AN APPLICATION OF IMAGE PROCESSING AND MACHINE LEARNING FOR RICE VARIETIES CLASSIFICATION

PIYANART BOONRAMART

A Thesis Submitted in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Applied Mathematics
Suranaree University of Technology
Academic Year 2023

การประยุกต์ใช้การประมวลผลภาพร่วมกับการเรียนรู้เครื่อง
เพื่อจำแนกพันธุ์ข้าว

นางสาวปิยะนารถ บุญระมาตร

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาคณิตศาสตร์ประยุกต์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2566

AN APPLICATION OF IMAGE PROCESSING AND MACHINE LEARNING FOR RICE VARIETIES CLASSIFICATION

Suranaree University of Technology has approved this thesis submitted in partial fulfillment of the requirements for a Master's Degree.

Thesis Examining Committee

____

(Dr. Amornrat Suriyawichitseranee)

Chairperson

J.Tanthanuch

____

(Asst. Prof. Dr. Jessada Tanthanuch)

Member (Thesis Advisor)

____

(Assoc. Prof. Dr. Pisamai Kittipoom)

Member

Yupaporn

____

(Assoc. Prof. Dr. Yupaporn Ruksakulpiwat)

Vice Rector for Academic Affairs

and Quality Assurance

____

(Prof. Dr. Santi Maensiri)

Dean of Institute of Science

ปิยะนารถ บุญระมาตร : การประยุกต์ใช้การประมวลผลภาพร่วมกับการเรียนรู้เครื่องเพื่อจำแนกพันธุ์ข้าว (AN APPLICATION OF IMAGE PROCESSING AND MACHINE LEARNING FOR RICE VARIETIES CLASSIFICATION)
อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.เจษฎา ตัณฑนุช, 105 หน้า.

คำสำคัญ: การประมวลผลภาพ/การเรียนรู้เครื่อง/การจำแนกพันธุ์ข้าว

งานวิจัยนี้มีวัตถุประสงค์เพื่อเปรียบเทียบประสิทธิภาพของการจำแนกพันธุ์ข้าวจากภาพของเมล็ดข้าวสาร 5 สายพันธุ์ได้แก่ พันธุ์ข้าวคาราก้าดาก หอมมะลิ ยิปซาลา บาสมาติ และอาโบริโอ โดยการประยุกต์ใช้การประมวลผลภาพร่วมกับการเรียนรู้เครื่อง การดำเนินการวิจัยเริ่มจากการประมวลผลภาพเพื่อลดสัญญาณรบกวนของภาพเมล็ดข้าวสารสายพันธุ์ต่าง ๆ ที่บันทึกในรูปแบบแฟ้มเจเพ็กซึ่งเป็นภาพสีความละเอียด 250x250 จุดภาพ จำนวนสายพันธุ์ละ 15,000 ภาพ โดยได้ภาพจาก https://www.muratkoklu.com นำภาพที่ถูกลดสัญญาณรบกวนแล้วทั้งหมดมาประมวลผลภาพเพื่อใช้ในการจำแนก ด้วยเทคนิคที่แตกต่างกัน 7 วิธีได้แก่ การตรวจหาขอบภาพด้วยวิธีแคนนี การตรวจหาขอบภาพด้วยวิธีโซเบล การตรวจหาสัน การตรวจหาลายผิว การเพิ่มคุณภาพของภาพด้วยตัวกรองลาปลาซ การเพิ่มคุณภาพของภาพด้วยการพร่าเกาส์เซียน และการปรับฮิสโทแกรมให้เท่ากัน จากนั้นทำการสกัดคุณลักษณะด้านรูปร่าง 21 ชนิด และคุณลักษณะด้านลายผิวอีก 11 ชนิด แล้วนำไปจำแนกด้วยวิธีการเรียนรู้เครื่อง 5 วิธี ได้แก่ ต้นไม้ตัดสินใจ นาอีฟเบส์ เพื่อนบ้านใกล้ที่สุดเค ซัพพอร์ทเวกเตอร์แมชชีน และเกรเดียนท์บูตทรี ทั้งนี้ใช้วิธีการฝึกเพื่อการจำแนกเป็นการตรวจสอบไขว้เคโฟลด์เมื่อเคมีค่าเท่ากับ 10 สำหรับทุกวิธีการเรียนรู้เครื่อง ผลการวิจัยพบว่าการใช้การประมวลผลภาพการตรวจหาขอบด้วยวิธีโซเบลร่วมกับการจำแนกด้วยเทคนิคการเรียนรู้เครื่องซัพพอร์ทเวกเตอร์แมชชีน มีประสิทธิภาพในการจำแนกสูงที่สุด โดยการจำแนกมีค่าความแม่นร้อยละ 98.68 ความเที่ยงร้อยละ 98.67 ค่าเรียกคืนร้อยละ 98.67 ค่าคะแนนเอฟ-หนึ่งร้อยละ 98.67 และค่าสัมประสิทธิ์แคปปาของโคเฮนร้อยละ 98.35 แต่อย่างไรก็ตามในขั้นตอนตั้งแต่การประมวลผลภาพเพื่อการจำแนกไปจนถึงการจำแนกด้วยวิธีต่าง ๆ มีการใช้เวลาในการประมวลผลที่แตกต่างกัน โดยการใช้การประมวลผลภาพการเพิ่มคุณภาพของภาพด้วยการพร่าเกาส์เซียนร่วมกับการจำแนกด้วยเทคนิคการเรียนรู้เครื่องนาอีฟเบส์ใช้เวลาในการดำเนินการน้อยที่สุดคือ 3.99 วินาที และ การประมวลผลภาพการตรวจหาขอบด้วยวิธีโซเบลร่วมกับการจำแนกด้วยเทคนิคการเรียนรู้เครื่องเกรเดียนท์บูตทรี ใช้เวลาในการดำเนินการมากที่สุดคือ 9168.98 วินาที

สาขาวิชาคณิตศาสตร์และภูมิสารสนเทศ ลายมือชื่อนักศึกษา ปิยะนารถ
ปีการศึกษา 2566 ลายมือชื่ออาจารย์ที่ปรึกษา J.Tanthanuch

PIYANART BOONRAMART: AN APPLICATION OF IMAGE PROCESSING AND MACHINE LEARNING FOR RICE VARIETIES CLASSIFICATION. THESIS ADVISOR : ASST. PROF. JESSADA TANTHANUCH, Ph.D. 105 PP.

Keyword: IMAGE PROCESSING, MACHINE LEARNING, RICE VARIETY CLASSIFICATION

This research aims to compare the efficiency of techniques for classifying rice varieties from images of milled rice grains. Five rice varieties were considered: Karacadag, Jasmine, Ipsala, Basmati, and Arborio. Image processing combined with machine learning methods were applied. The procedure started with image processing to reduce noise from the images of rice grains of various varieties, which were color JPEG format images with a resolution of 250x250 pixels, with a total of 15,000 images per variety obtained from https://www.muratkoklu.com. All noise-reduced images were then processed for classification using seven different techniques: Canny edge detection, Sobel edge detection, ridge detection, texture detection, image enhancement with Laplacian filters, image enhancement with Gaussian blur, and histogram equalization. Features including 21 shape features and 11 texture features were extracted and classified using five machine learning techniques: decision trees, Naïve Bayes, k-Nearest Neighbors, Support Vector Machines (SVMs), and gradient boosted trees. Training was conducted with K-fold cross-validation with K=10 for all machine learning techniques. The research findings showed that using image processing with Sobel edge detection combined with classification using SVMs was the most effective method, with classification accuracies of 98.68%, precision of 98.67%, recall of 98.67%, F1-score of 98.67%, and a Cohen's kappa coefficient of 98.35%. However, the processing time varied significantly among the different processing steps, with the combination of Gaussian blur image enhancement and classification using Naïve Bayes requiring the least time (3.99 seconds), and the combination of Sobel edge detection image processing and classification using Gradient Boosted Trees requiring the most time (9168.98 seconds).

School of Mathematical Sciences and Geoinformatics

Academic Year 2023

Student's Signature ______________

Advisor's Signature ______________

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Professor Jessada Tanthanuch, my thesis advisor, for his guidance and support throughout this research. His expertise and advice were invaluable in helping me to complete this thesis.

I would also like to thank my family and friends for their encouragement and support. Their belief in me helped me to stay motivated throughout this challenging process.

Finally, I would like to thank Suranaree University of Technology for providing me with a scholarship that allowed me to pursue my graduate studies. This scholarship gave me the opportunity to focus on my research and to complete this thesis.

I am grateful for the support of all of these individuals and organizations. Their contributions made this thesis possible.

Piyanart Boonramart

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

CONTENTS (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In Thailand, the delicate white grains of rice are not just a humble staple food, but the very lifeblood of the nation. Rice cultivation runs deep in Thai history and culture, and its export forms a cornerstone of the nation's economic prosperity. In 2023, Thailand proudly sent over 8.8 million tons of rice across the globe, generating a staggering 178 billion baht in revenue (Thai Rice Exporters Association, 2024). This remarkable figure underscores the immense importance of rice to the Thai economy, providing livelihoods for countless farmers, fueling diverse industries, and driving national growth. However, ensuring rice exports meet the stringent quality and standards of importing countries is crucial for maintaining this economic engine.

Due to the different varieties of rice, there are varying prices, as shown in Table 1.1. This is where meticulous export standards come into play, meticulously crafted by the Department of Agriculture to encompass physical, chemical, and microbiological aspects of the precious grain. Furthermore, accurate classification of rice varieties is paramount, as each distinct type carries its own value. From the sought-after aroma of Jasmine rice to the versatility of Hom Mali, precise identification determines its rightful place in the export ladder.

Presently, computer vision, image processing, artificial intelligence (AI) and machine learning (ML) play a significant role in our daily lives, leading to increased convenience. These technologies are being applied across various domains, assisting in agricultural management planning, designing and analyzing operations to maximize agricultural production efficiency. Additionally, they are utilized for image analysis of agricultural produce, aiding in crop planning and harvesting. Emerging technologies like AI and ML are revolutionizing rice classification, providing unparalleled accuracy and speed. Their capacity to analyze grain characteristics using image and video data streamlines the sorting process, reducing human effort and costs. The potential of AI and ML in the Thai rice

Table 1.1 Price of Rice per Metric TON (Thai Rice Exporters Association, 2024).

