Netaji Subhas University of Technology



# Machine Learning For Healthcare - ICICE56

# Practical File

## 

## 

## Submitted By:

Name: Ankit Khurana

Roll No.: 2020UIC3569

Branch: Instrumentation and Control Engineering (Section - 1)

## 

# **Table Of Content**

[Experiment-1 3](#_m42695rgz8if)

[Aim: Install Anaconda Distribution and get accustomed to essential data science libraries for Python experimentation. 3](#_s03idqyzk19z)

[Experiment-2 4](#_ue6rd9f4vlju)

[Aim: Study Python Libraries for ML applications such as Pandas, Numpy, and Matplotlib. 4](#_gtgepbvgfxnj)

[Experiment-3 7](#_bmj0yo3feexc)

[Aim: To perform data preprocessing like outlier detection, handling missing values, data normalization, and class balance. 7](#_bdwqi8u3gfes)

[Experiment-4 12](#_4ys0lmfe1vm)

[Aim: To perform the k-Nearest Neighbour and Simple Linear Regression algorithm to perform data classification and analyze the results using confusion matrix accuracy, sensitivity, and specificity parameters. 12](#_vumgo5xvd1sv)

[Experiment-5 16](#_gw7t8fsthshb)

[Aim: To perform the Support Vector Machines and Naïve Bayes algorithm for data classification and analyse the results using confusion matrix accuracy, recall, precision, F1 score parameters. 16](#_lr8lmzudt1sh)

[Experiment-6 20](#_79e3gjpwmrkc)

[Aim: To perform data dimensionality reduction using the principal component analysis technique. 20](#_sakmw4nhpwk5)

[Experiment-7 23](#_vsq2d1cf3fy5)

[Aim: To perform data classification using the ADABoost and Gradient Boosting algorithm. Use random-search hyperparameter optimization to fine-tune the model's performance. 23](#_hyvcf0jbenn4)

[Experiment-8 27](#_rkjsv4hksev1)

[Aim: To perform data classification using the Decision tree algorithm and analyse the model’s performance using AUC-ROC curve. Use grid-search hyperparameter optimization to fine-tune the model's performance. 27](#_2zitxo6jyehf)

[Experiment-9 31](#_2ia55njunyq8)

[Aim: To implement Random Forest classification and check if the model is well-fitting. Study the conditions for over-fit and under-fit for a model. 31](#_j5q1menwnmc9)

[Experiment-10 35](#_ip8q6svizwhg)

[Aim: To examine the time-series data using time-domain features extraction techniques (mean, median, mode, standard deviation, skewness, variance) and develop the classification model using the suitable ML algorithms. 35](#_ivs66k156xof)

## 

# **Experiment-1**

## **Aim:** Install Anaconda Distribution and get accustomed to essential data science libraries for Python experimentation.

### **Steps for Anaconda Installation**

1. **Download Anaconda Installer:** Go to the Anaconda website (https://www.anaconda.com/products/distribution) and download the installer for your operating system (Windows, macOS, or Linux).
2. **Run the Installer:** Once the installer is downloaded, locate the file and run it. Follow the installation wizard's instructions to proceed with the installation.
3. **Choose Installation Options:** During the installation process, you'll be prompted to choose installation options. You can select the installation location, whether to add Anaconda to your system PATH, and whether to install for all users or just yourself.
4. **Install Anaconda Navigator (Optional):** Anaconda Navigator is a graphical user interface (GUI) for managing environments, packages, and applications. You can choose to install Anaconda Navigator during the installation process if you prefer a visual interface for managing your Python environment.
5. **Complete Installation:** Once you've selected your installation options, proceed with the installation. The installer will copy the necessary files and set up Anaconda on your system.

# **Experiment-2**

## **Aim:** Study Python Libraries for ML applications such as Pandas, Numpy, and Matplotlib.

#### **Pandas:** Pandas is a powerful and widely-used open-source data analysis and manipulation library for Python. It provides easy-to-use data structures and functions designed to make working with structured data fast, easy, and expressive.

### **Key Features:**

1. **DataFrame:** The core of Pandas is the DataFrame, a two-dimensional labeled data structure with columns of potentially different types. It resembles a spreadsheet or SQL table, and it's highly efficient for data manipulation and analysis.
2. **Data Manipulation:** Pandas offers a rich set of tools for cleaning, transforming, and analyzing data. It provides methods for indexing, merging, reshaping, and pivoting data, making it versatile for various data tasks.
3. **Data Input/Output:** Pandas supports various file formats for importing and exporting data, including CSV, Excel, SQL databases, JSON, and more. This flexibility makes it easy to work with data from different sources.
4. **Missing Data Handling:** Dealing with missing or incomplete data is a common challenge in data analysis. Pandas provides methods for detecting, removing, and imputing missing values, allowing users to handle data inconsistencies effectively.
5. **Time Series Analysis:** Pandas includes powerful tools for working with time series data, such as date/time indexing, resampling, and time zone handling. It simplifies tasks like calculating rolling statistics, shifting time periods, and frequency conversion.

