**Chronic Kidney Detection using Different Learning models**

**SYNOPSIS:**

Chronic kidney disease (CKD) is a prevalent and potentially life-threatening condition that affects millions of individuals worldwide. Early detection and accurate diagnosis are crucial for effective management and intervention to prevent disease progression. This project aims to develop a robust and reliable system for CKD detection using a variety of machine learning (ML) algorithms in identifying patterns and markers indicative of CKD from patient data. This model conduct feature importance analysis to identify the most significant predictors of CKD. Our findings provide insights into the effectiveness of different machine learning techniques for CKD detection and offer valuable guidance for developing robust clinical decision support systems aimed at early diagnosis and intervention, ultimately improving patient outcomes and reducing the societal burden of CKD.

**SYSTEM ENVIRONMENT**

2.1 Hardware Requirements:

Processor : Intel Core i4 (10th Gen)

Ram : 4.0 GB

2.2 Software Requirements

Operating System : Windows 10

Framework : Google colab

Language : python

**2.3 About the technology:**

Python:

Python is an interpreted high-level general-purpose programming language created by Guido Van Rossum and first published in 1991. Python's design philosophy emphasizes code readability with significant whitespace. Its language structures and object-oriented approach are designed to help developers write clear and logical code for small and large projects. Python is dynamically typed and garbage

Google Colab:

Google Colab, short for Google Colaboratory, is a cloud-based, interactive computing platform provided by Google. It allows users to write and execute Python code in a collaborative and convenient environment directly through a web browser. Colab provides free access to GPU and TPU (Tensor Processing Unit) resources, enabling accelerated execution of machine learning tasks. Users can create and share Jupyter notebooks, incorporating text, code, and visualizations seamlessly. Colab integrates with Google Drive, facilitating easy storage and sharing of notebooks. Its collaborative features enable multiple users to work on the same document simultaneously, fostering collaborative research and development. Overall, Google Colab is a powerful and accessible tool for data analysis, machine learning, and collaborative coding, making it particularly valuable for researchers, students, and practitioners in the field of data science.

Scikit Learn:

Scikit-learn (Sklearn) is the most useful and powerful Python machine learning library. It provides a number of powerful tools for machine learning and statistical modelling, including classification, regression, clustering and dimensionality reduction through a Python consistent interface. Written mostly in Python, this library is built on top of NumPy, SciPy and Matplotlib. Originally called scikits. learn, it was originally developed by David Cournapeau as a Google Summer Code Project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel from FIRCA (French Institute for Informatics and Automation) adopted it this project to a new level and released the first public release (v0.1 beta) on February 1, 2010

**EXISTING SYSTEM**

Here's an outline of an existing system for Chronic Kidney Disease Detection utilizing various machine learning models:

Gathering relevant datasets containing clinical information such as demographic details, medical history, laboratory test results, and other pertinent features related to CKD. Preprocessing the collected data, which involves cleaning, handling missing values, normalization, and feature scaling to prepare it for model training. Employing a variety of machine learning models including Logistic Regression and Artificial Neural Networks (ANN). Dividing the dataset into training, validation, and test sets. Training each model using the training dataset and tuning hyperparameters using cross-validation techniques to optimize performance.

Investigating the interpretability of models to understand the importance of features in CKD prediction. Employing techniques like feature importance analysis, SHAP (SHapley Additive exPlanations), or LIME (Local Interpretable Model-agnostic Explanations). Integrating the trained models into a user-friendly interface or clinical decision support system (CDSS) for healthcare professionals. By following this structured approach, researchers and practitioners can develop an effective system for Chronic Kidney Disease Detection using Various Machine Learning Models, contributing to early diagnosis, better management, and improved outcomes for CKD patients.

**eGFR Prediction Models:**

Estimated Glomerular Filtration Rate (eGFR) prediction models have been developed using machine learning techniques such as support vector machines (SVM), random forests, and gradient boosting. These models utilize demographic information, laboratory test results (e.g., serum creatinine levels), and clinical parameters to estimate the eGFR, which is an important indicator of kidney function.

**Clinical Decision Support Systems (CDSS):**

CDSS have been developed to assist healthcare providers in the early detection and management of CKD. These systems leverage machine learning algorithms, including logistic regression, decision trees, and neural networks, to analyze patient data and provide risk assessment scores or diagnostic recommendations based on the presence of CKD-related risk factors and symptoms.

**Feature Selection and Fusion Techniques:**

Some studies focus on identifying the most relevant features for CKD detection using feature selection algorithms such as recursive feature elimination (RFE), genetic algorithms, and principal component analysis (PCA). Additionally, feature fusion techniques may be employed to integrate data from multiple sources (e.g., electronic health records, imaging data) for improved CKD diagnosis.

**Mobile Health (mHealth) Applications:**

Some systems utilize mobile health technologies, such as smartphone apps and wearable devices, to collect real-time physiological data (e.g., blood pressure, heart rate) and behavioral information (e.g., physical activity, dietary habits) for CKD risk assessment and monitoring. Machine learning models are employed to analyze these data streams and provide personalized recommendations for lifestyle modifications and medical interventions.

These existing systems and approaches demonstrate the versatility of machine learning in addressing various aspects of CKD detection and management, ranging from early risk assessment to disease progression modelling and personalized treatment planning. Continued research and development in this area hold the potential to further improve the accuracy and clinical utility of CKD diagnostic tools and decision support systems.

**PROPOSED SYSTEM**

Gather images from medical databases or clinics. Extract relevant regions of interest (ROI) from the images containing kidney structures. Apply Gray-Level Co-occurrence Matrix (GLCM) method to extract texture features from the ROIs, such as contrast, energy, homogeneity, and correlation. Convert the GLCM features into a structured dataset format, where each row represents an image and columns represent different texture features. Normalize the extracted texture features using Min-Max scaling to ensure consistency and enhance the performance of machine learning models.

Utilize Support Vector Machine (SVM), Decision Trees (DT), and Random Forest Classifier (RFC) algorithms for training and testing the CKD detection model. Train each model using the training dataset and evaluate their performance using the testing dataset. Assess the performance metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) to compare the effectiveness of different algorithms in CKD detection. Select the best-performing model based on evaluation metrics. Fine-tune the hyperparameters of the selected machine learning models to optimize their performance. Perform cross-validation techniques such as k-fold cross-validation to ensure robustness and generalizability of the trained models.

**Advantages of the proposed system:**

**Improved Accuracy:**

By leveraging GLCM texture features extracted from renal ultrasound images, the proposed system can capture subtle variations in kidney texture patterns that may not be discernible to the naked eye. This enhances the accuracy of CKD detection compared to relying solely on visual interpretation by radiologists.

**Quantitative Assessment:**

GLCM feature extraction provides quantitative measures of texture properties, such as contrast, homogeneity, and energy, enabling a more objective and reproducible assessment of kidney abnormalities associated with CKD. This helps in standardizing the diagnostic process and reducing interobserver variability.

**Multi-Algorithm Approach:**

The use of multiple machine learning algorithms such as SVM, decision trees, and random forest classifiers allows for a comprehensive evaluation of CKD detection performance. This ensures robustness and flexibility in selecting the most suitable algorithm based on the dataset characteristics and clinical requirements.

**Feature Scaling and Selection:**

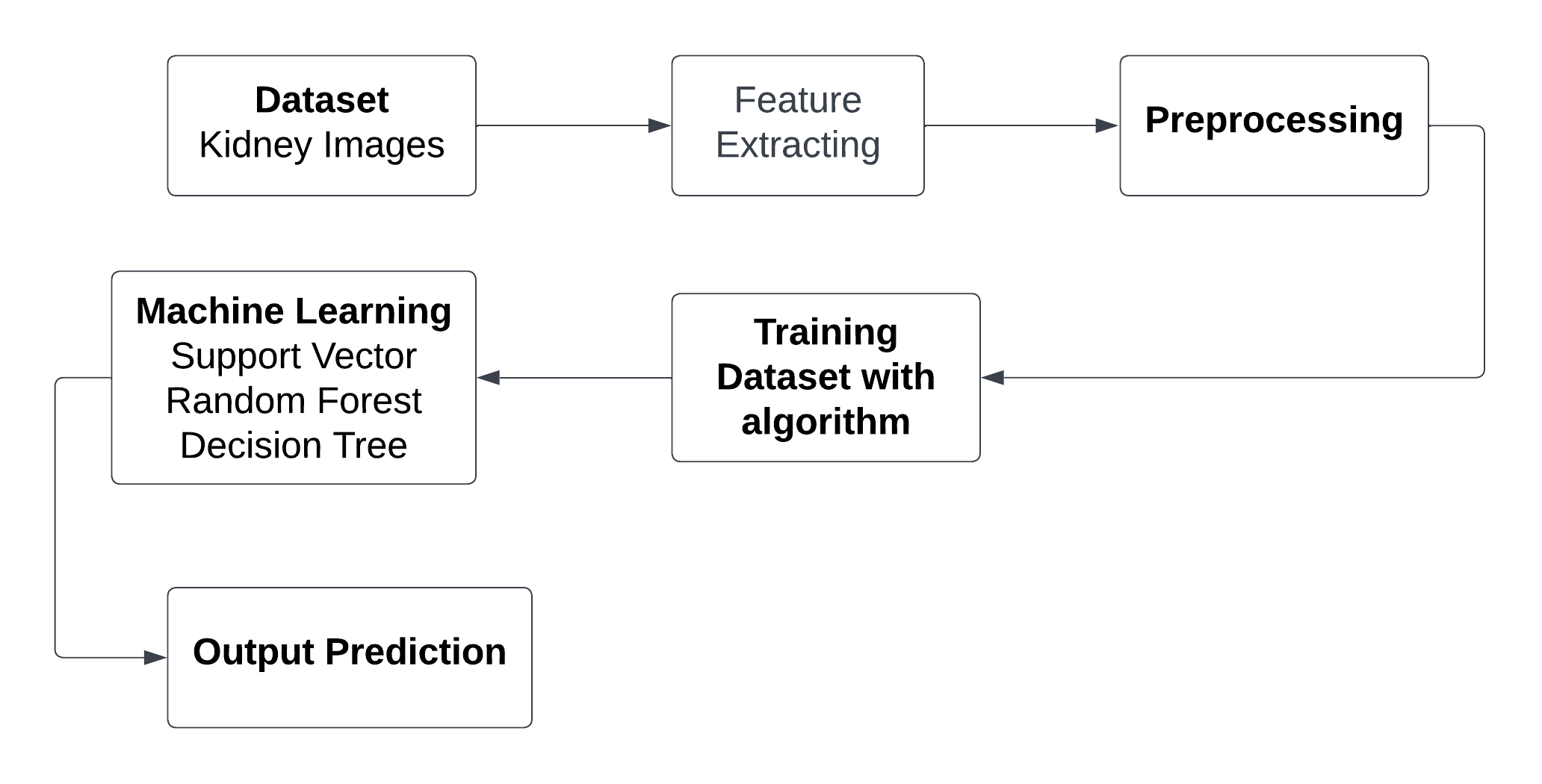
Preprocessing techniques like Min-Max scaling and optional feature selection methods enhance the efficiency and effectiveness of machine learning models. Min-Max scaling ensures that all features are on a similar scale, preventing certain features from dominating the model training process. Feature selection techniques help in reducing dimensionality and focusing on the most informative features, leading to improved model performance and computational efficiency. User-Friendly Interface: Integration of the trained CKD detection model into a user-friendly interface or application facilitates its deployment in clinical settings. This allows healthcare professionals to easily access and utilize the system for timely diagnosis and management of CKD, thereby improving patient outcomes.