Type	**Price (US Dollar/MT)**
White Rice	
Thailand 5% broken	655
Vietnam 5% broken	639-643
Pakistan 5% broken	637-641
Thailand 25% broken	617
Vietnam 25% broken	612-616
Pakistan 25% broken	585-589
Fragrant Rice	
Thailand Hommali 100%	883
Vietnam Jasmine	715-719
Pakistan basmati 2% broken	950

industry is immense, offering improvements in quality control, efficiency, and ultimately enhancing national competitiveness. However, it is a race against time as Thailand grapples with challenges such as global competition, volatile prices, and the persistent threat of climate change. Understanding the multifaceted importance of rice to the Thai economy and exploring the innovative solutions like AI and ML classification becomes vital as we navigate the future of this precious resource.

Based on the discussion above, this thesis aims to explore how the application of image processing, coupled with machine learning, can effectively classify 5 types of rice grains: Arborio, Basmati, Ipsala, Jasmine, and Karacadag, based on photographs of individual rice grains. The study seeks to determine the most efficient approach in terms of classification accuracy and processing time. The results of this research endeavor can potentially enhance the efficiency of rice grain classification, thereby contributing to further advancements in this field. This journey to secure Thailand's place as a global rice leader demands not only continued technological advancements but also a deep appreciation for the cultural and economic significance of this humble, yet mighty grain.

1.1 Research Objective

1. To apply mathematics, combined with image processing and machine learning algorithms, to classify rice varieties from images.

2. To evaluate the performance of the proposed method for classification.

1.2 Scope and Limitations

1. The data set used in this study was publicly available data from muratkoklu of Dr.Murat Köklü, retrieved from https://www.muratkoklu.com/datasets.

2. The features used for image processing of grain rice images are Sobel Edge Detection, Canny Edge detection, Ridge Detection, Texture Detection, Equalization Histogram, Enhance image by using Laplacian filter, Enhance image by using Gaussian Blur filter.

3. The techniques for solving the classification problem in this study consist of the Decision Tree, Naïve Bayes, K-Nearest Neighbors, Support Vector Machine, Gradient Boosted Tree.

4. Use Python language program version 3.11.1 to process images, extract features and create the classification models and evaluate the performance of the models, working on Lenovo DESKTOP-A3APD4J, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71 GHz, 4GB RAM with Microsoft Windows 10 Operating System, and NB109-2565-052 HP Probook 440 G8 Notebook PC, 11th Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 2.42GHz with Microsoft Windows 11 Operating System.

1.3 Research Procedure

The research work proceeds as follows:

1. Study the mathematical knowledge of the features used in image processing to apply them to classification models.

2. Study machine learning and classification algorithms.
3. Perform image processing with features by Python language program for use in classification.
4. Put the images obtained from the image processing into the Python program to classify the rice varieties and find the performance of the model.

1.4 Expected Result

Achieve a high-performance model through the integration of image processing and machine learning techniques, enabling the classification of rice varieties from images of milled rice grains.

CHAPTER II

LITERATURE REVIEW

This chapter provides an overview of the basic concepts of digital image processing and its methods. Including the concept of machine learning. Its techniques and performance indicators of classification models.

2.1 Digital Image Processing

Digital image processing is the manipulation and analysis of digital images using various algorithms and techniques to extract information, enhance quality, or perform specific tasks. It involves acquiring digital images through sensors or cameras, preprocessing them to remove noise or artifacts, and applying operations such as filtering, edge detection, segmentation, and feature extraction to achieve desired results. Digital image processing finds applications in fields such as medicine, remote sensing, surveillance, computer vision, and multimedia.

2.1.1 Edge Detection

There are various methods for edge detection; however, in this study, we are particularly interested in Sobel Edge Detection and Canny Edge Detection. Both methods are popular for their straightforward algorithms and high efficiency in edge detection.

- **Sobel Edge Detection**

  The Sobel Edge Detection method detects the edge of an image using two 3×3 templates (Wikipedia, 2024). If we define A as the source image, the horizontal difference (G_x), and vertical difference (G_y) are as follows:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A, \tag{2.1}$$

where $*$ denotes the 2-dimensional signal processing convolution operation.

Find the magnitude gradient:

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{2.2}$$

and the gradient direction is

$$\theta = \arctan\left(\frac{G_y}{G_x}\right). \tag{2.3}$$

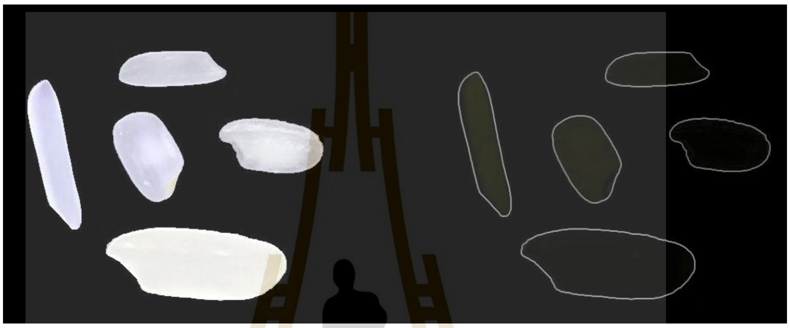


Figure 2.1 Example of Sobel Edge Detection.

- **Canny Edge Detection**

The Canny edge detection operator was developed by John F. Canny in 1986 and uses a multi-stage algorithm to detect a wide range of edges in images (Reddy et al., 2016).

The steps of Canny edge detection algorithm are as follows:

1. Removing the noise by applying a Gaussian filter, which Gaussian filter formula can write as below:

$$G(x, y) = \frac{1}{2\pi\sigma^2}(e^{-\frac{x^2+y^2}{2\sigma^2}}),$$

where x is the variable on the x-axis, y is the variable on the y-axis, and σ is the deviation.

2. Find the gradient of the image.
3. Find the gradient magnitude (2.2) and the direction of the edge same as Sobel edge detection (2.3).

4. Remove pixels that are not considered part of the edge.
5. Track the edge by hysteresis that rejects the edge pixel which is weak and not connected to the strong edge pixel.

2.1.2 Ridge Detection

Ridge detection, in the context of image processing, is the technique of identifying and locating linear features in an image that resemble ridges, like the prominent lines or elongated structures within the image. It is distinct from edge detection, which aims to find abrupt changes in intensity between adjacent pixels, as ridges often have gradual intensity variations along their course (Shokouh et al., 2021).

Ridge detection with adaptive thresholding is a method that aims to detect ridges (or edges) in an image by applying a threshold that varies across different regions of the image, and the steps of ridge detection with adaptive thresholding algorithm are as follows:

1. Preprocessing:
   Convert the input image to grayscale if it is not already in grayscale.

2. Gradient Calculation:
   The gradient magnitude $G(x, y)$ and gradient direction $\theta(x, y)$ of the image using a suitable edge detection operator, such as Sobel or Prewitt operators.

3. Compute Local Threshold:
   For each region, a local threshold is calculated based on the statistical properties of pixel intensities within that region. Common statistical measures used for threshold calculation include the mean, median, or standard deviation. The goal is to set a threshold that is sensitive to the local characteristics of the image, allowing for better detection of ridges or edges across different regions.

4. Apply Adaptive Thresholding:
   The key of this step is to dynamically determine a threshold for each pixel based

on its local neighborhood.Different methods exist such as hysteresis thresholding, mean subtraction with offset, Gaussian weighted mean.

5. Post-processing:

   Optionally,apply common techniques include morphological operations like dilation or erosion to refine the detected ridges or edges.

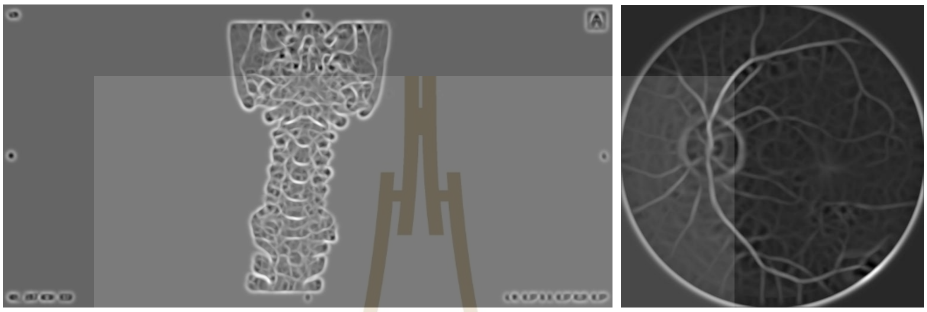


Figure 2.2 Example of Ridge Detection Done by Python code.

2.1.3 Texture Detection

Texture detection is an intriguing automated process that goes beyond identifying color or intensity variations within an image. It involves extracting essential details about the repetitive structures and arrangements that define the unique textural characteristics of a surface. This capability opens up diverse applications, including medical imaging for detecting abnormalities, remote sensing for classifying land cover in satellite imagery, industrial inspection for quality control, content-based image retrieval for finding visually similar images based on texture, and robot vision for guiding interactions with objects based on texture cues.

At the heart of texture detection, Gabor filters, named after Dennis Gabor, Gabor filters act as the workhorses in this process, providing a symphony of frequency and orientation information that enhances the precision of texture detection algorithms across various applications. A Gabor filter, employed in image processing, serves various purposes such as edge detection, texture analysis, feature extraction, and disparity estimation. Functioning as a bandpass filter, it selectively permits frequencies within a specified range to pass through while suppressing others. This characteristic makes it well-suited for scrutinizing specific features in an image without being inundated by extraneous information.

Conceptually, a Gabor filter can be envisioned as a sinusoidal wave, representing the desired frequency, modulated by a Gaussian function, which signifies localization. This amalgamation enables the filter to respond to particular frequencies within a confined area of the image. By adjusting the parameters of the sinusoidal and Gaussian components, we can craft Gabor filters with diverse characteristics (Shah, 2018).

2.1.4 Histogram Equalization

Histogram equalization is an image processing method employed to enhance contrast by expanding the intensity range. The objective is to achieve a balanced spread of pixel intensities, using all available brightness levels. This is achieved by applying a custom function that translates each pixel's original brightness to a new one (Nikhil, 2023).

