metrics:
          Classifier  Accuracy  Precision  Recall  F1 Score
0      Random Forest  0.973684   0.951220   0.975  0.962963
1  Gradient Boosting  0.964912   0.928571   0.975  0.951220
2           AdaBoost  0.991228   1.000000   0.975  0.987342
3            Bagging  0.973684   0.951220   0.975  0.962963
```