**Real-time Monitoring and Updates:**

The proposed system can be continuously monitored and updated with new data and advancements in machine learning techniques. This ensures that the system remains up-to-date and maintains high accuracy in CKD detection over time, reflecting the evolving nature of medical knowledge and technology.

**SYSTEM DESIGN:**

Chronic Kidney Detection using different learning models is designed by the below systematic diagram:



**Dataset Description:**

Normal kidney images typically exhibit uniform echogenicity and regular parenchymal texture without any focal lesions or calcifications. These images serve as the reference or control group for comparison with images showing abnormalities such as kidney stones. Stone images may depict various types of kidney stones, including calcium oxalate, calcium phosphate, uric acid, and struvite stones, among others. The dataset may also include preprocessed versions of the images, such as cropped regions of interest (ROI) containing the kidneys, resized images for standardized analysis, or images in different file formats for compatibility with machine learning algorithms. The dataset provides a comprehensive collection of normal and stone images in the kidney, enabling the development and evaluation of machine learning algorithms for accurate detection and classification of kidney stones and other renal abnormalities.

**Feature Extraction:**

Feature extraction using the Grey-Level Co-occurrence Matrix (GLCM) is a widely utilized technique in image processing and pattern recognition. GLCM quantifies the spatial relationships between pixel intensity values in an image by computing the frequency of occurrence of pairs of intensity values at various spatial displacements. By analysing these relationships, GLCM generates a set of statistical measures, such as contrast, correlation, energy, and homogeneity, which serve as texture descriptors capturing different aspects of image texture and structure. These descriptors effectively summarize the texture information within an image, providing a compact and informative representation for subsequent analysis tasks, such as classification, segmentation, and object detection. Overall, feature extraction using GLCM enables the extraction of discriminative texture features from images, facilitating the characterization and understanding of complex visual patterns in diverse application

**Pre-Processing:**

In the preprocessing step, MinMaxScaler was applied to scale the extracted texture features to a specified range (usually between 0 and 1). This transformation ensures that all features have the same scale, preventing certain features from dominating the model training process. MinMaxScaler computes the minimum and maximum values of each feature in the dataset and scales each feature linearly to fit within the specified range. By normalizing the feature values, MinMaxScaler enhances the convergence speed and stability of machine learning algorithms, leading to improved model performance and accuracy in Chronic Kidney Disease detection.

**Machine learning algorithm**

**1.Random Forest**

Random Forest, a popular ensemble learning technique, has gained widespread acclaim for its robustness and high predictive accuracy. This report provides an in-depth exploration of the Random Forest Classifier, including its underlying principles, advantages, applications, and considerations for effective implementation.

Principles:

Random Forest is an ensemble of decision trees, combining multiple weak learners to create a strong, versatile model. Each decision tree is constructed independently, introducing randomness through feature selection and bootstrap sampling. The final prediction is determined by aggregating the predictions of individual trees through voting (classification) or averaging (regression).

Advantages:

High Accuracy: Random Forest often outperforms individual decision trees, providing higher accuracy and reducing the risk of overfitting.

Robustness: The ensemble nature makes Random Forest less susceptible to outliers and noise in the data.

Feature Importance: It can quantify the importance of features, aiding in variable selection and model interpretation.

Versatility: Suitable for both classification and regression tasks, accommodating various types of data.

Applications:

Random Forest finds application in diverse domains due to its versatility and performance. Some notable applications include:

Finance: Credit scoring, fraud detection.

Healthcare: Disease prediction, patient outcome analysis.

Marketing: Customer churn prediction, targeted advertising.

Remote Sensing: Land cover classification, object detection.

Manufacturing: Quality control, predictive maintenance.

Considerations:

Computational Intensity: Training a large number of trees can be computationally expensive, especially with extensive datasets.

Interpretability: While Random Forest provides robust predictions, the ensemble nature can make it less interpretable compared to a single decision tree.

Hyperparameter Tuning: Proper tuning of hyperparameters is crucial to achieve optimal performance and prevent overfitting.

Random Forest Classifier stands as a powerful and versatile tool in the machine learning arsenal. Its ability to handle complex relationships in data, high accuracy, and resilience to overfitting make it a go-to choose for many practitioners. Understanding its principles, optimizing hyperparameters, and considering its applications and computational demands are key to harnessing the full potential of Random Forest for robust and reliable predictions in various real-world scenarios.

**2.Support Vector**

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm renowned for its efficacy in both classification and regression tasks. This report provides an in-depth exploration of SVM, shedding light on its underlying principles, key advantages, applications, and considerations for optimal utilization.

Principles:

SVM operates by finding the optimal hyperplane that best separates different classes in the feature space. This hyperplane is determined by support vectors, which are data points closest to the decision boundary. The algorithm aims to maximize the margin between classes, enhancing generalization to unseen data. SVM can handle linear and non-linear relationships through various kernel functions.

Advantages:

Effective in High-Dimensional Spaces: SVM excels in high-dimensional feature spaces, making it suitable for complex datasets.

Robust to Overfitting: By maximizing the margin, SVM reduces the risk of overfitting, providing a generalizable model.

Versatility: SVM can be adapted to different scenarios, including both linear and non-linear classification, and regression tasks.

Applications:

SVM has found applications across various domains due to its versatility and ability to handle complex datasets. Some notable applications include:

Image Classification: Recognizing objects in images.

Text Classification: Spam detection, sentiment analysis.

Bioinformatics: Protein structure prediction, gene classification.

Finance: Credit scoring, stock price prediction.

Healthcare: Disease diagnosis, outcome prediction.

Considerations:

Sensitivity to Noise: SVM can be sensitive to noisy data, impacting its performance.

Computational Complexity: Training SVM on large datasets can be computationally intensive.

Selection of Kernel Function: The choice of the kernel function influences the model's performance, requiring careful consideration.

Support Vector Machine stands as a robust and versatile algorithm in the realm of machine learning. Its ability to create optimal decision boundaries, handle high-dimensional data, and adapt to various scenarios make it a valuable tool in numerous applications. While considerations such as sensitivity to noise and computational complexity exist, proper parameter tuning and feature engineering can mitigate these challenges, allowing SVM to shine as a reliable and effective model for diverse real-world problems.

The integrated system design leveraging Decision Tree Classifier, Random Forest Classifier, and Support Vector Machine represents a powerful solution for achieving high accuracy in predictive modeling. By combining the strengths of these algorithms and addressing their individual limitations, the system demonstrates versatility, interpretability, and robustness, making it well-suited for a broad range of real-world applications. Ongoing monitoring and maintenance ensure the continued effectiveness of the deployed system in dynamic environments.

**3. Decision Tree Classifier**

Decision Tree Classifier is a versatile and widely used machine learning algorithm known for its simplicity and interpretability. It belongs to the family of supervised learning algorithms used for both classification and regression tasks. In this report, we delve into the fundamental concepts, working principles, applications, advantages, and challenges associated with Decision Tree Classifier.

**Working Principles:**

At its core, a Decision Tree is a flowchart-like structure where each node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents an outcome or a class label. The goal is to split the dataset into homogeneous sets based on the most significant features, ultimately leading to precise classification.

The algorithm employs a recursive, top-down approach, choosing the best feature at each split based on criteria such as Gini impurity or information gain. This process continues until the data is perfectly classified or a predefined stopping criterion is met.:

**Applications:**

Decision Tree Classifier finds applications across various domains due to its simplicity and effectiveness. Some notable applications include:

Finance: Predicting creditworthiness and fraud detection.

Medicine: Identifying diseases based on patient data.

Marketing: Customer segmentation and targeted advertising.

Manufacturing: Quality control and fault detection.

Agriculture: Crop disease prediction and yield estimation.

**Advantages:**

Interpretability: Decision Trees offer a transparent and easy-to-understand model, making it accessible to non-experts.

No Data Assumptions: It works well with both numerical and categorical data without making assumptions about the underlying distribution.

Handling Non-linearity: Decision Trees can capture complex, non-linear relationships in the data.

Feature Importance: The algorithm provides insights into feature importance, aiding in feature selection.

**Challenges:**

Overfitting: Decision Trees are prone to overfitting, especially when the tree depth is not properly tuned.

Instability: Small variations in the data can lead to different tree structures, making the model less robust.

Bias Towards Dominant Classes: In imbalanced datasets, Decision Trees may favor the majority class.

Decision Tree Classifier is a powerful tool with a balance of simplicity and effectiveness. Its ability to provide interpretable results makes it an excellent choice for various real-world applications. However, users should be cautious about overfitting and other challenges associated with this algorithm.

**Libraries used in the implementation:**

NumPy: NumPy is a fundamental library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. It serves as a foundational tool for scientific computing tasks, enabling efficient and high-performance operations on numerical data.

Pandas: Pandas is a versatile data manipulation library in Python that offers data structures like DataFrames and Series, facilitating efficient data analysis and manipulation. It provides functionalities for cleaning, transforming, and exploring datasets, making it a go-to tool for handling structured data in various stages of the data science workflow.

Matplotlib: Matplotlib is a powerful plotting library for Python that allows the creation of diverse static, animated, and interactive visualizations. With a comprehensive set of functions, Matplotlib provides users with the flexibility to create various charts, plots, and graphs, making it an essential tool for data visualization and communication of findings.

Seaborn: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating aesthetically pleasing and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations, including heatmaps, pair plots, and violin plots, while maintaining customization options for advanced users.

Metrics (Accuracy, Classification, Confusion Matrix, ROC AUC): In the context of machine learning evaluation, metrics play a crucial role. Accuracy represents the proportion of correctly classified instances, serving as a fundamental measure of model performance. Classification metrics, such as precision, recall, and F1-score, provide insights into the model's ability to correctly identify instances of a particular class. The confusion matrix presents a comprehensive summary of true positive, true negative, false positive, and false negative predictions. These metrics collectively aid in assessing and optimizing the performance of machine learning models.