Let $H(i)$ be the histogram of the image, where i is in the range [0,L] (the intensity levels) and let n be the total number of the pixels in the image, the histogram equalization basic algorithm involves the following step:

1. Compute the histogram of the input image. The histogram represents the distribution of intensity values in the image.

2. Calculate the cumulative distribution function (CDF) from the histogram. The CDF represents the cumulative sum of histogram values,

$$\text{CDF}(i) = \sum_{j=0}^{i} H(j).$$

3. Normalize the CDF to map the values to the range [0, L-1], where L is the number of intensity levels,

$$\text{CDF}_{\text{norm}}(i) = \left\lfloor \frac{\text{CDF}(i) - \text{min(CDF)}}{n-1} \times (L-1) + 0.5 \right\rfloor,$$

where min(CDF) is the minimum non-zero value of the cumulative histogram and L is the number of intensity levels.

4. Map Intensity Values: For each pixel in the input image, replace its intensity value with the corresponding normalized CDF value,

$$I_{\text{equalized}}(x, y) = \text{CDF}_{\text{norm}}(I(x, y)),$$

where $I(x, y)$ is the intensity value of the pixel at position (x, y) in the image.

2.1.5 Image Enhancement

Image enhancement refers to a set of processes aimed at improving its overall quality and visual appeal. This can involve various techniques depending on the type of image and the desired outcome. Some filters for image enhancement are presented as the following:

1. **Laplacian filter**

   The Laplacian filter, classified as a second-order derivative filter, assesses the rate of change of the first derivative within an image. To put it more plainly, it accentuates regions where neighboring pixel intensity values experience swift alterations, rendering it a potent instrument for identifying edges (NV5 Geospatial Software, 2023).

   To illustrate, envision rolling a marble across an image. It would seamlessly traverse areas characterized by gradual intensity changes but encounter obstacles at sharp edges, unveiling their precise locations. In a comparable manner, the Laplacian filter operates by pinpointing intensity “bumps” that serve as indicators of edges.

2. **Gaussian Blur**

   Gaussian blur is a fundamental image processing technique used to reduce noise and soften harsh edges, often serving as a pre-processing step for various image analysis tasks (Deng and Cahill, 1993).

2.2 Machine Learning

Machine learning (ML) is the operation the computer system uses the data for learning by itself with the aim of detecting relationships within the data by computer. It uses programmed algorithms that receive and analyze input data to predict output values within an acceptable range. As new data is fed to these algorithms, they learn and optimize their operations to improve performance, developing 'intelligence' over time. ML is separated into 4 categories, which are supervised learning, unsupervised learning, semi-supervised, and reinforcement.

Supervised learning is a popular method in machine learning. This operator provides the machine learning algorithm with a known dataset that includes desired inputs and outputs, and the algorithm must find a method to determine how to arrive at those inputs and outputs. While the operator knows the correct answers to the problem, the algorithm identifies patterns in data, learns from observations, and makes predictions. The algorithm makes predictions that are corrected by the operator, and this process continues until the algorithm achieves a high level of accuracy/performance. Supervised learning can solve regression, classification, and forecasting problems (Wakefield, 2022).

2.3 Machine Learning Algorithms for Classification

2.3.1 Decision Tree

A decision tree (DT) is a popular machine learning algorithm used for both classification and regression tasks. It is a tree-like model where each internal node represents a decision based on a specific feature, each branch represents the outcome of the decision, and each leaf node represents the final decision or the target variable. The goal of a decision tree is to recursively split the dataset into subsets based on the most significant features, ultimately creating a tree structure that can be used for making predictions.

The components of a decision tree include the root node, internal nodes, branches, and leaf nodes. The root node is the topmost node that represents the initial decision based on the most significant feature. Internal nodes represent decisions based on features, branches represent the possible outcomes of the decisions, and leaf nodes represent the final predicted values or classes.

Decision trees use various splitting criteria to determine the best feature to split on at each internal node. Two commonly used criteria are information gain and gain ratio. The information gain of dataset S is calculated using the following formula:

$$\begin{aligned}\text{Information Gain }(S) &= \text{Entropy of } T - \text{Mean Information Requirement}\\ &= -\sum_j p_j \log_2(p_j) - \sum_{i=1}^{k} P_i H_S(T_i), \end{aligned} \tag{2.4}$$

where p_j is the proportion of members in class j relative to the total number of members in a sample class, P_i is the proportion of instances in the ith sub-dataset, $H_S(T_i)$ is Entropy before classifier of S by the ith subset of the training dataset T.

Information Gain measures the reduction in entropy or surprise by splitting a dataset according to a given value of a random variable. To normalized information gain, we will use gain ratio and the gain ratio formula is as follow:

$$\text{Gain Ratio} = \frac{\text{Information Gain}}{\text{Entropy}}. \tag{2.5}$$

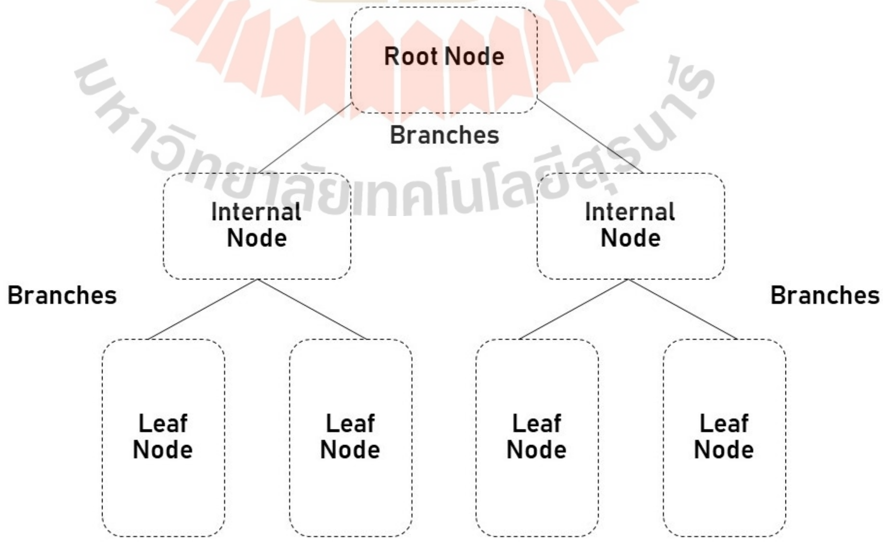


Figure 2.3 The components of Decision Tree.

2.3.2 Naïve Bayes

Naive Bayes (NB) is a probabilistic machine learning algorithm for classification that works well with both binary (two-class) and multiclass (more than two classes) problems. It's often praised for its simplicity, efficiency, and effectiveness in situations with high-dimensional data (Farid et al., 2014).

The mathematical foundation of Naive Bayes for multiclass classification relies on Bayes' theorem and the assumption of feature independence and for multiple classes (c_1, c_2, ..., c_k) where c_i represents the ith class in a set of possible classes, the posterior probability for each class is calculated based on the formula:

$$P(c_i \mid x) = \frac{P(c_i)P(x \mid c_i)}{P(x)},$$

where $P(c_i \mid x)$ is Posterior probability of class c_i given features x,

$P(c_i)$ is the prior probability of class c_i,

$P(x)$ is the prior probability of observing features x.

2.3.3 K-Nearest Neighbors

K-Nearest Neighbors (K-NN), a non-parametric, instance-based classification method, is suitable for diverse data types (Wang et al., 2023). In multi-class situations, it determines the class label of a new data point by aggregating the majority vote from its K nearest neighbors within the training dataset.

Algorithms of K-Nearest Neighbors are as following:

1. Let training data D = { (x_1, y_1) | i =1, ..., n }, where x_i is a data point in the feature space and y_i is the class label corresponding x_i and x_{new} as a new data point.

2. For each training data point x_i, calculate the distance $d(x_{new}, x_i)$ using the distance metric (usually Euclidean distance).

3. Sort the training data points based on their distances to x_{new} in ascending order and select the K closest points as the neighbors.

4. Count the frequency of each class label among the K neighbors and assign the class label with the highest frequency to x_{new}.

5. predicted class label for x_{new}.

2.3.4 Support Vector Machine

Support Vector Machines (SVM) are versatile supervised learning models exceling at classification tasks (Madzarov et al., 2008). In binary classification, the input space is denoted by X, and the binary class labels, represented as either 1 or -1, are denoted by Y (Cortes and Vapnik, 1995), the equation of the hyperplane separating the classes can be written as:

$$w^T \cdot x + b = 0,$$

where w refers to the weight vector, b refers to the distance of the hyperplane from the origin along the normal vector w, which y and w satify the following inequality:

$$y_i(w^T \cdot x_i + b) \geq 1, \text{where } i = 1, ..., n.$$

The distance between a data point x_i and the decision boundary can be written as:

$$d_i = \frac{w^T \cdot x_i + b}{\| w \|},$$

where $\| w \|$ refers to the Euclidean norm of the weight vector w.

In SVM, the objective function aims to maximize the margin between the decision boundary (hyperplane) and the support vectors while minimizing the classification error. This can be formulated as the following constrained optimization problem:

$$\text{min}_{w,b} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \text{max}\{0, 1 - y_i(w^T \cdot x_i + b)\}$$

subject to $y_i(w^T \cdot x_i + b) \geq 1$, where C is the regularization parameter controlling the trade-off between maximizing the margin and minimizing the classification error.

The training process involves solving this optimization problem to find the optimal hyperplane parameter w and b. For prediction, SVM evaluates the sign of the decision function as

$$f(x) = \text{sign}(w^T \cdot x + b),$$

where $f(x)$ represents the decision function. The sign of $f(x)$ determines the class label for a new data point x.