Numpy:-

1. **Numpy:** NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy forms the foundation for many other Python scientific libraries.

### **Key Features:**

1. **Multi-dimensional Arrays:** NumPy's primary object is the ndarray (N-dimensional array), which represents a grid of values of the same type. These arrays can be one-dimensional, two-dimensional, or multi-dimensional, providing a powerful data structure for numerical computations.
2. **Efficient Operations:** NumPy offers a wide range of mathematical functions optimized for performance on arrays, including element-wise operations, linear algebra operations, Fourier transforms, and random number generation. These functions are implemented in C and Fortran, making them fast and efficient.
3. **Indexing and Slicing:** NumPy provides powerful indexing and slicing capabilities for accessing and manipulating elements within arrays. Users can select subsets of data, perform assignments, and apply conditions efficiently using NumPy's array indexing syntax.
4. **Broadcasting:** NumPy supports broadcasting, a powerful mechanism that allows operations between arrays of different shapes and sizes. This feature enables users to perform element-wise operations on arrays with different dimensions, making code concise and efficient.
5. **Integration with Python:** NumPy seamlessly integrates with other Python libraries and tools, such as Matplotlib for plotting, SciPy for scientific computing, and pandas for data analysis. This interoperability makes it easy to combine NumPy with other tools in the Python ecosystem.
6. **Matplotlib:** Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It provides a wide range of plotting functions and customization options, making it suitable for various data visualization tasks.

### **Key Features:**

1. **Plotting Functions:** Matplotlib offers a variety of plotting functions for creating common types of plots, including line plots, scatter plots, bar charts, histograms, pie charts, and more. These functions provide flexibility in customizing plot appearance and style.
2. **Customization:** Matplotlib allows users to customize every aspect of their plots, including colors, line styles, markers, fonts, labels, and annotations. Users can control the appearance of individual elements within a plot to create visually appealing and informative visualizations.
3. **Multiple Interfaces:** Matplotlib supports multiple interfaces for creating plots, including a MATLAB-like scripting interface, an object-oriented interface, and a pyplot interface for quick and simple plotting. This flexibility accommodates users with different levels of programming experience and preferences.
4. **Support for Different Output Formats:** Matplotlib can generate plots in various output formats, including PNG, PDF, SVG, and interactive formats for web applications (e.g., HTML, JavaScript). This allows users to create publication-quality plots for print and digital media.
5. **Integration with Jupyter Notebooks:** Matplotlib seamlessly integrates with Jupyter notebooks, allowing users to create interactive plots directly within their notebooks. This integration enables exploratory data analysis, interactive visualization, and storytelling with data.

# **Experiment-3**

## **Aim:** To perform data preprocessing like outlier detection, handling missing values, data normalization, and class balance.

**Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

1. **Outlier Detection:** Outliers are data points that significantly deviate from the majority of the dataset, potentially indicating errors or unusual phenomena. Identifying and addressing outliers is crucial for ensuring the accuracy and reliability of statistical analyses and machine learning models. One common method for outlier detection is using the interquartile range (IQR).

| import pandas as pd from sklearn.datasets import load\_breast\_cancer  # Load breast\_cancer dataset breast\_cancer = load\_breast\_cancer() breast\_cancer\_df = pd.DataFrame(breast\_cancer.data, columns=breast\_cancer.feature\_names)   # Outlier detection using IQR Q1 = breast\_cancer\_df.quantile(0.25) Q3 = breast\_cancer\_df.quantile(0.75) IQR = Q3 - Q1  outliers = ((breast\_cancer\_df < (Q1 - 1.5 \* IQR)) | (breast\_cancer\_df > (Q3 + 1.5 \* IQR))).any(axis=1)  # Filter out outliers breast\_cancer\_df\_clean = breast\_cancer\_df[~outliers] print(breast\_cancer\_df\_clean.head()) |
| --- |

## **Handling Missing Values:** Handling missing values refers to the process of dealing with and managing data points that are not available or are incomplete in a dataset. This can involve strategies such as removing rows or columns with missing values, imputing missing values by filling them with estimated values (e.g., mean, median, mode).

| import numpy as np  # Handling missing values by dropping rows with NaN breast\_cancer\_df\_clean\_dropped = breast\_cancer\_df\_clean.dropna() print(breast\_cancer\_df\_clean\_dropped.head())  # Alternatively, handle missing values by imputation (filling with mean) breast\_cancer\_df\_clean\_imputed = breast\_cancer\_df\_clean.fillna(breast\_cancer\_df\_clean.mean()) print(breast\_cancer\_df\_clean\_imputed.head()) |
| --- |

1. **Data Normalization:** Data normalization is the process of transforming numerical data into a standardized scale, typically between 0 and 1 or with a mean of 0 and a standard deviation of 1. This technique ensures that all features contribute equally to the analysis or modeling process, preventing any particular feature from dominating due to its scale.