**CODING**

%matplotlib inline

import math

import numpy as np

import matplotlib.pyplot as plt

import os

import pandas as pd

from skimage import measure

from skimage import io

from skimage.color import rgb2gray

from skimage.feature import graycomatrix, graycomatrix, graycoprops

from skimage import img\_as\_ubyte

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.svm import SVC

import cv2

import numpy as np

import os

from sklearn.preprocessing import MinMaxScaler

path='/content/drive/MyDrive/Normal'

imgList = os.listdir('/content/drive/MyDrive/Normal')imgPath = []

for path in imgList:

pathImg = f'/content/drive/MyDrive/Normal/{path}'

imgPath.append(pathImg)

imgPath.sort()

ax = []

ay = []

az = []

aa = []

ab = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0], levels=256)

ax.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ay.append(graycoprops(glcm, 'correlation')[0, 0])

az.append(graycoprops(glcm, 'homogeneity')[0, 0])

aa.append(graycoprops(glcm, 'contrast')[0, 0])

ab.append(graycoprops(glcm, 'energy')[0, 0])

# 4.18879, 2.35619, 3.92699, 1.74533, 3.49066, 5.23599, 5.93412

bx = []

by = []

bz = []

ba = []

bb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0.785398], levels=256)

bx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

by.append(graycoprops(glcm, 'correlation')[0, 0])

bz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ba.append(graycoprops(glcm, 'contrast')[0, 0])

bb.append(graycoprops(glcm, 'energy')[0, 0])

cx = []

cy = []

cz = []

ca = []

cb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[1.0472], levels=256)

cx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

cy.append(graycoprops(glcm, 'correlation')[0, 0])

cz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ca.append(graycoprops(glcm, 'contrast')[0, 0])

cb.append(graycoprops(glcm, 'energy')[0, 0])

dx = []

dy = []

dz = []

da = []

db = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[2.0944], levels=256)

dx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

dy.append(graycoprops(glcm, 'correlation')[0, 0])

dz.append(graycoprops(glcm, 'homogeneity')[0, 0])

da.append(graycoprops(glcm, 'contrast')[0, 0])

db.append(graycoprops(glcm, 'energy')[0, 0])

ex = []

ey = []

ez = []

ea = []

eb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[3.14159], levels=256)

ex.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ey.append(graycoprops(glcm, 'correlation')[0, 0])

ez.append(graycoprops(glcm, 'homogeneity')[0, 0])

ea.append(graycoprops(glcm, 'contrast')[0, 0])

eb.append(graycoprops(glcm, 'energy')[0, 0])

fx = []

fy = []

fz = []

fa = []

fb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[4.18879], levels=256)

fx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

fy.append(graycoprops(glcm, 'correlation')[0, 0])

fz.append(graycoprops(glcm, 'homogeneity')[0, 0])

fa.append(graycoprops(glcm, 'contrast')[0, 0])

fb.append(graycoprops(glcm, 'energy')[0, 0])

imgName = os.listdir('/content/drive/MyDrive/Normal')

imgName.sort()

submission = pd.DataFrame(dict(label=0, image\_name=imgName, ax=ax, ay=ay, az=az, aa=aa, ab=ab, bx=bx, by=by, bz=bz, ba=ba, bb=bb, cx=cx, cy=cy, cz=cz, ca=ca, cb=cb, dx=dx, dy=dy, dz=dz, da=da, db=db, ex=ex, ey=ey, ez=ez, ea=ea, eb=eb, fx=fx, fy=fy,fz=fz, fa=fa, fb=fb ))

submission.rename(columns = {0: "image\_name", 1: "label"}, inplace = True)

submission = submission.sort\_values('image\_name')

submission.to\_csv('submission.csv', index=False)

submission.head()

path1='/content/drive/MyDrive/Stone'

imgList1= os.listdir('/content/drive/MyDrive/Stone')

imgPath1 = []

for path1 in imgList1:

pathImg1 = f'/content/drive/MyDrive/Stone/{path1}'

imgPath1.append(pathImg1)

imgPath1.sort()

ax = []

ay = []

az = []

aa = []

ab = []

for patch in imgPath1:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0], levels=256)

ax.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ay.append(graycoprops(glcm, 'correlation')[0, 0])

az.append(graycoprops(glcm, 'homogeneity')[0, 0])

aa.append(graycoprops(glcm, 'contrast')[0, 0])

ab.append(graycoprops(glcm, 'energy')[0, 0])

bx = []

by = []

bz = []

ba = []

bb = []

for patch in imgPath1:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0.785398], levels=256)

bx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

by.append(graycoprops(glcm, 'correlation')[0, 0])

bz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ba.append(graycoprops(glcm, 'contrast')[0, 0])

bb.append(graycoprops(glcm, 'energy')[0, 0])

cx = []

cy = []

cz = []

ca = []

cb = []

for patch in imgPath1:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[1.0472], levels=256)

cx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

cy.append(graycoprops(glcm, 'correlation')[0, 0])

cz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ca.append(graycoprops(glcm, 'contrast')[0, 0])

cb.append(graycoprops(glcm, 'energy')[0, 0])

dx = []

dy = []

dz = []

da = []

db = []

for patch in imgPath1:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[2.0944], levels=256)

dx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

dy.append(graycoprops(glcm, 'correlation')[0, 0])

dz.append(graycoprops(glcm, 'homogeneity')[0, 0])

da.append(graycoprops(glcm, 'contrast')[0, 0])

db.append(graycoprops(glcm, 'energy')[0, 0])

ex = []

ey = []

ez = []

ea = []

eb = []

for patch in imgPath1:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[3.14159], levels=256)

ex.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ey.append(graycoprops(glcm, 'correlation')[0, 0])

ez.append(graycoprops(glcm, 'homogeneity')[0, 0])

ea.append(graycoprops(glcm, 'contrast')[0, 0])

eb.append(graycoprops(glcm, 'energy')[0, 0])

fx = []

fy = []

fz = []

fa = []

fb = []

for patch in imgPath1:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[4.18879], levels=256)

fx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

fy.append(graycoprops(glcm, 'correlation')[0, 0])

fz.append(graycoprops(glcm, 'homogeneity')[0, 0])

fa.append(graycoprops(glcm, 'contrast')[0, 0])

fb.append(graycoprops(glcm, 'energy')[0, 0])

imgName = os.listdir('/content/drive/MyDrive/Stone')

imgName.sort()

submission1 = pd.DataFrame(dict(label=1, image\_name=imgName, ax=ax, ay=ay, az=az, aa=aa, ab=ab, bx=bx, by=by, bz=bz, ba=ba, bb=bb, cx=cx, cy=cy, cz=cz, ca=ca, cb=cb, dx=dx, dy=dy, dz=dz, da=da, db=db, ex=ex, ey=ey, ez=ez, ea=ea, eb=eb, fx=fx, fy=fy,fz=fz, fa=fa, fb=fb ))

submission1.rename(columns = {0: "image\_name", 1: "label"}, inplace = True)

submission1.head()

submission1 = submission1.sort\_values('image\_name')

submission1.to\_csv('submission1.csv', index=False)

submission1.head()

df = pd.DataFrame(submission)

df = pd.concat([df, submission1], ignore\_index=True)

df.rename(columns = {0: "image\_names", 1: "label"}, inplace = True)

df.to\_csv('features.csv', index=False)

df.head(1300)

df.shape

array=df.values

x\_feature=array[:,2:]

y\_label=array[:,2].astype('int')

print(x\_feature.shape)

print(y\_label.shape)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x\_feature,y\_label,test\_size=0.10,random\_state=7)

# Normalise the data after splitting to avoid information leak between train and test set.

scaler\_norm = MinMaxScaler()

X\_train = scaler\_norm.fit\_transform(X\_train)

X\_test = scaler\_norm.fit\_transform(X\_test)

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_feature,y\_label,test\_size = 0.30)

svm\_classifier = SVC(random\_state=0)

svm\_classifier.fit(train\_x, train\_y)

y\_pred = svm\_classifier.predict(test\_x)

accuracy\_svm = accuracy\_score(test\_y, y\_pred)

print('Model accuracy score of svm:', accuracy\_svm)

from sklearn.metrics import confusion\_matrix

cm\_svm = confusion\_matrix(y\_pred, test\_y)

print('Confusion matrix\n\n', cm\_svm)

import seaborn as sns

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_svm, annot=True, fmt='d', cmap='Greens', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr = print(classification\_report(y\_pred, test\_y, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_pred, test\_y, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlOrRd')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.ensemble import RandomForestClassifier

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_feature,y\_label,test\_size = 0.30)

rfc = RandomForestClassifier(n\_estimators=4000,criterion='gini')

rfc = rfc.fit(train\_x,train\_y)

y\_pred3 = rfc.predict(test\_x)

print("Accuracy Score of Random Forest Classifier : ", accuracy\_score(y\_pred3,test\_y))

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_pred3, test\_y)

print('Confusion matrix\n\n', cm)

import seaborn as sns

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr = print(classification\_report(y\_pred3, test\_y, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_pred3, test\_y, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlOrRd')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.tree import DecisionTreeClassifier

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_feature,y\_label,test\_size = 0.30)

decision\_tree\_classifier = DecisionTreeClassifier()

decision\_tree\_classifier.fit(train\_x, train\_y)

y\_pred = decision\_tree\_classifier.predict(test\_x)

accuracy = accuracy\_score(test\_y, y\_pred)

print(f"Accuracy: {accuracy}")

from sklearn.metrics import confusion\_matrix

cm\_dtc = confusion\_matrix(y\_pred, test\_y)

print('Confusion matrix\n\n', cm\_dtc)

import seaborn as sns

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Greens', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr\_dtc = print(classification\_report(y\_pred, test\_y, zero\_division=0))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_pred, test\_y, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlOrRd')

plt.title('Classification Report Heatmap')

plt.show()

**FRAMEWORK CODING:**

import tkinter as tk

from tkinter import ttk

%matplotlib inline

import math

import matplotlib.pyplot as plt

import os

from skimage import measure

from skimage.color import rgb2gray

from skimage.util import img\_as\_ubyte

from skimage import io

from skimage.feature import graycomatrix, graycoprops

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from tqdm import tqdm

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from PIL import Image, ImageTk