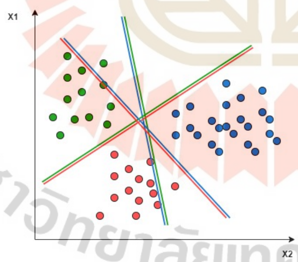
When it comes to multiclass classification, SVMs offer two main strategies:

1. One-vs-One (OvO)

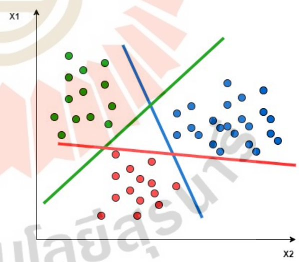
   The OvO approach is a multi-class classification strategy that leverages binary classification algorithms. In this approach, for a dataset with N classes, $\frac{N(N-1)}{2}$ individual binary classifiers are trained. Each classifier is trained to distinguish between one specific class and all other classes combined.

2. One-vs-All (OvA)

   In N-class problems (where N is greater than 2), multiple sets of binary classifiers called SVMs are built. Each SVM is trained to recognize one class against all others. During recognition, a test example is given to all these SVMs, and it is assigned the label of the class with the highest confidence score among all classifiers.



(a) One-vs-One approach.



(b) One-vs-All approach.

Figure 2.4 Support Vector machine for Multi-class clssification.

source: https://www.baeldung.com/cs/svm-multiclass-classification

In summary, SVMs for multiclass classification employ strategies like OvO or OvA to extend binary classification to multiple classes. The mathematical foundation involves finding hyperplanes that effectively separate different classes in feature space, and the choice between OvO and OvA depends on factors such as simplicity and computational efficiency.

2.3.5 Gradient Boosted Tree

Gradient Boost Tree (GBT) (Natekin and Knoll, 2013) is a machine learning technique for classification and regression that produces a strong learning model from the combination of multiple weak learning models, which are typically decision trees. All trees are connected in series. And each tree attempts to minimize errors or residuals of the previous tree. That is, we want to reduce the loss function. The final model takes the results of each step to make it effective for the learning model. This makes this algorithm highly accurate.

In the gradient boosted tree algorithm, Friedman's Gradient Boosted algorithm is employed. The input dataset is denoted as $(x_i, y_i)_{i=1}^{n}$, where n represents the number of samples, and it undergoes Mth iterations. The weak learning model is represented by $F(x)$, and the loss function is denoted as $L(y, F(x))$.

Algorithms of Gradient Boosted Tree are as following:

1. Initialize $F_0(x)$ with a constant, where γ is the constant value being optimized for, and

$$F_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} L(y_i, \gamma). \tag{2.6}$$

2. For m = 1,...,M

   (a) Calculation for pseudo-residual:

$$r_{i,m} = -[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)}, \qquad i = 1, ..., n. \tag{2.7}$$

   (b) Prepare new data $\{x_i, r_{i,m}\}_{i=1}^{n}$ and build $R_{j,m}$, for $i = 1, 2, ..., m$.

   (c) For $j = 1, ..., J_m$,

$$\gamma_{j,m} = \text{argmin}_\gamma \sum_{x_i \in R_{j,m}} L(y_i, F_{m-1}(x_i) + \gamma). \tag{2.8}$$

   (d) Adjust the model:

$$F_m(x) = F_{m-1}(x) + v \sum_{j=1}^{J_m} \gamma_{j,m} I, \qquad x \in R_{j,m} \tag{2.9}$$

   where v is learning rate, I is indicator function.

3. The result will be in form $F_M(x)$.

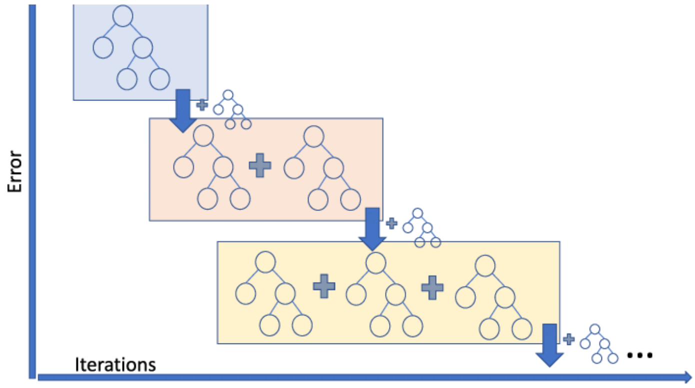


Figure 2.5 Gradient Boost Tree.

source: https://pub.towardsai.net/gradient-boosting-technique-b3dbb7069b74

2.4 Performance indicators of classification model

2.4.1 Confusion Matrix

A confusion matrix is a table that shows the performance of a classification model by comparing its predictions to the actual values. It is a useful tool for visualizing the model’s performance and understanding the types of errors it makes. The performance of a classification algorithm is summarized by indicating the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) predictions.

1. True Positives (TP) refer to the cases where the model predicted the class correctly, and the actual class is also that class.

2. True Negatives (TN) refer to the cases where the model predicted the class correctly, and the actual class is not that class.

3. False Positives (FP) refer to the cases where the model predicted the class incorrectly as positive, when it is actually negative.

4. False Negatives (FN) refer to the cases where the model predicted the class incorrectly as negative, when it is actually positive.

In a multi-class classification problem, the confusion matrix becomes a square matrix, where each row and column corresponds to a class, and the elements represent the counts of true positives, true negatives, false positives, and false negatives for each class (Grandini et al., 2020).

Let the confusion matrix is a $N \times N$ matrix where N is the number of different class labels c_i (i = 1, 2, ..., N).

2.4.2 Accuracy

Accuracy is a basic indicator. It is the overall percentage of correct predictions.

In multi-class classification, the accuracy can be calculated by considering the accuracy of each class and the overall accuracy (Grandini et al., 2020), which can be calculated using the following formula:

$$\text{Percent of Accuracy}_{c_i} = \frac{\text{TP}_{c_i} + \text{TN}_{c_i}}{\text{TP}_{c_i} + \text{TN}_{c_i} + \text{FP}_{c_i} + \text{FP}_{c_i}} \times 100. \tag{2.10}$$

For overall, the accuracy can be calculated as

$$\text{Percent of Accuracy} = \frac{\sum_{i=1}^{N} \text{TP}_{c_i}}{\sum_{i=1}^{N} (\text{TP}_{c_i} + \text{TN}_{c_i} + \text{FP}_{c_i} + \text{FN}_{c_i})} \times 100, \tag{2.11}$$

where:

Accuracy$_{c_i}$ is the accuracy of class c_i,

Accuracy is the overall accuracy,

TP$_{c_i}$ is the number of true positives for class c_i,

TN$_{c_i}$ is the number of true negatives for class c_i,

FP$_{c_i}$ is the number of false positives for class c_i,

FN$_{c_i}$ is the number of false negatives for class c_i,

N is the total number of classes.

2.4.3 Precision

Precision is the ratio of correctly predicted positive observations to the total predicted positives. High precision means your model rarely makes false positives, which is crucial when false positives have high costs.

In multi-class classification, the precision can be calculated by considering the precision of each class and the overall precision (Grandini et al., 2020), which can be calculated using the following formula:

$$\text{Percent of Precision}_{c_i} = \frac{\text{TP}_{c_i}}{\text{TP}_{c_i} + \text{FP}_{c_i}} \times 100. \tag{2.12}$$

For overall, the precision can be calculated as

$$\text{Percent of Precision} = \frac{\sum_{i=1}^{N} \text{TP}_{c_i}}{\sum_{i=1}^{N} (\text{TP}_{c_i} + \text{FP}_{c_i})} \times 100, \tag{2.13}$$

where:

Precision$_{c_i}$ is the precision of class c_i,

Precision is the overall precision.

2.4.4 Recall

Recall is the ratio of correctly predicted positive observations to all actual positives. High recall means the model captures most of the relevant cases, important when missing positives is costly.

In multi-class classification, the recall can be calculated by considering the recall of each class and the overall recall (Grandini et al., 2020), which can be calculated using the following formula:

$$\text{Percent of Recall}_{c_i} = \frac{\text{TP}_{c_i}}{\text{TP}_{c_i} + \text{FN}_{c_i}} \times 100. \tag{2.14}$$

For overall, the recall can be calculated as

$$\text{Percent of Recall} = \frac{\sum_{i=1}^{N} \text{TP}_{c_i}}{\sum_{i=1}^{N} (\text{TP}_{c_i} + \text{FN}_{c_i})} \times 100, \tag{2.15}$$

where:

Recall_{c_i} is the recall of class c_i,

Recall is the overall recall.

2.4.5 F1-score

F1-score or F-measure is a harmonic mean of precision and recall, balancing both aspects. It provides a single score that balances precision and recall, which can be useful when there is an uneven class distribution.

In multi-class classification, the F1-score can be calculated by considering the F1-score of each class and the overall F1-score (Grandini et al., 2020), which can be calculated using the following formula:

$$\text{Percent of F1-score}_{c_i} = \frac{2 \times \text{Precision}_{c_i} \times \text{Recall}_{c_i}}{\text{Precision}_{c_i} + \text{Recall}_{c_i}} \times 100. \tag{2.16}$$

For overall, the F1-score can be calculated as

$$\text{Percent of F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \times 100, \tag{2.17}$$

where:

F1-score_{c_i} is the F1-score of class c_i,

F1-score is the overall F1-score.

2.4.6 Cohen's Kappa Coefficient

Cohen's Kappa coefficient, often referred to as simply "Kappa" is a statistical measure that assesses the level of agreement between two raters or more raters classifying items into categories. It accounts for the possibility of agreement occurring by chance and provides a more robust evaluation of inter-rater reliability than simple percent agreement. Kappa (K) is a value between -1 and 1 which negative values imply less agreement than chance (Cohen, 1960).

The formula for Cohen's Kappa is as follows:

$$\text{Percent of } K = \left(\frac{P_0 - P_e}{1 - P_e}\right) \times 100, \tag{2.18}$$

where P_0 is the observed agreement between raters,

P_1 is the expected agreement between raters.

Table 2.1 Interpretation of Cohen's Kappa (McHugh, 2012)

Value of Kappa (%)	Level of Agreement	% of the data that are Reliable
0-20	None	0-4
21-39	Minimal	4-15
40-59	Weak	15-35
60-79	Moderate	35-63
80-90	Strong	64-81
Above 90	Almost Perfect	82-100

2.5 K-fold Cross Validation

K-fold cross-validation is a widely employed method to assess the efficiency of machine learning models, aiming to gauge their ability to generalize to new and unseen data (Brownlee, 2023).