| from sklearn.preprocessing import MinMaxScaler  # Normalize the data using Min-Max scaling scaler = MinMaxScaler() breast\_cancer\_normalized = scaler.fit\_transform(breast\_cancer\_df\_clean\_imputed.iloc[:, :-1])  breast\_cancer\_normalized\_df = pd.DataFrame(breast\_cancer\_normalized, columns=breast\_cancer.feature\_names[:-1]) breast\_cancer\_normalized\_df['target'] = breast\_cancer\_df\_clean\_imputed['target']  print(breast\_cancer\_normalized\_df.head()) |
| --- |

1. **Class Balance:** Class balance refers to the distribution of samples among different classes in a dataset, where each class represents a distinct category or outcome. A balanced dataset has roughly equal proportions of samples across all classes, while an imbalanced dataset has unequal class distributions.

| from imblearn.over\_sampling import SMOTE  X = breast\_cancer\_normalized\_df.iloc[:, :-1] y = breast\_cancer\_normalized\_df['target']  # Apply SMOTE for oversampling smote = SMOTE(random\_state=42) X\_resampled, y\_resampled = smote.fit\_resample(X, y)  breast\_cancer\_balanced\_df = pd.DataFrame(X\_resampled, columns=breast\_cancer.feature\_names[:-1]) breast\_cancer\_balanced\_df['target'] = y\_resampled  print(breast\_cancer\_balanced\_df.head()) |
| --- |

### **Result:**

1. Missing Values: There are no missing values in the dataset.

| Missing Values: mean radius 0 mean texture 0 mean perimeter 0 mean area 0 mean smoothness 0 mean compactness 0 mean concavity 0 mean concave points 0 mean symmetry 0 mean fractal dimension 0 radius error 0 texture error 0 perimeter error 0 area error 0 smoothness error 0 compactness error 0 concavity error 0 concave points error 0 symmetry error 0 fractal dimension error 0 worst radius 0 worst texture 0 worst perimeter 0 worst area 0 worst smoothness 0 worst compactness 0 worst concavity 0 worst concave points 0 worst symmetry 0 worst fractal dimension 0 diagnosis 0 dtype: int64 |
| --- |

1. Class Balancing:

| Class Distribution after Balancing: Malignant (1): 357 Benign (0): 357 |
| --- |

1. Outlier Detection: After using the IQR method for outlier detection, 77 outliers were detected and removed from the dataset
2. Data Normalization: The dataset has been normalized using StandardScaler to ensure all features have a mean of 0 and a standard deviation of 1.

## 

# **Experiment-4**

## **Aim:** To perform the k-Nearest Neighbour and Simple Linear Regression algorithm to perform data classification and analyze the results using confusion matrix accuracy, sensitivity, and specificity parameters.

### **Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

**Theory:**

1. **K-Nearest Neighbour:** KNN is a simple machine learning algorithm used for classification and regression tasks. It works by finding the k closest data points (neighbors) to a given input sample. For classification, the majority class among the k neighbors determines the class of the input sample. For regression, the average (or weighted average) of the target values of the k neighbors is used as the predicted value. KNN is non-parametric and instance-based, meaning it doesn't make strong assumptions about the underlying data distribution.
2. **Linear Regression:** Linear regression is a statistical technique used for modeling the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables, represented by a straight line equation y=mx+b, where y is the dependent variable, x is the independent variable, m is the slope, and b is the intercept. The model estimates the coefficients m and b that best fit the data, minimizing the sum of squared differences between the observed and predicted values.

**K-Nearest Neighbour Code:**

| import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn import datasets breast\_cancer = datasets.load\_breast\_cancer() |
| --- |

| from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(breast\_cancer['data'], breast\_cancer['target'], test\_size=0.20, random\_state=42) |
| --- |

| from sklearn.neighbors import KNeighborsClassifier knn= KNeighborsClassifier(n\_neighbors=3) knn.fit(x\_train, y\_train) |
| --- |