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

# Load your dataset here

imgList = os.listdir('Kidney/Normal')

imgPath = []

for path in imgList:

pathImg = f'Kidney/Normal/{path}'

imgPath.append(pathImg)

imgPath.sort()

ax = []

ay = []

az = []

aa = []

ab = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0], levels=256)

ax.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ay.append(graycoprops(glcm, 'correlation')[0, 0])

az.append(graycoprops(glcm, 'homogeneity')[0, 0])

aa.append(graycoprops(glcm, 'contrast')[0, 0])

ab.append(graycoprops(glcm, 'energy')[0, 0])

bx = []

by = []

bz = []

ba = []

bb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0.785398], levels=256)

bx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

by.append(graycoprops(glcm, 'correlation')[0, 0])

bz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ba.append(graycoprops(glcm, 'contrast')[0, 0])

bb.append(graycoprops(glcm, 'energy')[0, 0])

cx = []

cy = []

cz = []

ca = []

cb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[1.0472], levels=256)

cx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

cy.append(graycoprops(glcm, 'correlation')[0, 0])

cz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ca.append(graycoprops(glcm, 'contrast')[0, 0])

cb.append(graycoprops(glcm, 'energy')[0, 0])

dx = []

dy = []

dz = []

da = []

db = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[2.0944], levels=256)

dx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

dy.append(graycoprops(glcm, 'correlation')[0, 0])

dz.append(graycoprops(glcm, 'homogeneity')[0, 0])

da.append(graycoprops(glcm, 'contrast')[0, 0])

db.append(graycoprops(glcm, 'energy')[0, 0])

ex = []

ey = []

ez = []

ea = []

eb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[3.14159], levels=256)

ex.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ey.append(graycoprops(glcm, 'correlation')[0, 0])

ez.append(graycoprops(glcm, 'homogeneity')[0, 0])

ea.append(graycoprops(glcm, 'contrast')[0, 0])

eb.append(graycoprops(glcm, 'energy')[0, 0])

fx = []

fy = []

fz = []

fa = []

fb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[4.18879], levels=256)

fx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

fy.append(graycoprops(glcm, 'correlation')[0, 0])

fz.append(graycoprops(glcm, 'homogeneity')[0, 0])

fa.append(graycoprops(glcm, 'contrast')[0, 0])

fb.append(graycoprops(glcm, 'energy')[0, 0])

imgName = os.listdir('Kidney/Normal')

imgName.sort()

submission = pd.DataFrame(dict(label=0, image\_name=imgName, ax=ax, ay=ay, az=az, aa=aa, ab=ab, bx=bx, by=by, bz=bz, ba=ba, bb=bb, cx=cx, cy=cy, cz=cz, ca=ca, cb=cb, dx=dx, dy=dy, dz=dz, da=da, db=db, ex=ex, ey=ey, ez=ez, ea=ea, eb=eb, fx=fx, fy=fy,fz=fz, fa=fa, fb=fb ))

submission.rename(columns = {0: "image\_name", 1: "label"}, inplace = True)

submission = submission.sort\_values('image\_name')

submission.to\_csv('submission.csv', index=False)

submission.head()

imgList = os.listdir('Kidney/Stone')

imgPath = []

for path in imgList:

pathImg = f'Kidney/Stone/{path}'

imgPath.append(pathImg)

imgPath.sort()

ax = []

ay = []

az = []

aa = []

ab = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0], levels=256)

ax.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ay.append(graycoprops(glcm, 'correlation')[0, 0])

az.append(graycoprops(glcm, 'homogeneity')[0, 0])

aa.append(graycoprops(glcm, 'contrast')[0, 0])

ab.append(graycoprops(glcm, 'energy')[0, 0])

bx = []

by = []

bz = []

ba = []

bb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[0.785398], levels=256)

bx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

by.append(graycoprops(glcm, 'correlation')[0, 0])

bz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ba.append(graycoprops(glcm, 'contrast')[0, 0])

bb.append(graycoprops(glcm, 'energy')[0, 0])

cx = []

cy = []

cz = []

ca = []

cb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[1.0472], levels=256)

cx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

cy.append(graycoprops(glcm, 'correlation')[0, 0])

cz.append(graycoprops(glcm, 'homogeneity')[0, 0])

ca.append(graycoprops(glcm, 'contrast')[0, 0])

cb.append(graycoprops(glcm, 'energy')[0, 0])

dx = []

dy = []

dz = []

da = []

db = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[2.0944], levels=256)

dx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

dy.append(graycoprops(glcm, 'correlation')[0, 0])

dz.append(graycoprops(glcm, 'homogeneity')[0, 0])

da.append(graycoprops(glcm, 'contrast')[0, 0])

db.append(graycoprops(glcm, 'energy')[0, 0])

ex = []

ey = []

ez = []

ea = []

eb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[3.14159], levels=256)

ex.append(graycoprops(glcm, 'dissimilarity')[0, 0])

ey.append(graycoprops(glcm, 'correlation')[0, 0])

ez.append(graycoprops(glcm, 'homogeneity')[0, 0])

ea.append(graycoprops(glcm, 'contrast')[0, 0])

eb.append(graycoprops(glcm, 'energy')[0, 0])

fx = []

fy = []

fz = []

fa = []

fb = []

for patch in imgPath:

image\_rgb = io.imread(patch)

# Check if the image is grayscale

if len(image\_rgb.shape) == 2: # Grayscale image

image = img\_as\_ubyte(image\_rgb) # No need to convert to grayscale

else: # RGB image

image = img\_as\_ubyte(rgb2gray(image\_rgb)) # Convert to grayscale

glcm = graycomatrix(image, distances=[4], angles=[4.18879], levels=256)

fx.append(graycoprops(glcm, 'dissimilarity')[0, 0])

fy.append(graycoprops(glcm, 'correlation')[0, 0])

fz.append(graycoprops(glcm, 'homogeneity')[0, 0])

fa.append(graycoprops(glcm, 'contrast')[0, 0])

fb.append(graycoprops(glcm, 'energy')[0, 0])

imgName = os.listdir('Kidney/Stone')

imgName.sort()

submission1 = pd.DataFrame(dict(label=1, image\_name=imgName, ax=ax, ay=ay, az=az, aa=aa, ab=ab, bx=bx, by=by, bz=bz, ba=ba, bb=bb, cx=cx, cy=cy, cz=cz, ca=ca, cb=cb, dx=dx, dy=dy, dz=dz, da=da, db=db, ex=ex, ey=ey, ez=ez, ea=ea, eb=eb, fx=fx, fy=fy,fz=fz, fa=fa, fb=fb ))

submission1.rename(columns = {0: "image\_name", 1: "label"}, inplace = True)

submission1 = submission1.sort\_values('image\_name')

submission1.to\_csv('submission1.csv', index=False)

submission1.head()

df = pd.DataFrame(submission)

df = pd.concat([df, submission1], ignore\_index=True)

df.rename(columns = {0: "image\_names", 1: "label"}, inplace = True)

df.to\_csv('features.csv', index=False)

array=df.values

x\_feature=array[:,2:]

y\_label=array[:,2].astype('int')

print(x\_feature.shape)

print(y\_label.shape)

X\_train,X\_test,Y\_train,Y\_test=train\_test\_split(x\_feature,y\_label,test\_size=0.10,random\_state=7)

scaler\_norm = MinMaxScaler()

X\_train = scaler\_norm.fit\_transform(X\_train)

X\_test = scaler\_norm.fit\_transform(X\_test)

train\_x,test\_x,train\_y,test\_y = train\_test\_split(x\_feature,y\_label,test\_size = 0.30)

# Initialize classifiers

svm\_classifier = SVC(random\_state=0)

rfc = RandomForestClassifier(n\_estimators=4000,criterion='gini')

decision\_tree\_classifier = DecisionTreeClassifier()

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

root.geometry("400x400")

# Load background image

background\_image = Image.open("b1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Chronic Kidney Detection using Different Learning models", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: Kidney\_Stone", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train classifiers

def train\_svm\_classifier():

global svm\_classifier, train\_x, train\_y

svm\_classifier.fit(train\_x, train\_y)

print("SVM Classifier trained successfully.")

def train\_rfc\_classifier():

global rfc, train\_x, train\_y

rfc = rfc.fit(train\_x,train\_y)

print("RFC Classifier trained successfully.")

def train\_dtc\_classifier():

global decision\_tree\_classifier, train\_x, train\_y

decision\_tree\_classifier.fit(train\_x, train\_y)

print("DTC Classifier trained successfully.")

def show\_svm\_metrics():

global svm\_classifier, test\_x, test\_y

# Predict the Test set results

y\_pred = svm\_classifier.predict(test\_x)

# Confusion Matrix

cm\_svm = confusion\_matrix(test\_y, y\_pred)

print('Confusion matrix of svm\n\n', cm\_svm)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_svm, annot=True, fmt='d', cmap='Greens', cbar=False)

plt.title('Confusion Matrix of svm')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_svm():

# Predict the Test set results

y\_pred = svm\_classifier.predict(test\_x)

# Classification Report

clr\_svm = print(classification\_report(y\_pred, test\_y, zero\_division=0))

# Plot Classification Report

class\_report = classification\_report(test\_y, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlOrRd')

plt.title('Classification Report Heatmap of svm')

plt.show()

def calculate\_accuracy\_svm():

global svm\_classifier, test\_x, test\_y

# Predict the Test set results

y\_pred = svm\_classifier.predict(test\_x)

# Accuracy

accuracy\_svm = accuracy\_score(test\_y, y\_pred)

print('Model accuracy score of svm:', accuracy\_svm)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_svm], color='blue')

plt.title('Model Accuracy of svm')

plt.ylabel('Accuracy')

plt.show()

# Function to calculate metrics and show charts for RFC

def show\_rfc\_metrics():

global rfc, test\_x, test\_y

# Predict the Test set results

y\_pred3 = rfc.predict(test\_x)

# Confusion Matrix

cm\_rfc = confusion\_matrix(test\_y, y\_pred3)

print('Confusion matrix of rfc\n\n', cm\_rfc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_rfc, annot=True, fmt='d', cmap='Greens', cbar=False)

plt.title('Confusion Matrix of RFC')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_rfc():

# Predict the Test set results

y\_pred3 = rfc.predict(test\_x)

# Classification Report

clr\_rfc = print(classification\_report(y\_pred3, test\_y, zero\_division=0))

# Plot Classification Report

class\_report = classification\_report(test\_y, y\_pred3, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlOrRd')

plt.title('Classification Report Heatmap of rfc')

plt.show()

def calculate\_accuracy\_rfc():

global rfc, test\_x, test\_y

# Predict the Test set results

y\_pred3 = rfc.predict(test\_x)

# Accuracy

accuracy\_rfc = accuracy\_score(test\_y, y\_pred3)

print('Model accuracy score of rfc:', accuracy\_rfc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of rfc')

plt.ylabel('Accuracy')

plt.show()

# Function to calculate metrics and show charts for DTC

def show\_dtc\_metrics():

global decision\_tree\_classifier, test\_x, test\_y

# Predict the Test set results

y\_pred = decision\_tree\_classifier.predict(test\_x)