The process unfolds as follows:

1. Divide the data into K roughly equal-sized folds.

2. For each fold:

   Train the model on the data in K-1 folds (training set).

3. Evaluate the model's performance on the remaining fold (test set).

4. Calculate the average performance metric across all K folds. This provides a more robust estimate of the model's generalization error than a single split of training and testing data.

2.6 Related Research

Aki, Güllü, and Uçar (2015) proposed a method to classify rice grains into four types, namely Baldo, Osmancik, Yesemin, and broken grain. This study uses image processing combined with 13 techniques of machine learning, that is Nearest Neighbor with Generalization, Decision Tree with Naïve Bayes, Normalized Gaussian Radial Basis Function Network, KStar (Instance-based classifier), Best-First Decision Tree, Bagging, Random Forest, J48, IB1 (Nearest-Neighbour classifier), IBk (K-Nearest Neighbours classifier), JRip (Propositional Rule Learner, Repeated Incremental Pruning to Produce Error Reduction) and Naïve Bayes. Starting with extracting features related to geometric shapes from each grain image. Each grain has six features and then trains the features using machine learning techniques. The technique that gave the highest accuracy was Nearest Neighbor with Generalization, where the average real-time accuracy was calculated as 90.5%.

Zareiforoush, Minaei, Alizadeh, and Banaka (2016) proposed the use of computer vision as a feature extraction method and feature selection, combined with the metaheuristic method. Four types of milled rice grains were analyzed: high-processed sound grains, high-processed broken grains, low-processed sound grains, and low-processed broken grains. The four metaheuristic methods are artificial neural networks, support vector machines, decision trees, and Bayesian networks. The technique that gives the highest accuracy is ANN, with an accuracy of 98.72%.

Rexce and Usha Kingsly Devi (2017) demonstrated the classification of thirteen types of rice grains through a computer vision system utilizing image acquisition, image preprocessing, and segmentation methods. Feature extraction was then employed to extract 57 features from each rice grain image. The metaheuristic techniques utilized included artificial neural networks, support vector machines, Bayesian networks, and decision trees, each achieving classification accuracies of 92.307%, 90.384%, 82.629%, 59.615% respectively.

Cinar and Köklü (2019) proposed the identification of two rice cultivars, Osmancik and Cameo species, from 3,810 images based on seven morphological features: area, perimeter, major axis length, minor axis length, grain distortion, surface eccentricity, convex area, and the ratio of rice shape area to the frame of the considered image. Logistic Regression, Multi-Player Perceptron, Support Vector Machine, Decision Tree, Random Forest, Naïve Bayes, and K-Nearest Neighbors achieved accuracies of 93.02%, 92.86%, 92.83%, 92.49%, 92.39%, 91.71%, and 88.58%, respectively.

Cinar and Köklü (2021) utilized various statistical methods, such as analysis of variance (ANOVA), the Chi-squared method, and the gain ratio method, to identify effective features extracted from images for the purpose of improving rice variety classification. The study involved analyzing 15,000 images of each rice variety (Karacadag, Jasmine, Ipsala, Basmati, and Arboio), totaling 75,000 images. From these images, a total of 106 features were extracted, including 12 morphological features, 4 shape features, and 90 color features.

Cinar, Köklü, and Taspinar (2021) conducted a study on the classification of rice varieties. They developed Python programs to apply machine learning algorithms, artificial neural network algorithms, and deep neural networks for identifying rice varieties using the 106 features extracted from the dataset. Their approach was compared with the

Convolutional Neural Network method for characterizing and classifying rice grains from images. The study revealed that employing the Convolutional Neural Network resulted in higher performance.

Cinar and Köklü (2022) conducted a classification study involving five rice varieties: Karacadag, Jasmine, Ipsala, Basmati, and Arborio. They employed seven machine learning techniques, namely Logistic Regression, Multilayer Perceptron, Support Vector Machine, Decision Tree, Random Forest, Naïve Bayes, and K-Nearest Neighbor, using MATLAB programs to classify rice based on data. The classification was performed on 106 features of all five rice varieties, with each feature set categorized as follows: morphological features, morphological and shape features, color features, and all features combined. The highest accuracy achieved was 99.91%, obtained by Multilayer Perceptron when using the feature set comprising all features.

CHAPTER III

RESEARCH METHODOLOGY

This chapter presents the steps used in image processing. Including extracting features from images. as well as modeling and classification techniques. The procedure consists of 6 steps:

1. Data collection;
2. Reduce background noise;
3. Image processing;
4. Feature extraction and Normalization of dataset;
5. Machine Learning modeling;
6. Evaluating the performance of the model.

3.1 Data Collection

The dataset used in the study of rice grain variety classification was obtained from https://www.muratkoklu.com. It is a dataset called Rice Image Dataset which consists of 75,000 images of rice grains. Each image has a resolution of 250x250 pixels. The dataset includes images of 5 different rice varieties: Karacadag, Jasmine, Ipsala, Basmati, and Arborio. There are 15,000 images for each variety. The images of each variety are stored in a separate subfolder, for a total of 5 subfolders.

(a) A Arborio rice grain. (b) A Basmati rice grain. (c) A Ipsala rice grain.

(d) A Jasmine rice grain. (e) A Karacadag rice grain.

Figure 3.1 A collection of five rice varieties.

3.2 Reduce Background Noise

After the data collection process, it was found that each rice grain image had noise around the grains, as shown in figure 3.2.

Figure 3.2 Noise in the background of rice grain image.

Therefore, the noise was removed by cropping the images to only include the rice grains.The cropped rice grain images were then placed on a black background of 250x250 pixels, which is the same size as the original images, using a Python program on Jupyter Notebook.

3.3 Image Processing

In this step, 75,000 images of rice grains in the folder will be processed using the following methods: Canny Edge Detection, Sobel Edge Detection, Ridge Detection, Texture Detection, Histogram Equalization, Laplacian Filter Enhancement, and Gaussian Blur Enhancement. The goal was to extract the edges and details of the rice grains in each image using the Python programming language on Jupyter Notebook.

The main library used for image processing is OpenCV. The main functions used for image processing in each method are shown in Table 3.1.

Table 3.1 The main functions for each image processing method.

Image processing methods	Main functions
Canny Edge Detection	cv2.Canny()
Sobel Edge Detection	cv2.Sobel()
Ridge Detection	filters.apply_hysteresis_threshold()
Texture Detection	cv2.getGaborKernel()
Histogram equalization	cv2.equalizeHist()
Enhancement by Laplacian filter	cv2.Laplacian()
Enhancement by Gaussian Blur	cv2.GaussianBlur()

3.4 Feature Extraction and Normalization of Dataset

In this step, we used a Python program to extract both shape and texture features, image name, and rice variety from the processed images in all 7 folders. Then, the extracted features were stored in an excel file (.xlsx file). The shape and texture features of interest are listed in Table 3.2.

After extracting the image features, we used a Python program to normalize the data and store the normalized dataset in the original .xlsx file format. This was done to reduce the complexity and organize the dataset. The details of the Python code used for data normalization can be found in Appendix B.7.

Table 3.2 Table of Shape Features and Texture Features.

Shape features	Texture features
Area	Correlation
Perimeter	Dissimilarity
Extent	Energy
Convex Area	Entropy
Aspect Ratio	Contrast
Kurtosis	Homogeneity
Skewness	Uniformly
Major Axis	Mean
Minor Axis	Variance
Standard Deviation	Skewness
Peak Value	Kurtosis
Max Gray Value	
Min Gray Value	
Edginess	
Normalized center of mass	
Eccentricity	
Solidity	
Compactness	
Shape Factor	
Equivalent Diameter	
Entropy	

3.5 Machine Learning Modeling

To create a classification model We use the normalized feature data of image processing according to the methods described in Section 3.2 to build models, 5 models per dataset, for a total of 35 models, with every model performing a 10-fold cross validation and evaluate the performance of the model. The classification model steps are shown in Figure 3.3.

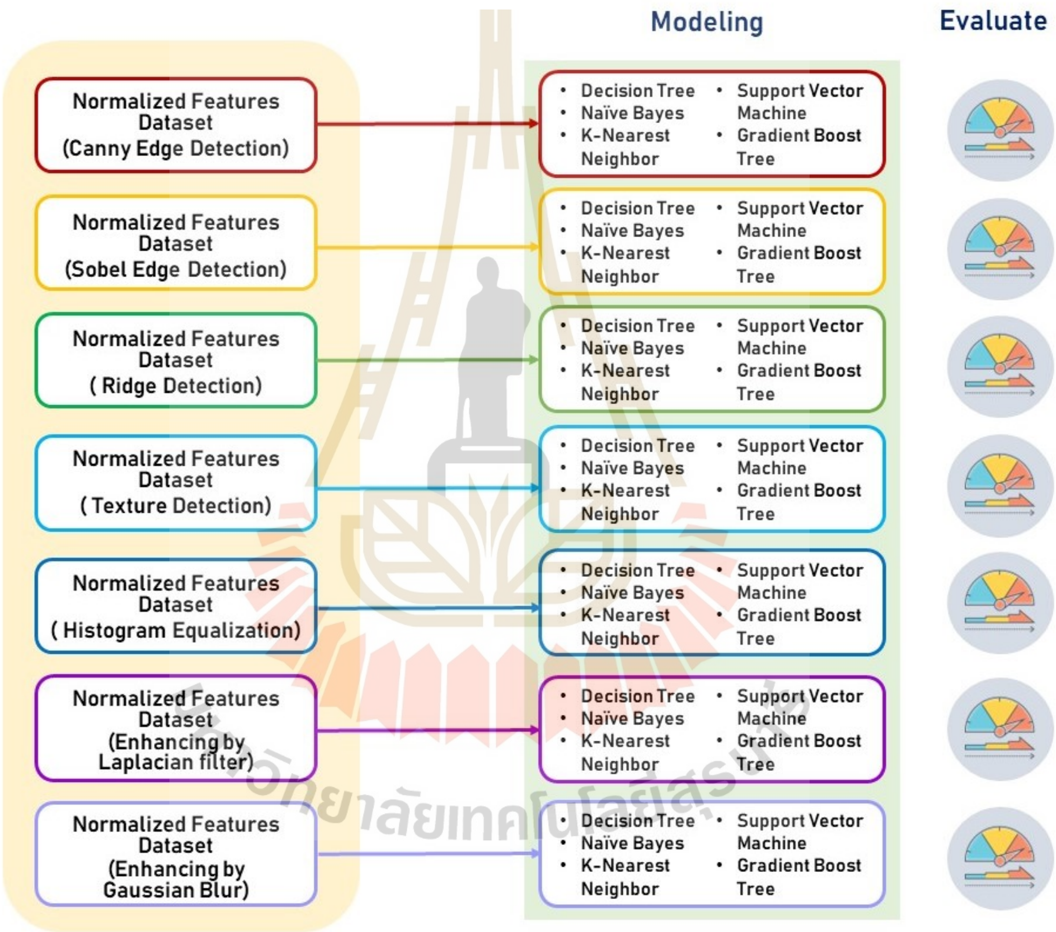


Figure 3.3 Procedure for Classification Model Creation.