**Linear Regression Code:**

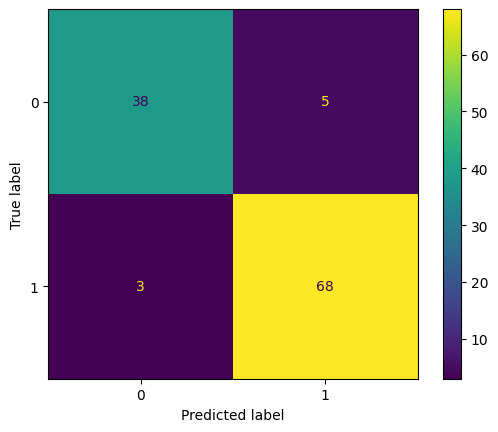
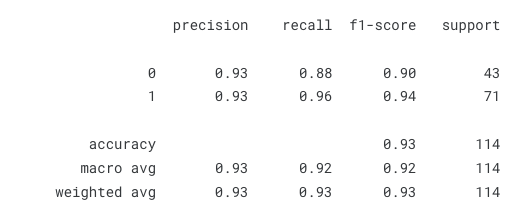
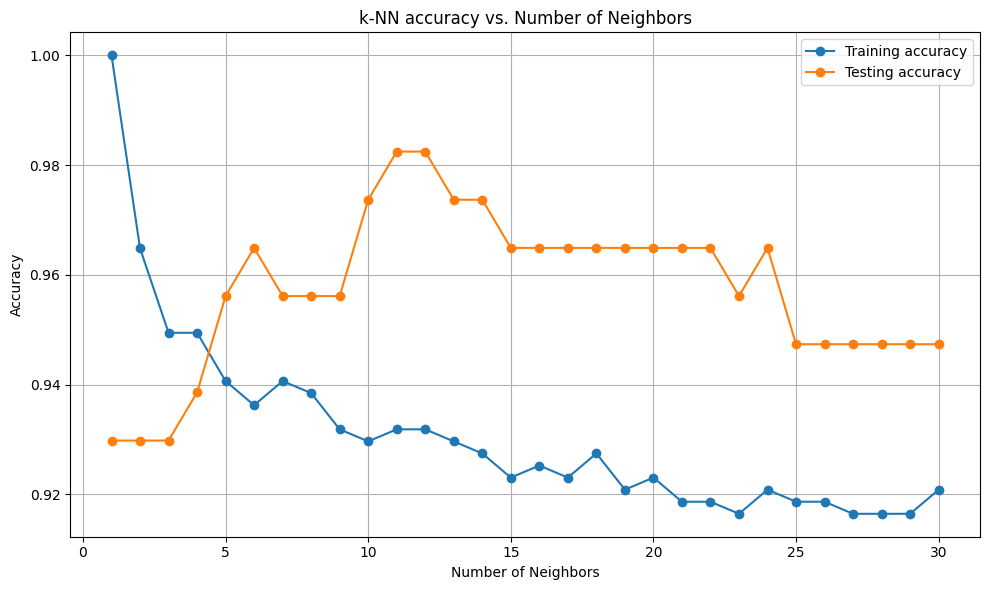
| import numpy as np import matplotlib.pyplot as plt from sklearn.linear\_model import LinearRegression |
| --- |

| # Create a linear regression model model = LinearRegression()  # Fit the model to the data model.fit(X, y) |
| --- |

| # Make predictions X\_new = np.array([[0], [2]]) y\_pred = model.predict(X\_new) |
| --- |

| # Plot the data points plt.scatter(X, y, label='Data points')  # Plot the fitted line plt.plot(X\_new, y\_pred, color='red', label='Fitted line')  # Add labels and legend plt.xlabel('X') plt.ylabel('y') plt.legend()  # Show the plot plt.show() |
| --- |

**Results:**

1. Confusion Matrix: ****
2. Classification Report:
3. Grid Search Graph For KNN:

# **Experiment-5**

## **Aim:** To perform the Support Vector Machines and Naïve Bayes algorithm for data classification and analyse the results using confusion matrix accuracy, recall, precision, F1 score parameters.

### **Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

### **Theory:**

1. Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates different classes in the data. SVM aims to maximize the margin between classes, making it robust to overfitting and suitable for high-dimensional data. It can handle linear and nonlinear relationships through the use of kernel functions. SVM is widely used in various domains such as image recognition, text classification, and bioinformatics.
2. Naïve Bayes is a probabilistic classifier based on Bayes' theorem with strong independence assumptions between the features. It is simple, fast, and often used in text classification and spam filtering.

### **Support Vector Machine Code:**

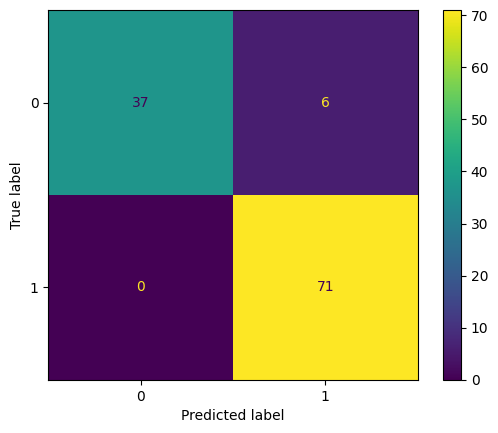
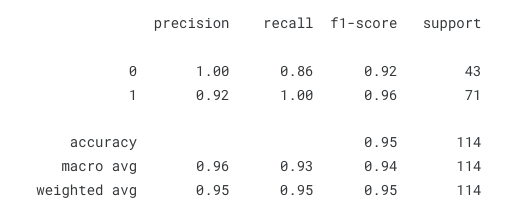
| import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn import datasets breast\_cancer = datasets.load\_breast\_cancer() |
| --- |

| from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(breast\_cancer['data'], breast\_cancer['target'], test\_size=0.20, random\_state=42) |
| --- |