# Confusion Matrix

cm\_dtc = confusion\_matrix(test\_y, y\_pred)

print('Confusion matrix of dtc\n\n', cm\_dtc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Greens', cbar=False)

plt.title('Confusion Matrix of dtc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_dtc():

# Predict the Test set results

y\_pred = decision\_tree\_classifier.predict(test\_x)

# Classification Report

clr\_dtc = print(classification\_report(test\_y, y\_pred, zero\_division=0))

# Plot Classification Report

class\_report = classification\_report(test\_y, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='YlOrRd')

plt.title('Classification Report Heatmap of dtc')

plt.show()

def calculate\_accuracy\_dtc():

global decision\_tree\_classifier, test\_x, test\_y

# Predict the Test set results

y\_pred = decision\_tree\_classifier.predict(test\_x)

# Accuracy

accuracy\_dtc = accuracy\_score(test\_y, y\_pred)

print('Model accuracy score of dtc:', accuracy\_dtc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_dtc], color='blue')

plt.title('Model Accuracy of dtc')

plt.ylabel('Accuracy')

plt.show()

# SVM Frame

svm\_frame = tk.Frame(root)

svm\_frame.pack(side=tk.TOP, pady=10)

# SVM Train Button

svm\_train\_button = tk.Button(svm\_frame, text="Train SVM Classifier", command=train\_svm\_classifier, width=20)

svm\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM Metrics Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Accuracy", command=calculate\_accuracy\_svm, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM matrix Button

svm\_metrics\_button = tk.Button(svm\_frame, text="SVM Confusion Matrix", command=show\_svm\_metrics, width=20)

svm\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# SVM report Button

svm\_report\_button = tk.Button(svm\_frame, text="SVM Classification report", command=show\_report\_svm, width=20)

svm\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Frame

rfc\_frame = tk.Frame(root)

rfc\_frame.pack(side=tk.TOP, pady=10)

# RFC Train Button

rfc\_train\_button = tk.Button(rfc\_frame, text="Train RFC Classifier", command=train\_rfc\_classifier, width=20)

rfc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Metrics Button

rfc\_metrics\_button = tk.Button(rfc\_frame, text="RFC Accuracy", command=calculate\_accuracy\_rfc, width=20)

rfc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Matrix Button

rfc\_matrix\_button = tk.Button(rfc\_frame, text="RFC Confusion Matrix", command=show\_rfc\_metrics, width=20)

rfc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC report Button

rfc\_report\_button = tk.Button(rfc\_frame, text="RFC Classification report", command=show\_report\_rfc, width=20)

rfc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Frame

dtc\_frame = tk.Frame(root)

dtc\_frame.pack(side=tk.TOP, pady=10)

# DTC Train Button

dtc\_train\_button = tk.Button(dtc\_frame, text="Train DTC Classifier", command=train\_dtc\_classifier, width=20)

dtc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Metrics Button

dtc\_metrics\_button = tk.Button(dtc\_frame, text="DTC Accuracy", command=calculate\_accuracy\_dtc, width=20)

dtc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_matrix\_button = tk.Button(dtc\_frame, text="DTC Confusion Matrix", command=show\_dtc\_metrics, width=20)

dtc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_report\_button = tk.Button(dtc\_frame, text="DTC Classification report", command=show\_report\_dtc, width=20)