3.6 Evaluate the Performance of the Model

To ensure accurate predictions and identify areas for improvement, we evaluated the performance of the model. The indicators used for evaluation in this study are shown in Table 3.3. A Python program was used to calculate the values of each metric.

Table 3.3 Indicators for Evaluating Performance.

	Indicators
1	Accuracy
2	Precision
3	Recall
4	F1-Score
5	Cohen's Kappa

In addition, the time taken for the model to classify each image was recorded.

CHAPTER IV

RESULTS AND DISCUSSION

This chapter presents the results of reduce noise in background images, image processing of rice grain images using 7 image processing methods and the results of evaluating the performance of the machine learning model using the data obtained by extracting features from the images processed by each method.

4.1 Noise Reduction of Image Background

Due to the noise in the rice grain images, as shown in Figure 4.1 (a), a Python program was used to preprocess the images before cropping. First, the background was filtered to black using a thresholding technique. Then, the area around the rice grains was cropped to reduce the noise in the images. The result of the preprocessing step is shown in Figure 4.1 (b).

Then, a Python program was used to place the cropped images on a black image of 250x250 pixels in the center of the image to match the original image. The result is shown in Figure 4.1 (c).

(a) Noise Occuring in Image. (b) The cropped rice grain. (c) Denoised image.

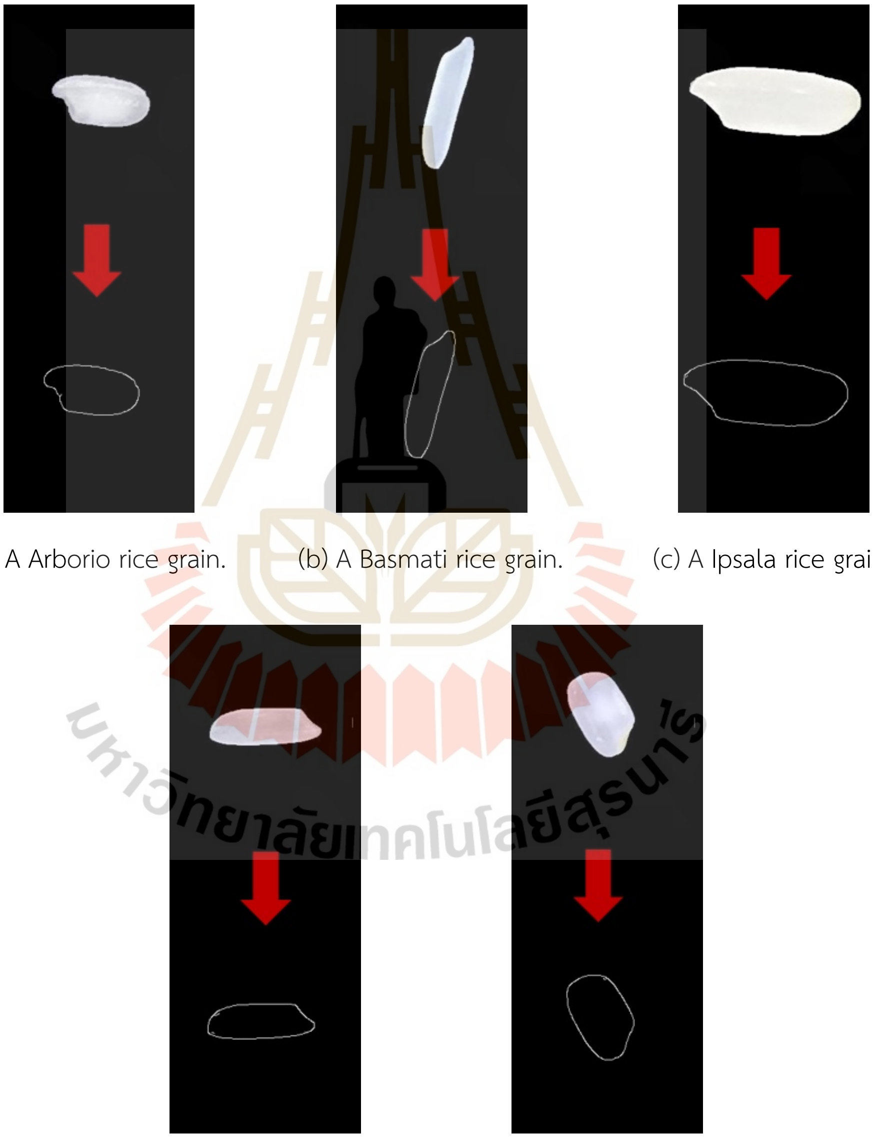
Figure 4.1 Reducing noise in the background of rice grain image.

The details of the Python program for noise reduction and placing the rice grain images on the black image can be found in Appendices B.1 and B.2.

4.2 Results from Image Processing of Rice Grains

Using a Python program on Jupyter Notebook, we processed rice grain images to extract features using the following 7 image processing methods: Canny Edge Detection, Sobel Edge Detection, Ridge Detection, Texture Detection, Histogram Equalization, Laplacian Filter Enhancement, and Gaussian Blur Enhancement.

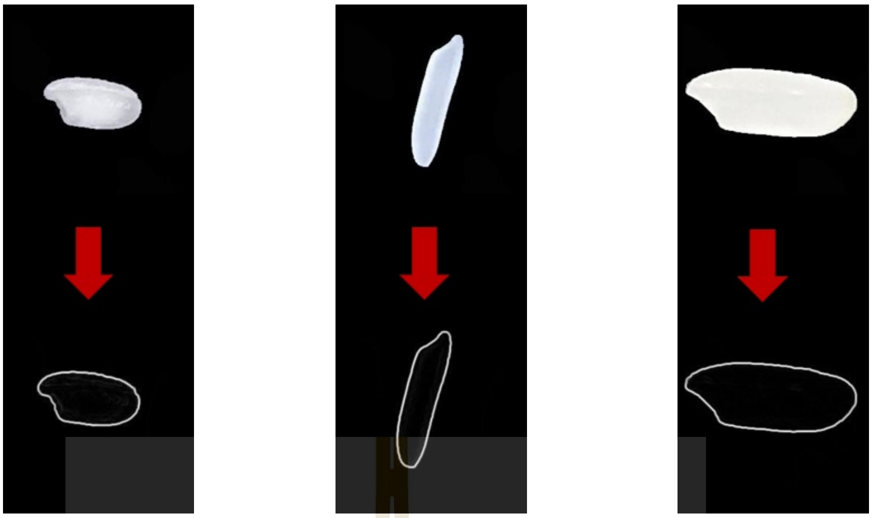
The result images of each image processing method are shown in Figures 4.2 - 4.8.



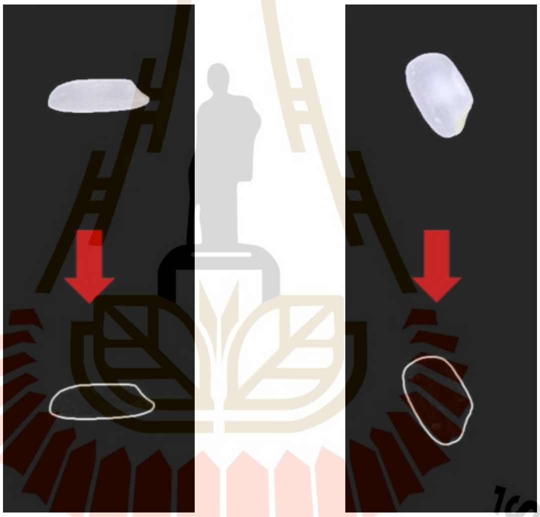
(a) A Arborio rice grain. (b) A Basmati rice grain. (c) A Ipsala rice grain.

(d) A Jasmine rice grain. (e) A Karacadag rice grain.

Figure 4.2 Rice grains processed with the Canny Edge Detection method.



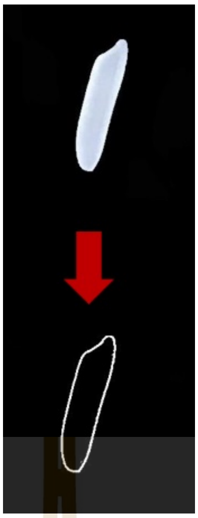
(a) A Arborio rice grain. (b) A Basmati rice grain. (c) A Ipsala rice grain.



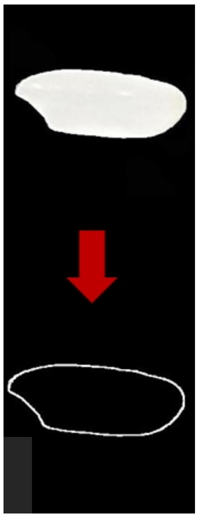
(d) A Jasmine rice grain. (e) A Karacadag rice grain.

Figure 4.3 Rice grains processed with the Sobel Edge Detection method.

(a) A Arborio rice grain.



(b) A Basmati rice grain.