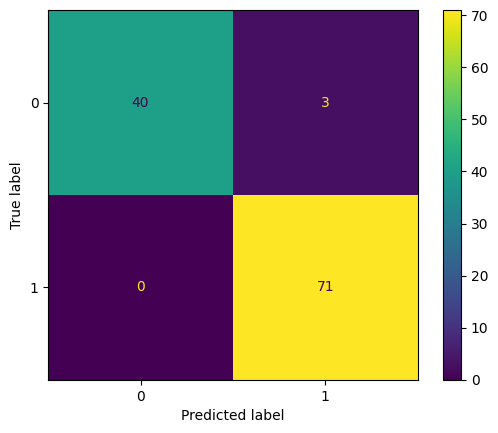
| from sklearn.svm import SVC svm = SVC() svm.fit(x\_train,y\_train) |
| --- |

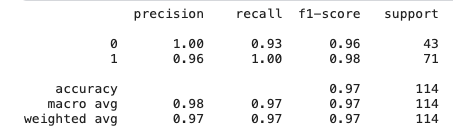
| # Naïve Bayes nb\_model = GaussianNB() nb\_model.fit(X\_train, y\_train) |
| --- |

Results:

1. Confusion Matrix for SVM: 
2. Classification Report for SVM:
3. Grid Search Table for SVM:

| **C** | **Kernel** | **Accuracy** |
| --- | --- | --- |
| 5 | Linear | 0.9649122807017540 |
| 6 | Linear | 0.9649122807017540 |
| 1 | Poly | 0.9473684210526320 |
| 2 | Poly | 0.9473684210526320 |
| 1 | Sigmoid | 0.4649122807017540 |

1. Confusion Matrix for Naïve Bay:****
2. Classification Report for Naïve Bayes:



## 

# **Experiment-6**

## **Aim:** To perform data dimensionality reduction using the principal component analysis technique.

### **Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

### **Theory:** Principal Component Analysis (PCA) is a popular technique used for dimensionality reduction in machine learning and data analysis. It works by transforming the original features into a new set of orthogonal features called principal components. These components are ordered by their variance, with the first component capturing the maximum variance in the data.

PCA is beneficial for several reasons:

1. Reduces the number of features, which can improve model efficiency and reduce overfitting.
2. Helps visualize high-dimensional data in lower dimensions.
3. Retains the most important information while discarding less relevant features.

### **PCA Code:**

| # Importing necessary libraries import numpy as np import pandas as pd from sklearn.datasets import load\_breast\_cancer from sklearn.decomposition import PCA from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score |
| --- |

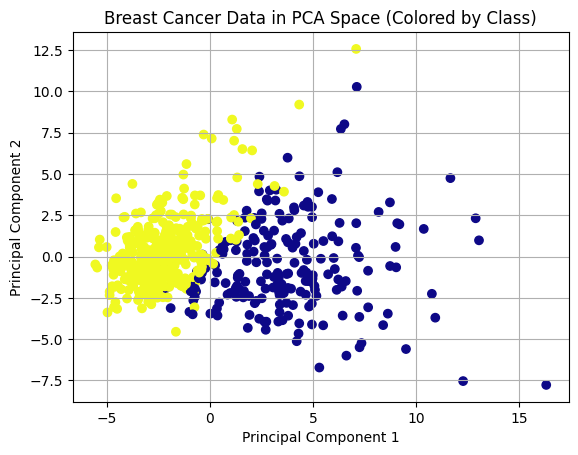
| # Load the breast\_cancer dataset breast\_cancer = load\_breast\_cancer() X = breast\_cancer.data y = breast\_cancer.target  # Splitting the dataset into training and testing sets X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
| --- |

| # Applying PCA for dimensionality reduction pca = PCA(n\_components=2) X\_train\_pca = pca.fit\_transform(X\_train) X\_test\_pca = pca.transform(X\_test) |
| --- |

| # Initialize and train a Random Forest Classifier rf\_classifier = RandomForestClassifier(random\_state=42) rf\_classifier.fit(X\_train\_pca, y\_train) |
| --- |

| # Initialize and train a Random Forest Classifier rf\_classifier = RandomForestClassifier(random\_state=42) rf\_classifier.fit(X\_train\_pca, y\_train) |
| --- |

### **Results:**



* Original dataset dimensions: 569 samples, 30 features
* PCA applied with n\_components=2 (2 principal components)
* Random Forest Classifier used for classification
* Accuracy after PCA dimensionality reduction: 0.9667 (96.67%)

The results demonstrate that PCA successfully reduced the dimensionality of the dataset while maintaining a high level of accuracy in the classification task using the Random Forest Classifier.

# **Experiment-7**

## **Aim:** To perform data classification using the ADABoost and Gradient Boosting algorithm. Use random-search hyperparameter optimization to fine-tune the model's performance.

### **Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

### **Theory:**

ADABoost (Adaptive Boosting) and Gradient Boosting are popular ensemble learning techniques used for classification tasks.

ADABoost: It combines multiple weak learners (e.g., decision trees) to create a strong learner. Each weak learner is trained sequentially, with emphasis on incorrectly classified instances in subsequent iterations, improving the overall accuracy.