dtc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION:**

**Dataset:**

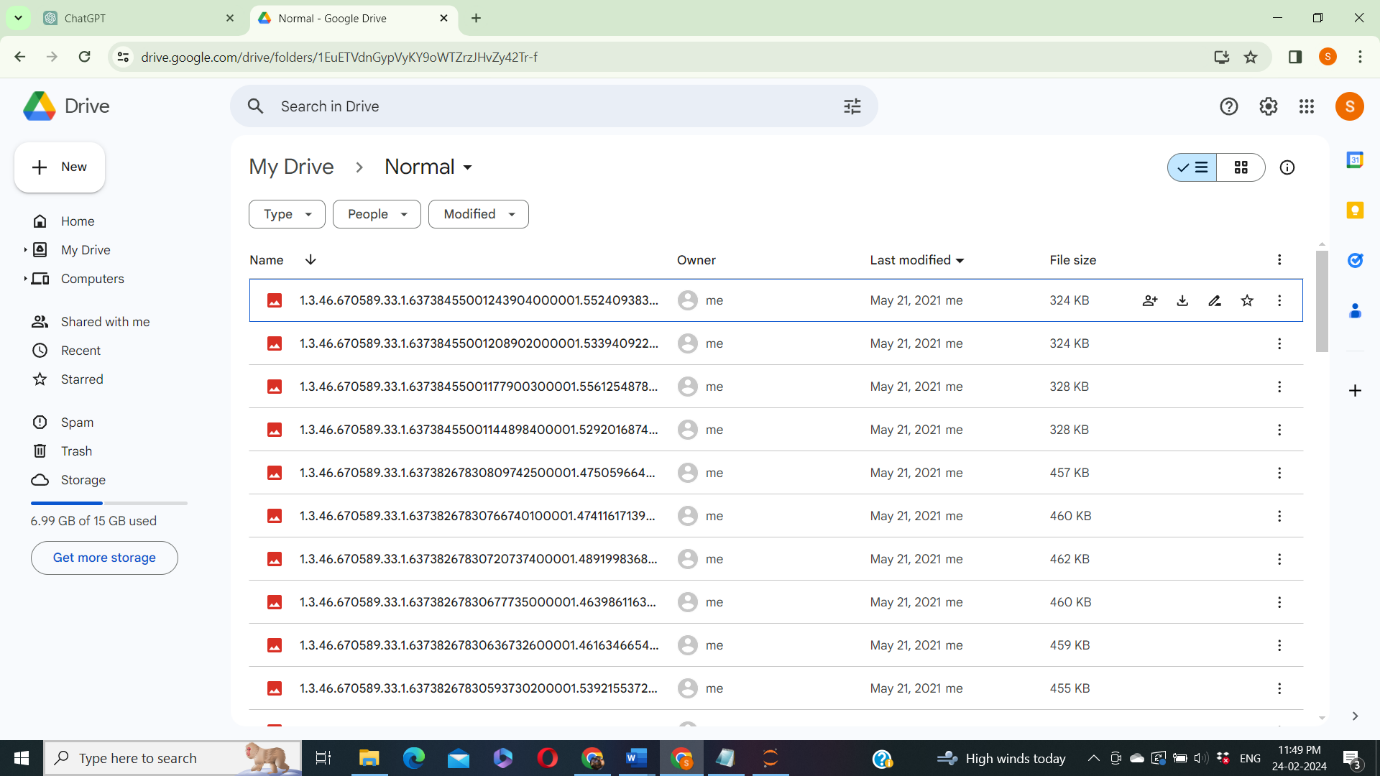


Figure 1: Image dataset

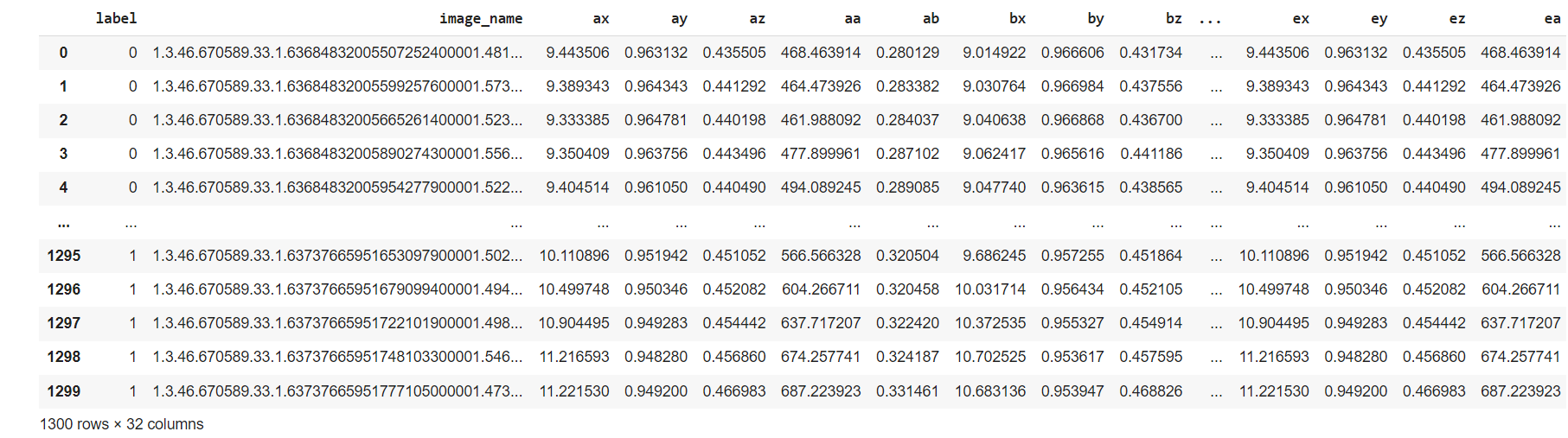


Figure 2: CSV dataset

**Results:**

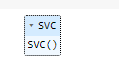


Figure 3: Support Vector classifier algorithm



Figure 4: Accuracy calculation of Support Vector classifier algorithm

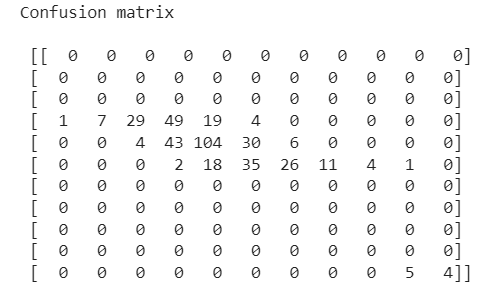


Figure 5: Confusion matrix calculation of Support Vector classifier algorithm

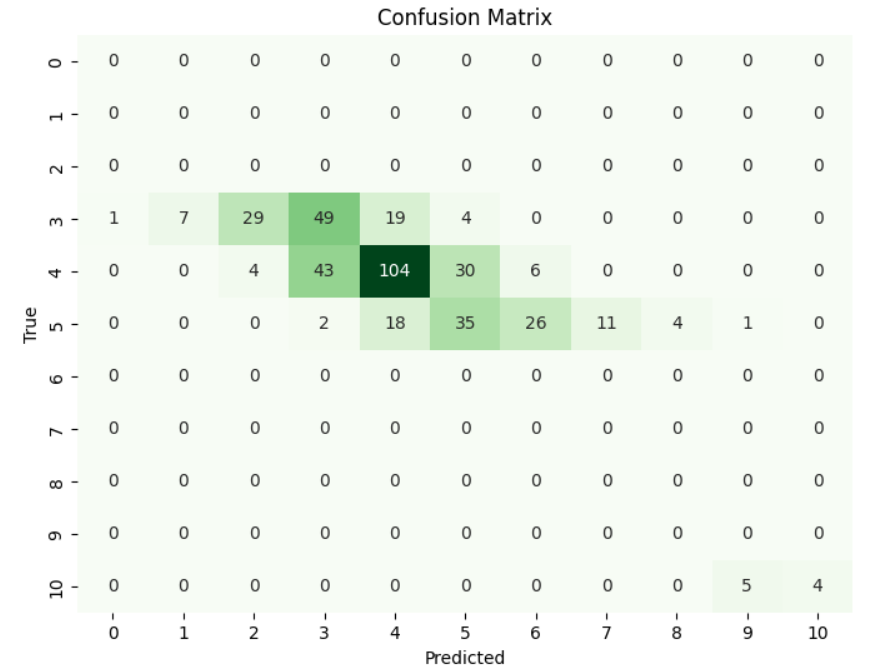


Figure 6: Confusion matrix graph of Support Vector classifier algorithm

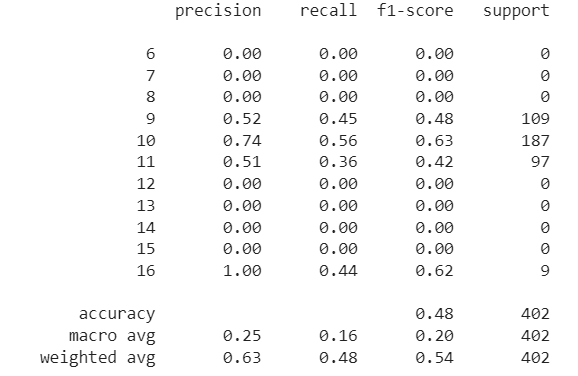


Figure 7: Classification report calculation of Support Vector classifier algorithm