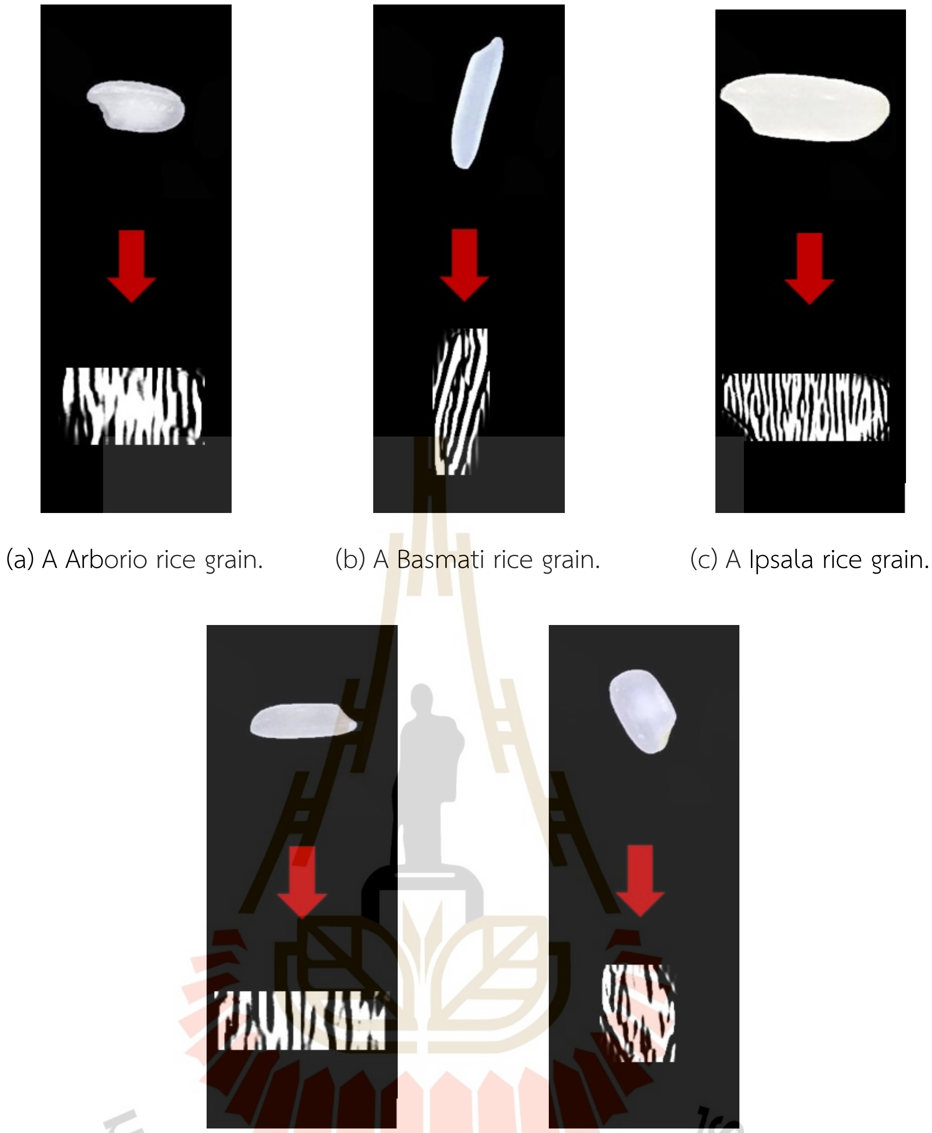


(c) A Ipsala rice grain.

(d) A Jasmine rice grain.

(e) A Karacadag rice grain.

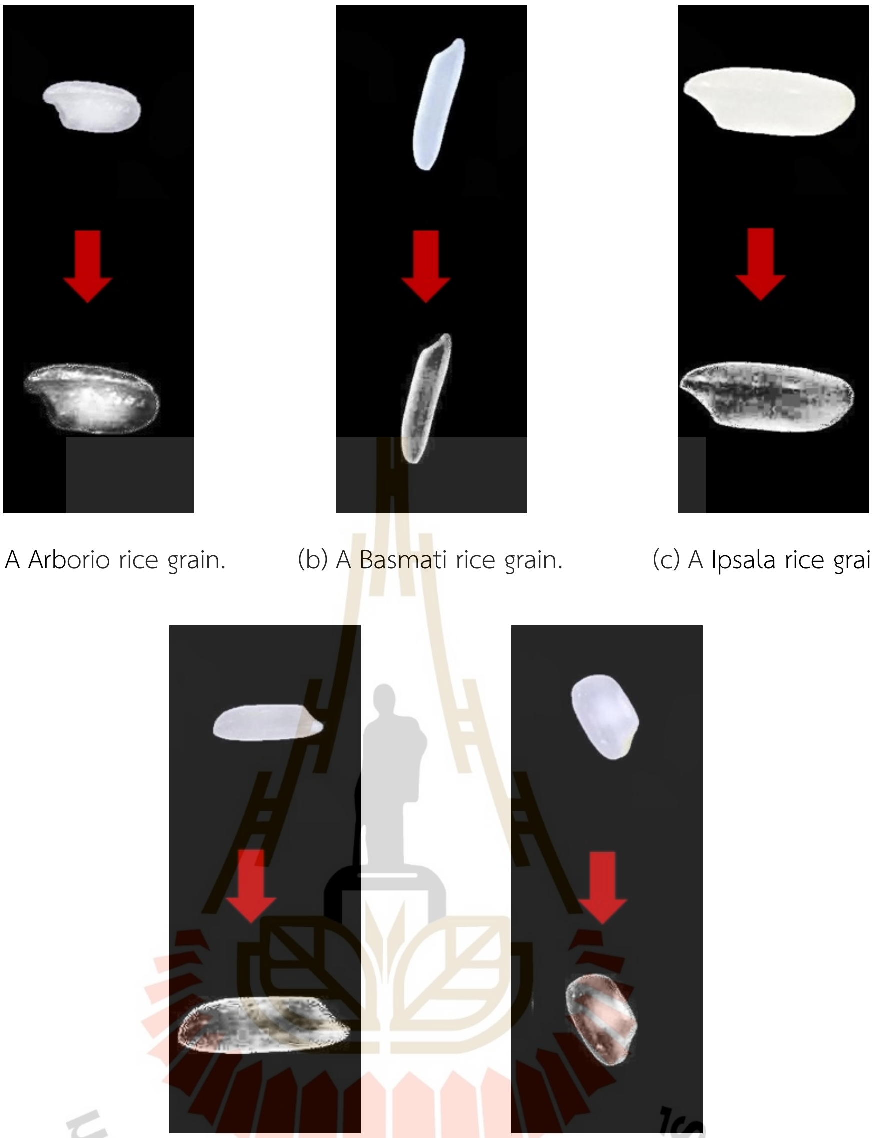
Figure 4.4 Rice grains processed with the Ridge Detection method.



(a) A Arborio rice grain. (b) A Basmati rice grain. (c) A Ipsala rice grain.

(d) A Jasmine rice grain. (e) A Karacadag rice grain.

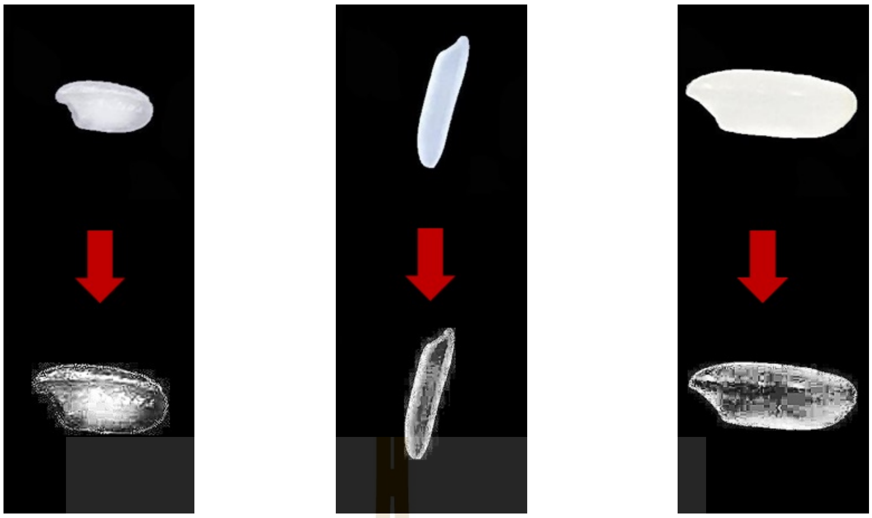
Figure 4.5 Rice grains processed with the Texture Detection method.



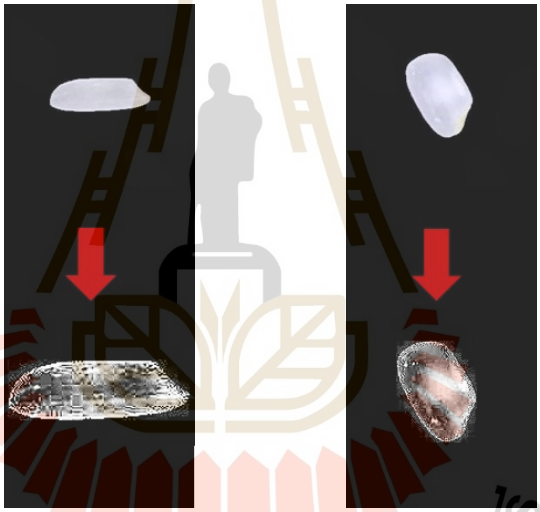
(a) A Arborio rice grain. (b) A Basmati rice grain. (c) A Ipsala rice grain.

(d) A Jasmine rice grain. (e) A Karacadag rice grain.

Figure 4.6 Rice grains processed with the Histogram Equalization method.



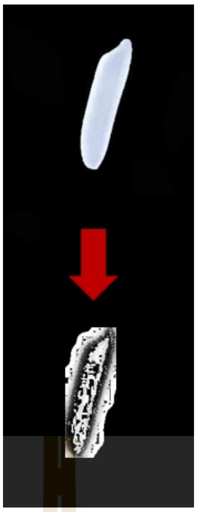
(a) A Arborio rice grain. (b) A Basmati rice grain. (c) A Ipsala rice grain.