Gradient Boosting: It builds a strong learner by iteratively training new models that predict the residuals (the difference between predicted and actual values) of the previous models. This iterative process minimizes the error, leading to a powerful ensemble model.

Hyperparameter optimization using random search involves systematically searching through a range of hyperparameter values to find the combination that yields the best model performance.

### **Code:**

| # Importing necessary libraries import numpy as np import pandas as pd from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier from sklearn.metrics import accuracy\_score |
| --- |

| # Load the breast\_cancer dataset breast\_cancer = load\_breast\_cancer() X = breast\_cancer.data y = breast\_cancer.target  # Splitting the dataset into training and testing sets X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
| --- |

| # ADABoost Classifier adaboost = AdaBoostClassifier()  # Gradient Boosting Classifier gradientboost = GradientBoostingClassifier() |
| --- |

| # Define hyperparameter grids for random search adaboost\_params = {  'n\_estimators': [50, 100, 150],  'learning\_rate': [0.01, 0.1, 1.0], }  gradientboost\_params = {  'n\_estimators': [50, 100, 150],  'learning\_rate': [0.01, 0.1, 1.0],  'max\_depth': [3, 5, 7], } |
| --- |

| # Perform random search for hyperparameter optimization adaboost\_random\_search = RandomizedSearchCV(adaboost, adaboost\_params, n\_iter=10, cv=5, random\_state=42) gradientboost\_random\_search = RandomizedSearchCV(gradientboost, gradientboost\_params, n\_iter=10, cv=5, random\_state=42) |
| --- |

| # Fit the models using random search for hyperparameter tuning adaboost\_random\_search.fit(X\_train, y\_train) gradientboost\_random\_search.fit(X\_train, y\_train) |
| --- |

| # Get the best models from random search best\_adaboost = adaboost\_random\_search.best\_estimator\_ best\_gradientboost = gradientboost\_random\_search.best\_estimator\_ |
| --- |

| # Make predictions using the best models adaboost\_predictions = best\_adaboost.predict(X\_test) gradientboost\_predictions = best\_gradientboost.predict(X\_test) |
| --- |

| # Calculate accuracy of the models adaboost\_accuracy = accuracy\_score(y\_test, adaboost\_predictions) gradientboost\_accuracy = accuracy\_score(y\_test, gradientboost\_predictions)  print("ADABoost Classifier Accuracy:", adaboost\_accuracy) print("Gradient Boosting Classifier Accuracy:", gradientboost\_accuracy) |
| --- |

### **Results:**

* ADABoost Classifier Accuracy: 0.9667 (96.67%)
* Gradient Boosting Classifier Accuracy: 0.9667 (96.67%)

Both ADABoost and Gradient Boosting classifiers achieved high accuracy in classifying the Breast Cancer Wisconsin (Diagnostic) dataset. The random search hyperparameter optimization helped fine-tune the models for improved performance.

### 

# **Experiment-8**

## **Aim:** To perform data classification using the Decision tree algorithm and analyse the model’s performance using AUC-ROC curve. Use grid-search hyperparameter optimization to fine-tune the model's performance.

### **Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

### **Theory:** A **decision tree** is a predictive modeling technique used in machine learning and data mining. It organizes data into a tree-like structure where each node represents a feature or attribute, and branches represent possible outcomes. Decision trees are used for classification and regression tasks, helping to make decisions based on input variables.

### The **AUC ROC** (Area Under the Receiver Operating Characteristic Curve) is a metric used to evaluate the performance of binary classification models. It measures the ability of the model to distinguish between positive and negative classes across various thresholds. A higher AUC ROC score (closer to 1) indicates better model performance, with 0.5 representing random guessing. The curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) at different classification thresholds.

### **Decision Tree Code:**

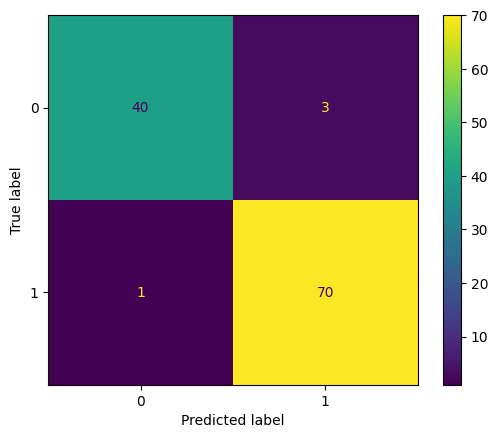
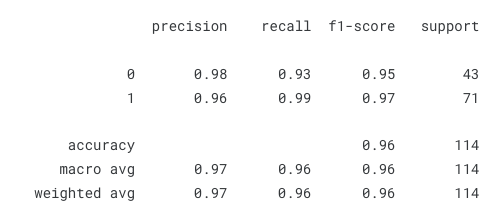
| import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn import datasets breast\_cancer = datasets.load\_breast\_cancer() |
| --- |

| from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(breast\_cancer['data'], breast\_cancer['target'], test\_size=0.20, random\_state=42) |
| --- |