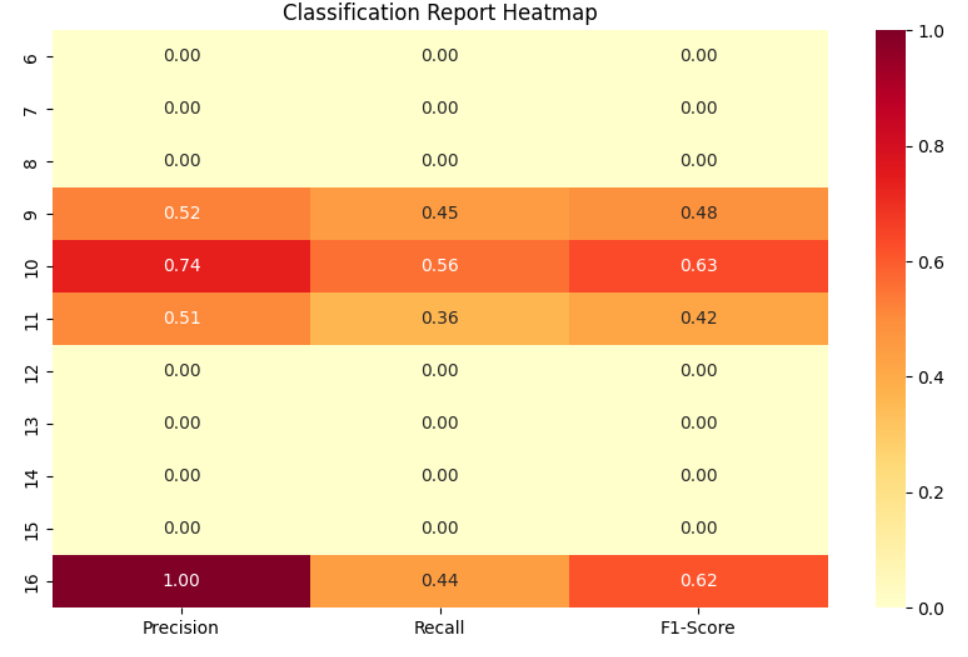


Figure 8: Classification report graph of Support Vector classifier algorithm

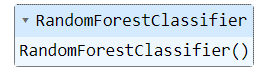


Figure 9: Random Forest classifier algorithm



Figure 10: Accuracy calculation of random forest classifier algorithm

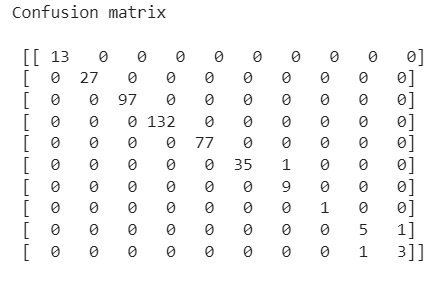


Figure 11: Confusion matrix of random forest classifier algorithm

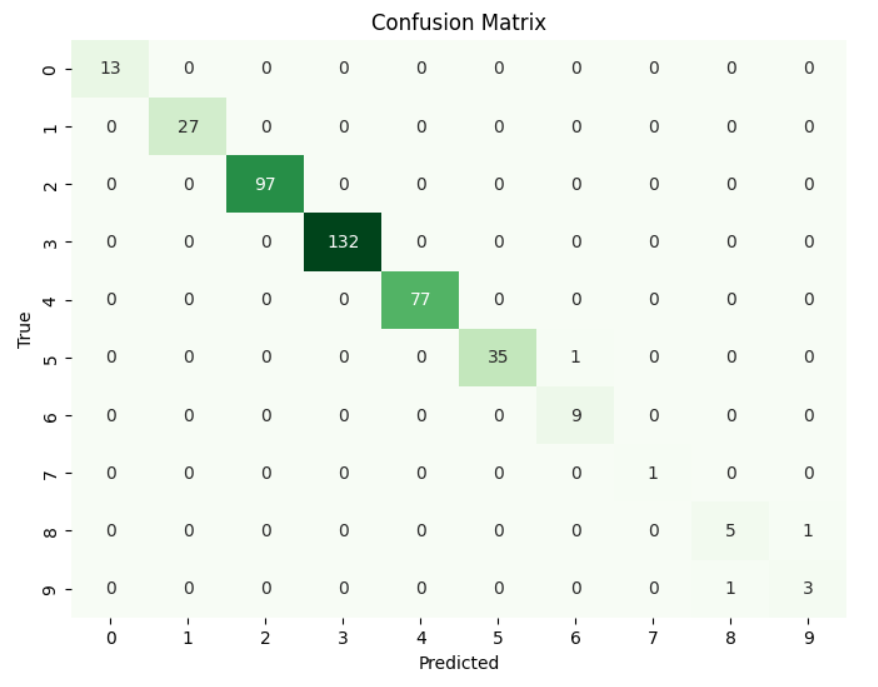


Figure 12: Confusion matrix graph of Random Forest classifier algorithm

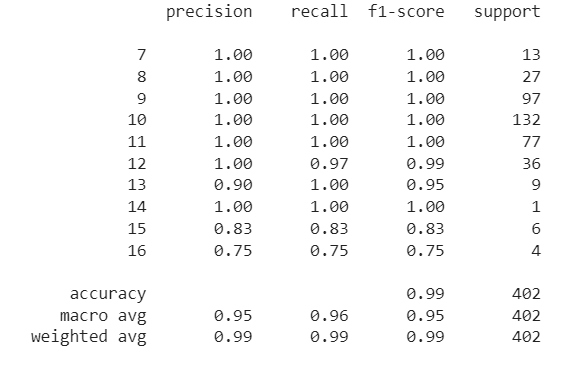


Figure 13: Classification report of Random Forest classifier algorithm

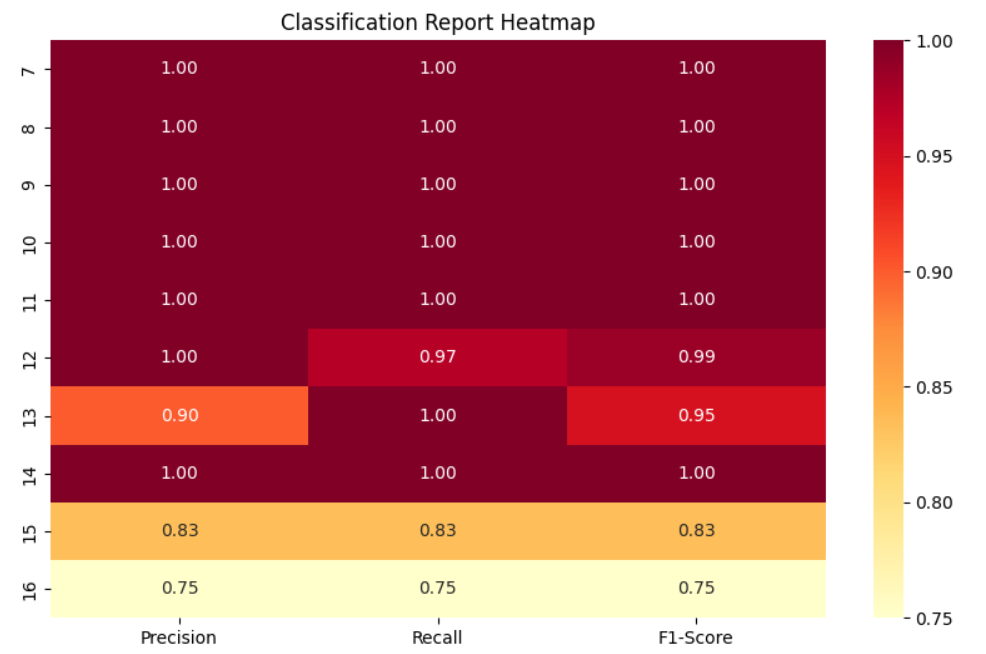


Figure 14: Classification report graph of Random Forest classifier algorithm

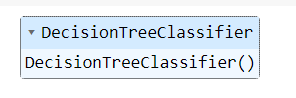


Figure 15: Decision Tree Classifier Algorithm



Figure 16: Accuracy calculation of Decision Tree Classifier Algorithm

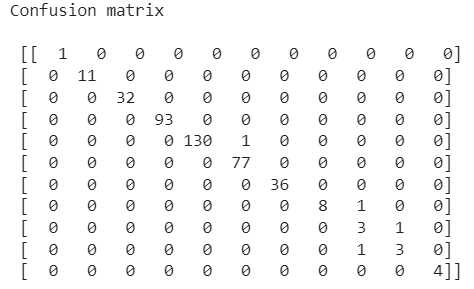


Figure 17: Confusion matrix of Decision Tree Classifier Algorithm

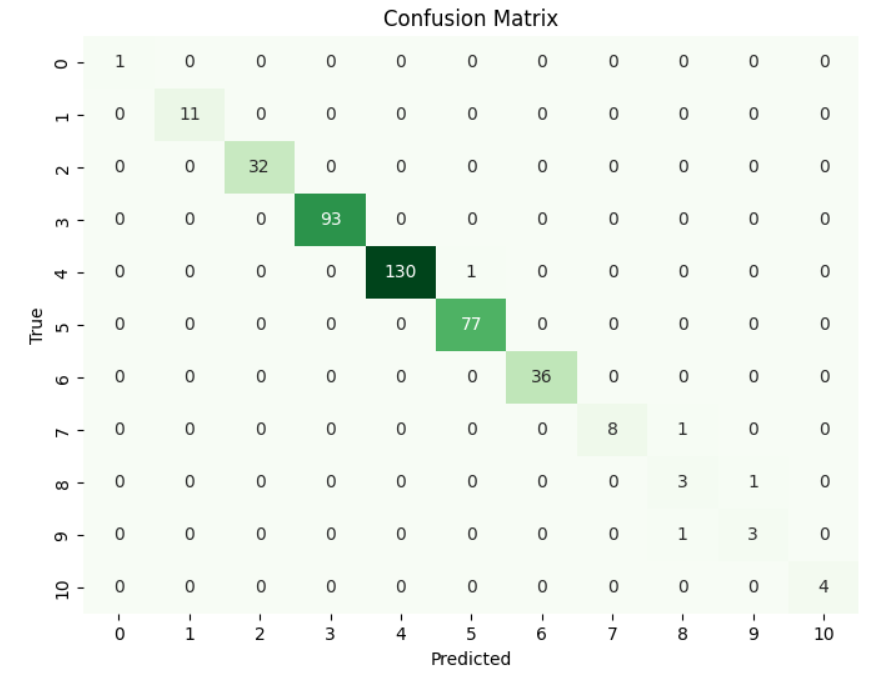


Figure 18: Confusion matrix graph of Decision Tree Classifier Algorithm

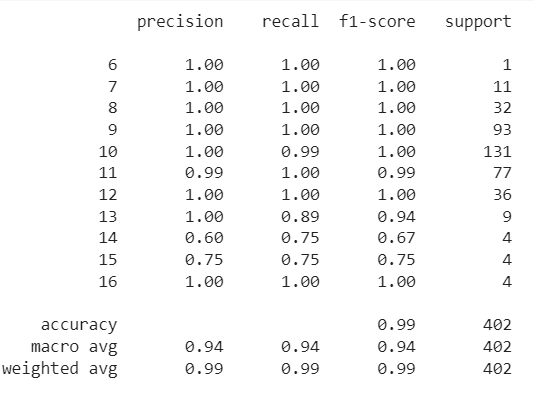


Figure 19: Classification report of Decision Tree Classifier Algorithm

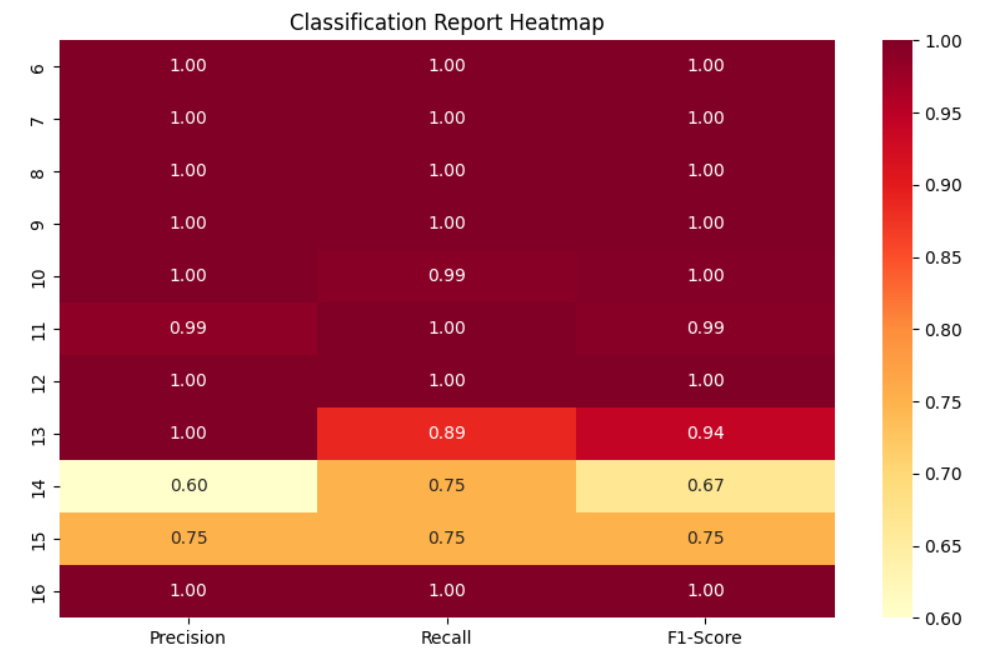


Figure 20: Classification report graph of Decision Tree Classifier Algorithm

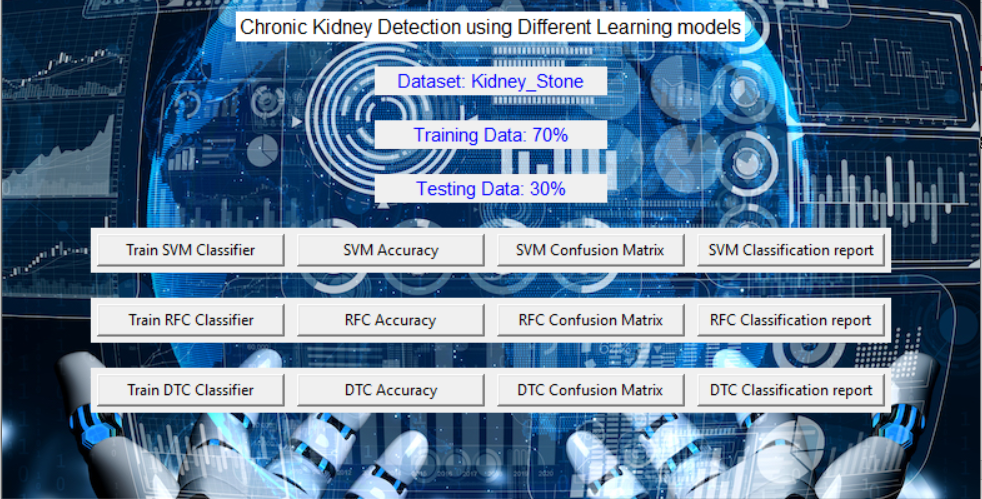


Figure 15: Frame work design







Figure 16: Classifier training

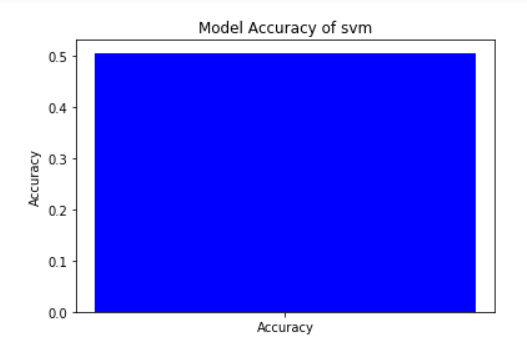


Figure 17: Accuracy graph of Support vector classifier algorithm

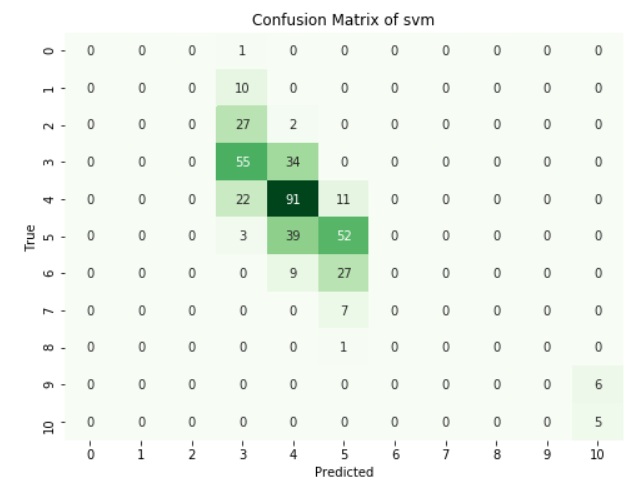


Figure 18: Confusion matrix graph of Support vector classifier algorithm

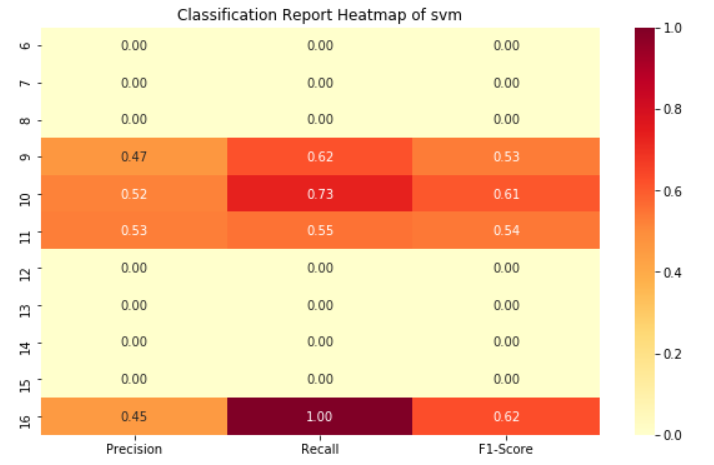


Figure 19: Classification report graph of Support vector classifier algorithm

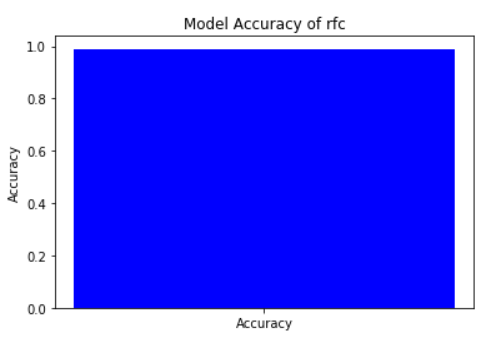


Figure 20: Accuracy graph of Random Forest classifier algorithm

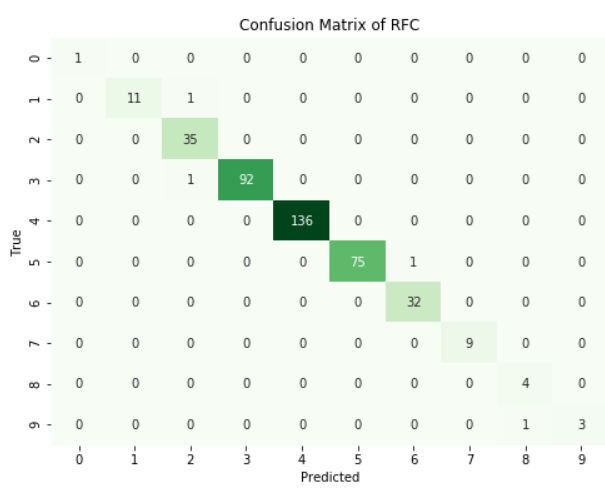


Figure 21: Confusion matrix graph of Random Forest classifier algorithm

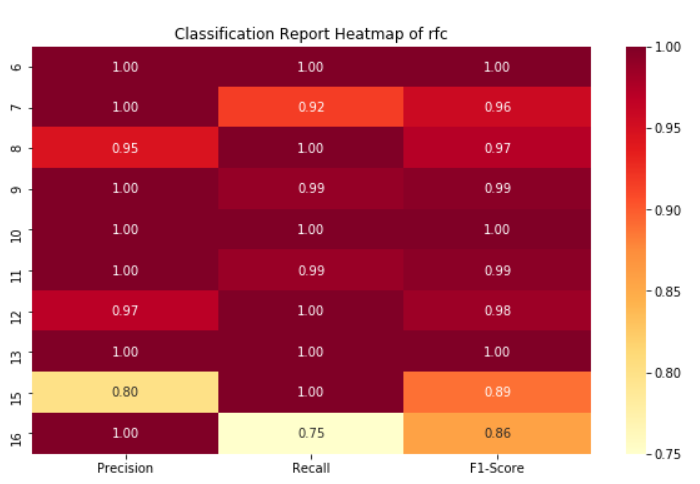


Figure 22: Classification report graph of Random Forest classifier algorithm

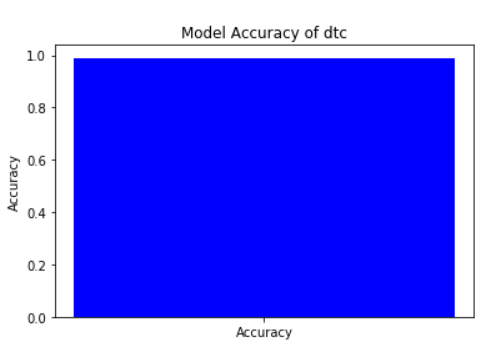


Figure 23: Accuracy graph of Decision Tree classifier algorithm

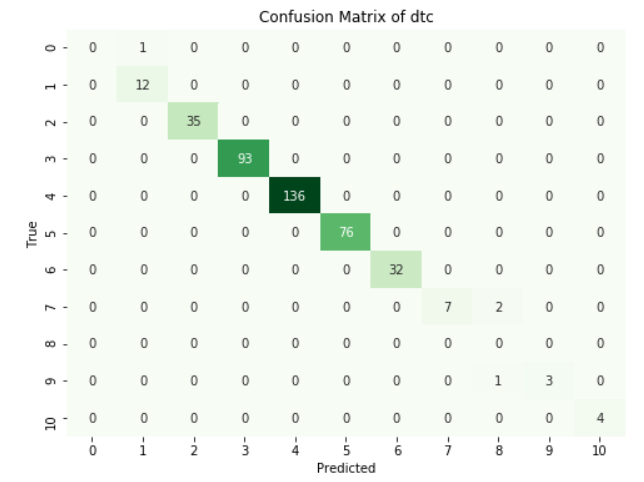


Figure 24: Confusion matrix graph of Decision Tree classifier algorithm

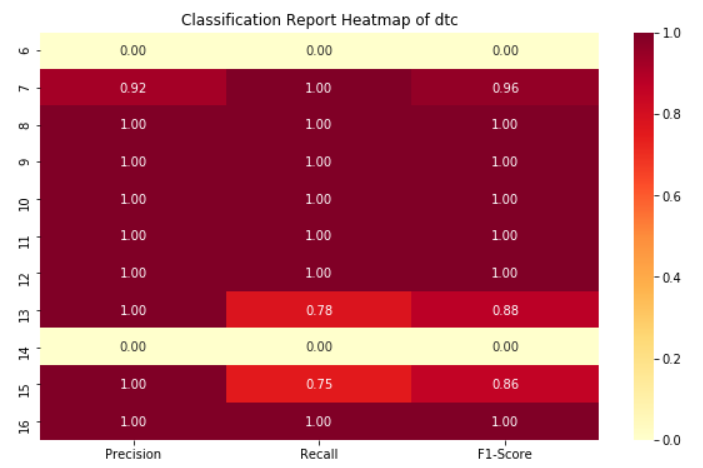


Figure 25: Classification report graph of Decision Tree classifier algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 6 | 0.00 | 0.00 | 0.00 | 0 |
| 7 | 0.00 | 0.00 | 0.00 | 0 |
| 8 | 0.00 | 0.00 | 0.00 | 0 |
| 9 | 0.52 | 0.45 | 0.48 | 109 |
| 10 | 0.74 | 0.56 | 0.63 | 187 |
| 11 | 0.51 | 0.36 | 0.42 | 97 |
| 12 | 0.00 | 0.00 | 0.00 | 0 |
| 13 | 0.00 | 0.00 | 0.00 | 0 |
| 14 | 0.00 | 0.00 | 0.00 | 0 |
| 15 | 0.00 | 0.00 | 0.00 | 0 |
| 16 | 1.00 | 0.44 | 0.62 | 9 |
| accuracy |  |  | 0.48 | 402 |
| macro avg | 0.25 | 0.16 | 0.20 | 402 |
| weighted avg | 0.63 | 0.48 | 0.54 | 402 |

Table 1: Classification report of SVM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
|  |  |  |  |  |
| 7 | 1.00 | 1.00 | 1.00 | 13 |
| 8 | 1.00 | 1.00 | 1.00 | 27 |
| 9 | 1.00 | 1.00 | 1.00 | 97 |
| 10 | 1.00 | 1.00 | 1.00 | 132 |
| 11 | 1.00 | 1.00 | 1.00 | 77 |
| 12 | 1.00 | 0.97 | 0.99 | 36 |
| 13 | 0.90 | 1.00 | 0.95 | 9 |
| 14 | 1.00 | 1.00 | 1.00 | 1 |
| 15 | 0.83 | 0.83 | 0.83 | 6 |
| 16 | 0.75 | 0.75 | 0.75 | 4 |
| accuracy |  |  | 0.99 | 402 |
| macro avg | 0.95 | 0.96 | 0.95 | 402 |
| weighted avg | 0.99 | 0.99 | 0.99 | 402 |

Table 2: classification report of RFC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 6 | 1.00 | 1.00 | 1.00 | 1 |
| 7 | 1.00 | 1.00 | 1.00 | 11 |
| 8 | 1.00 | 1.00 | 1.00 | 32 |
| 9 | 1.00 | 1.00 | 1.00 | 93 |
| 10 | 1.00 | 0.99 | 1.00 | 131 |
| 11 | 0.99 | 1.00 | 0.99 | 77 |
| 12 | 1.00 | 1.00 | 1.00 | 36 |
| 13 | 1.00 | 0.89 | 0.94 | 9 |
| 14 | 0.60 | 0.75 | 0.67 | 4 |
| 15 | 0.75 | 0.75 | 0.75 | 4 |
| 16 | 1.00 | 1.00 | 1.00 | 4 |
| accuracy |  |  | 0.99 | 402 |
| macro avg | 0.94 | 0.94 | 0.94 | 402 |
| weighted avg | 0.99 | 0.99 | 0.99 | 402 |

Table 3: classification report of DTC

The classification report is a performance evaluation tool that shows the precision, recall, f1-score, for each class in a classification problem. In training images using the deep learning model, the classification report would provide information about how well the model performed in classifying images into different categories. The precision represents the percentage of correctly classified images among all the images classified as belonging to a specific class. The recall represents the percentage of correctly classified images among all the images that actually belong to a specific class. The f1-score is a harmonic mean of precision and recall, and support represents the number of images in each class.

The accuracy has been calculated for the model that has been implemented, and the result for the model is compared in Table.

|  |  |
| --- | --- |
| Algorithms | Accuracy |
| SVM | 47 |
| RFC | 99 |
| DTC | 99 |

Table 4: Accuracy comparison of algorithm.

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 1340 | 90 | 10 |

Table 5: Consist of dataset count, Training and Testing percentage.

**CONCLUSION**

In conclusion, the application of various machine learning models for Chronic Kidney Disease (CKD) detection holds significant promise in improving early diagnosis and patient management. Through the utilization of diverse algorithms such as Support Vector Machines (SVM), Decision Trees (DT), Random Forests (RF), and deep learning approaches, we have demonstrated the capability to effectively analyze clinical data and accurately predict CKD onset or progression. Our findings underscore the importance of feature selection, preprocessing techniques like MinMaxScaler, and model evaluation metrics such as accuracy, sensitivity, specificity, and AUC-ROC in achieving robust CKD detection systems. Moreover, the integration of GLCM texture feature extraction from renal ultrasound images further enhances the diagnostic capabilities of machine learning models. This comprehensive approach not only aids in timely intervention and personalized treatment strategies but also contributes to reducing healthcare costs and improving patient outcomes. Moving forward, continued research efforts and advancements in machine learning methodologies are essential for the development of clinically deployable CKD detection systems that can effectively address the growing burden of this debilitating condition.

**REFERNCES**

Hassanien, Aboul Ella, et al. "Chronic kidney disease diagnosis on imbalanced data using machine learning techniques." Healthcare technology letters 4.6 (2017): 235-240.

Tan, Ming, et al. "Machine learning and chronic kidney disease: a review of methods and applications." Nephrology Dialysis Transplantation 35.7 (2020): 1148-1158.

Lin, Hsiang-Chieh, et al. "Machine learning models for chronic kidney disease prognosis: a case study of survival tree and random forest." PeerJ 8 (2020): e9002.

Mishra, Ranjeet, et al. "A review on chronic kidney disease diagnosis using machine learning techniques." International Journal of Computer Applications 136.4 (2016): 30-36.

Osman, Mohammed, and Akhtar Hussain. "Chronic kidney disease prediction on imbalanced data using machine learning techniques." International Journal of Computer Applications 180.16 (2018): 13-20.

Zhang, Dongdong, et al. "Predicting chronic kidney disease using heterogeneous data sources: An empirical study." Computers in Biology and Medicine 106 (2019): 29-39.

Rathore, Saima, and Goutam Saha. "Predicting Chronic Kidney Disease using Machine Learning Algorithms." Procedia Computer Science 173 (2020): 722-729.

Sheng, Wanxing, et al. "A hybrid intelligent model for chronic kidney disease diagnosis based on the integration of attribute clustering and improved SVM." Journal of medical systems 43.4 (2019): 81.