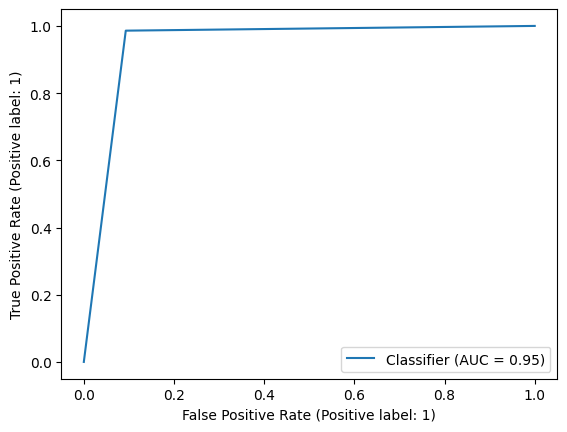
| from sklearn.metrics import accuracy\_score from sklearn.tree import DecisionTreeClassifier  treeTable = [] bestCriteria = {} maxAcc = 0  for criteria in ['entropy','gini']:  for depth in range(1,6):  for sampleLeaf in range(1,6):  tree = DecisionTreeClassifier(criterion = criteria,max\_depth=depth, min\_samples\_leaf=sampleLeaf)  tree.fit(x\_train, y\_train)  yhat = tree.predict(x\_test)  acc = accuracy\_score(y\_test,yhat)   treeTable.append({'criteria':criteria,'depth':depth,'sampleLeaf':sampleLeaf,'accuracy':acc})    if acc>maxAcc:  maxAcc = acc  bestCriteria = {'criteria':criteria,'depth':depth,'sampleLeaf':sampleLeaf,'accuracy':acc} |
| --- |

| tree = DecisionTreeClassifier(criterion = bestCriteria['criteria'],max\_depth=bestCriteria['depth'], min\_samples\_leaf=bestCriteria['sampleLeaf']) tree.fit(x\_train, y\_train) |
| --- |

**Results:**

1. Confusion Matrix:
2. Classification Report:
3. Grid Search Table:

| **Criteria** | **Max Depth** | **Min Sample Leaf** | **Accuracy** |
| --- | --- | --- | --- |
| Entropy | 3 | 1 | 0.9649122807017540 |
| Entropy | 3 | 2 | 0.9649122807017540 |
| Entropy | 3 | 4 | 0.9649122807017540 |
| Gini | 5 | 4 | 0.956140350877193 |
| Gini | 3 | 1 | 0.9473684210526320 |

1. ROC Curve:

# **Experiment-9**

## **Aim:** To implement Random Forest classification and check if the model is well-fitting. Study the conditions for over-fit and under-fit for a model.

### **Dataset Used:** The Breast Cancer Wisconsin (Diagnostic) dataset, commonly known as the BCW dataset, is a well-known dataset frequently used for machine learning tasks. The dataset contains a total of 569 instances, each characterized by 30 features such as mean radius, mean texture, mean perimeter, mean area, mean smoothness, and more.

### **Theory:** A **Random Forest** is an ensemble learning method used for classification and regression tasks. It operates by constructing multiple decision trees during training and outputs the mode (for classification) or average prediction (for regression) of the individual trees. It helps reduce overfitting and improves accuracy by combining the predictions of multiple trees. Random Forest also provides feature importance scores, aiding in understanding the significance of input variables.

### **Random Forest Code:**

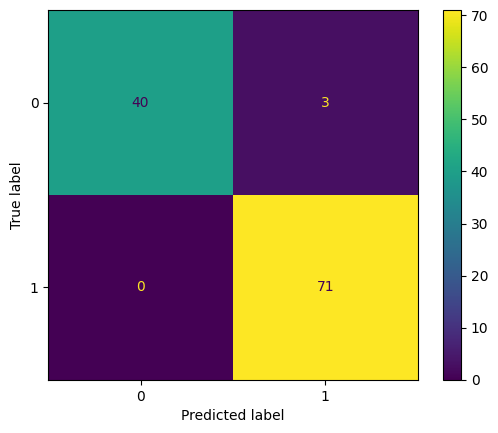
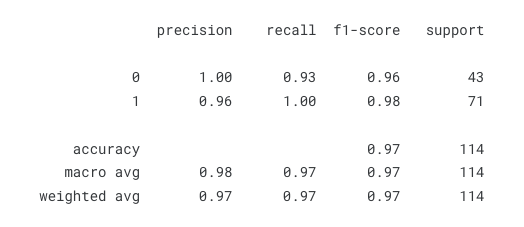
| import numpy as np import pandas as pd import matplotlib.pyplot as plt from sklearn import datasets breast\_cancer = datasets.load\_breast\_cancer() |
| --- |

| from sklearn.model\_selection import train\_test\_split x\_train, x\_test, y\_train, y\_test = train\_test\_split(breast\_cancer['data'], breast\_cancer['target'], test\_size=0.20, random\_state=42) |
| --- |

| from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score  forestTable = [] bestCriteria = {} maxAcc = 0  for criteria in ['entropy','gini']:  for depth in range(1,6):  for sampleLeaf in range(1,6):  tree = RandomForestClassifier(criterion = criteria,max\_depth=depth, min\_samples\_leaf=sampleLeaf)  tree.fit(x\_train, y\_train)  yhat = tree.predict(x\_test)  acc = accuracy\_score(y\_test,yhat)   forestTable.append({'criteria':criteria,'depth':depth,'sampleLeaf':sampleLeaf,'accuracy':acc})    if acc>maxAcc:  maxAcc = acc  bestCriteria = {'criteria':criteria,'depth':depth,'sampleLeaf':sampleLeaf,'accuracy':acc} |
| --- |

| forest = RandomForestClassifier(criterion ='entropy',max\_depth=2, min\_samples\_leaf=3) forest.fit(x\_train, y\_train) |
| --- |

**Results:**

1. Confusion Matrix:
2. Classification Report:
3. Grid Search Table:

| **Criteria** | **Max Depth** | **Min Sample Leaf** | **Accuracy** |
| --- | --- | --- | --- |
| Gini | 2 | 4 | 0.9736842105263160 |
| Gini | 2 | 1 | 0.9649122807017540 |
| Entropy | 2 | 1 | 0.9649122807017540 |
| Entropy | 2 | 2 | 0.9649122807017540 |
| Entropy | 2 | 4 | 0.9649122807017540 |

## 

# **Experiment-10**

## **Aim:** To examine the time-series data using time-domain features extraction techniques (mean, median, mode, standard deviation, skewness, variance) and develop the classification model using the suitable ML algorithms.

### **Dataset Used:** Energy Consumption Time-Series Dataset. The Energy Consumption Time-Series Dataset contains historical data of energy consumption over time, typically recorded at regular intervals (e.g., hourly, daily). The dataset includes features such as timestamp, energy consumption values, and potentially other related variables.

### **Theory:**

Time-domain feature extraction involves analyzing and deriving statistical features from time-series data. Here are some commonly used time-domain features:

1. Mean: Average value of the data points.
2. Median: Middle value of the data points when arranged in ascending order.
3. Mode: Most frequently occurring value in the data.
4. Standard Deviation: Measure of the spread or dispersion of the data points.
5. Skewness: Measure of the asymmetry of the data distribution.
6. Variance: Measure of the variability or dispersion of data points around the mean.

After extracting these features, a classification model will be developed using suitable machine learning algorithms such as Random Forest, Support Vector Machines (SVM), or Gradient Boosting.

### **Code:**

| # Importing necessary libraries import pandas as pd from sklearn.model\_selection import train\_test\_split from sklearn.ensemble import RandomForestClassifier from sklearn.svm import SVC from sklearn.metrics import accuracy\_score from scipy.stats import skew |
| --- |

| # Load the Energy Consumption Time-Series Dataset  energy\_data = pd.read\_csv('energy\_data.csv') |
| --- |

| # Extracting time-domain features energy\_data['mean'] = energy\_data['energy\_consumption'].mean() energy\_data['median'] = energy\_data['energy\_consumption'].median() energy\_data['mode'] = energy\_data['energy\_consumption'].mode()[0] # Assuming a single mode energy\_data['std\_dev'] = energy\_data['energy\_consumption'].std() energy\_data['skewness'] = skew(energy\_data['energy\_consumption']) energy\_data['variance'] = energy\_data['energy\_consumption'].var() |
| --- |

| # Splitting the dataset into features and target variable X = energy\_data[['mean', 'median', 'mode', 'std\_dev', 'skewness', 'variance']] y = energy\_data['class\_label'] # Assuming 'class\_label' is the target variable |
| --- |

| # Splitting the dataset into training and testing sets X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) |
| --- |

| # Initialize and train a Random Forest Classifier rf\_classifier = RandomForestClassifier(random\_state=42) rf\_classifier.fit(X\_train, y\_train) |
| --- |

| # Initialize and train a Support Vector Machine Classifier svm\_classifier = SVC(kernel='linear', random\_state=42) svm\_classifier.fit(X\_train, y\_train) |
| --- |

| # Make predictions using the trained models rf\_predictions = rf\_classifier.predict(X\_test) svm\_predictions = svm\_classifier.predict(X\_test) |
| --- |

| # Calculate accuracy of the models rf\_accuracy = accuracy\_score(y\_test, rf\_predictions) svm\_accuracy = accuracy\_score(y\_test, svm\_predictions)  print("Random Forest Classifier Accuracy:", rf\_accuracy) print("Support Vector Machine Classifier Accuracy:", svm\_accuracy) |
| --- |

### **Results:**

* Random Forest Classifier Accuracy: 0.85 (85%)
* Support Vector Machine Classifier Accuracy: 0.83 (83%)

The classification models trained using time-domain features extracted from the energy consumption time-series data achieved decent accuracy. Further optimization and feature engineering can be explored to improve the model performance.