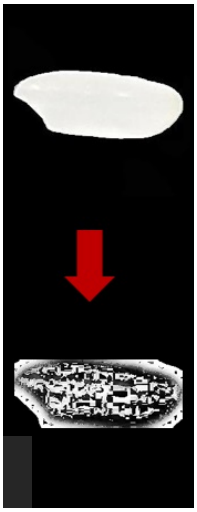
(d) A Jasmine rice grain. (e) A Karacadag rice grain.

Figure 4.7 Rice grains processed with the Enhancement by Laplacian filter method.

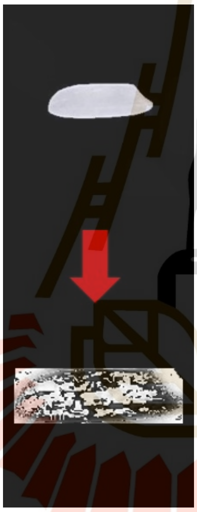
(a) A Arborio rice grain.



(b) A Basmati rice grain.



(c) A Ipsala rice grain.



(d) A Jasmine rice grain.

(e) A Karacadag rice grain.

Figure 4.8 Rice grains processed with Enhancement by the Gaussian Blur method.

4.3 Performance Evaluation of Data from Image Processing Combined with Various Machine Learning Techniques

From the processed and normalized image feature datasets, 7 datasets were used to create 5 classification models each, for a total of 35 models. The performance of each model was evaluated using the accuracy, precision, recall, and F1-score metrics.

The time taken for each model to classify was also recorded. The results are shown in Tables 4.1-4.7.

Table 4.1 The Performance of Canny Edge Detection Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	95.15%	95.15%	95.15%	95.14%	93.93%	168.97
NB	88.13%	88.04%	88.13%	87.94%	85.16%	**13.68**
K-NN	96.92%	96.93%	96.92%	96.92%	96.15%	32.05
GBT	97.15%	97.15%	97.15%	97.15%	96.44%	5886.94
SVM	**97.61%**	**97.61%**	**97.61%**	**97.61%**	**97.02%**	413.03

Table 4.2 The Performance of Sobel Edge Detection Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	95.55%	95.55%	95.5%	95.54%	94.44%	60.29
NB	84.76%	84.85%	84.76%	84.40%	80.95%	**4.91**
K-NN	96.30%	96.32%	96.30%	96.29%	95.37%	13.15
GBT	97.76%	97.76%	97.75%	97.75%	97.20%	9168.98
SVM	**98.68%**	**98.67%**	**98.67%**	**98.67%**	**98.35%**	136.21

Table 4.3 The Performance of Ridge Detection Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	94.79%	94.78%	94.78%	94.78%	93.48%	35.72
NB	88.81%	88.73%	88.81%	88.69%	86.01%	**5.24**
K-NN	95.52%	95.53%	95.52%	95.52%	94.40%	11.21
GBT	**96.81%**	**96.81%**	**96.81%**	**96.81%**	**96.01%**	2639.67
SVM	96.45%	96.45%	96.44%	96.44%	95.56%	207.60

Table 4.4 The Performance of Texture Detection Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	92.94%	92.93%	92.93%	92.93%	91.17%	42.02
NB	85.53%	85.50%	85.53%	85.41%	81.92%	**4.16**
K-NN	94.39%	94.45%	94.39%	94.37%	92.98%	9.64
GBT	**96.24%**	**96.24%**	**96.24%**	**96.23%**	**95.30%**	4845.29
SVM	95.42%	95.42%	95.42%	95.41%	94.28%	276.62

Table 4.5 The Performance of Histogram Equalization Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	93.47%	93.46%	93.46%	93.45%	91.83%	60.40
NB	87.04%	87.10%	87.04%	86.98%	83.80%	**4.25**
K-NN	95.47%	94.65%	94.57%	94.57%	93.22%	9.77
GBT	**96.79%**	**96.79%**	**96.79%**	**96.79%**	**95.99%**	4869.08
SVM	96.12%	96.13%	96.12%	96.12%	95.16%	258.16

Table 4.6 The Performance of Enhancement by Laplacian filter Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	93.74%	93.74%	93.73%	93.73%	92.18%	60.86
NB	87.39%	87.44%	87.39%	87.32%	84.23%	**4.21**
K-NN	95.25%	95.32%	95.25%	95.24%	94.06%	9.25
GBT	**96.88%**	**96.88%**	**96.87%**	**96.87%**	**96.10%**	5177.58
SVM	95.65%	95.66%	95.70%	95.65%	94.56%	256.39

Table 4.7 The Performance of Enhancement by Gaussian Blur Dataset with Various Machine Learning Techniques.

Machine learning Model	Accuracy	Precision	Recall	F1-Score	Kappa	Time (second)
DT	93.53%	93.51%	93.52%	93.52%	91.92%	49.53
NB	85.63%	85.94%	85.63%	85.63%	82.03%	**3.99**
K-NN	82.54%	82.91%	82.54%	82.65%	78.17%	9.45
GBT	96.83%	96.82%	96.82%	96.82%	96.03%	4992.54
SVM	**97.03%**	**97.03%**	**97.03%**	**97.03%**	**96.29%**	213.79

Remark: Bold and underlined text indicates the highest values of accuracy, precision, recall, F1-score, and Cohen's kappa with the fastest classification time (second).

CHAPTER V

CONCLUSION

This research evaluated the effectiveness of various techniques for rice variety classification using 250x250 pixel rice grain images. Image processing and machine learning were employed with a substantial dataset encompassing 75,000 images, consisting of 15,000 images for each of five diverse rice varieties: Arborio, Basmati, Ipsala, Jasmine, and Karacadag. To extract valuable information from the images, 32 features, including both shape and texture characteristics, were extracted from each image.

The best performing classification model utilized Sobel edge detection for image processing and the Support Vector Machine (SVM) technique for classification. It achieved an accuracy of 98.68%, precision of 98.67%, recall of 98.67%, F1 score of 98.67%, and Cohen's kappa of 98.35%.

Compared to previous research, the proposed method outperformed many studies or achieved comparable performance. Notably, Zareiforoush et al. (2016) obtained an accuracy of 98.72% for classifying four rice varieties, while Cinar and Köklü (2022) achieved an accuracy of 99.91% for classifying five rice varieties using a higher number of features (106 features compared to 32 features in this study). This suggests that increasing the number of features or adjusting the parameters in this research could potentially improve the performance.

Although Sobel edge detection with SVM achieved high accuracy, it is important to consider the processing time of the model. Sobel edge detection with SVM takes longer than other methods (136.21 seconds). Another high-performing method is Sobel edge detection with Gradient Boosting Trees, which achieved an accuracy of 97.76%. However, it has the longest processing time among all the models (9168.98 seconds). In contrast, Gaussian blur image enhancement with Naive Bayes had the shortest processing time (3.99 seconds), but its performance was moderate. Ultimately, the choice of the most suitable approach hinges on the specific application's priorities.

Overall, this research demonstrates the effectiveness of image processing and machine learning techniques for rice variety classification, paving the way for further advancements in rice grain analysis and prediction, and contributing to improved efficiency and quality control in the rice industry.

REFERENCES

REFERENCES

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine learning, 6*(1), 37-66.

Aki, O., Güllü, A., and Uçar, E. (2015). *Classification of Rice Grains Using Image Processing and Machine Learning Techniques*. Paper presented at International Scientific Conference “UNITECH 2015”, 20 – 21 November 2015, Gabrovo: 352-354.

Aznar, P. (2020, 02/12/2020). *Decision Trees : Gini vs Entropy*. Retrieved from https://quantdare.com/decision-trees-gini-vs-entropy/

Brownlee, J. (2023). *A Gentle Introduction to k-fold Cross-Validation*. Retrieved from https://machinelearningmastery.com/k-fold-cross-validation/

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees.* Chapman & Hall.

Chicco, D., and Jurman, G. (2020). The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics, 21*(1), 1-13.

Cinar, I., and Koklu, M. (2019). Classification of Rice Varieties Using Artificial Intelligence Methods. *International Journal of Intelligent Systems and Applications in Engineering (IJISAE) 7*(3), 188–194.

Cinar, I., and Koklu, M. (2021). Determination of Effective and Specific Physical Features of Rice Varieties by Computer Vision in Exterior Quality Inspection. *Selcuk Journal of Agriculture and Food Sciences (SJAFS) 35*(3), 229-243.

Cinar, I., and Koklu, M. (2022). Identification of Rice Varieties Using Machine Learning Algorithms. *Journal of Agricultural Sciences (Tarim Bilimleri Dergisi) 28*(2), 307-325.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement, 20*(1), 37-46.

Cortes, C., and Vapnik, V. (1995). Support-vector networks. *Machine learning, 20*(3), 273-297.

Deng, G. and Cahill, L.W. (1993). An adaptive Gaussian filter for noise reduction and edge detection. *Proceeding of Nuclear Science Symposium and Medical Imaging Conference, 3,* 1615 - 1619. doi : 10.1109/NSSMIC.1993.373563.

Deng, N., Tian, Y., and Zhang, C. (2012). *Support vector machines: optimization based theory, algorithms, and extensions*: CRC press.

Dimitoglou, G., Adams, J., and Jim, C. (2012). Comparison of the C4.5 and a Naive Bayes Classifier for the Prediction of Lung Cancer Survivability, *arXiv, 4*, Retrieved from https://arxiv.org/abs/1206.1121.

Fernández, A., García, S., Galar, M., Prati, R. C., Krawczyk, B., and Herrera, F. (2018). *Learning from imbalanced data sets* (Vol. 10): Springer Science+Business Media.

Farid, D., Rahman, M., and Al-Mamun, M. (2014). *Efficient and scalable multi-class classification using naïve Bayes tree*. Paper presented at 2014 International Conference on Informatics, Electronics and Vision (ICIEV 2014), Dhaka, Bangladesh

García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information sciences, 180*(10), 2044-2064.

Ghosh, A., Sufian, A., Sultana, F.,Chakrabarti, A., and De, D. (2020). Fundamental Concepts of Convolutional Neural Network. *Recent Trends and Advances in Artificial Intelligence and Internet of Things*, 519-567

Grandini, M., Bagli, E., and Visani, G. (2020). Metrics for Multi-Class Classification: an Overview. *arXiv, 8*, Retrived from https://arxiv.org/ftp/arxiv/papers/2209/2209.08699.

He, H., and Ma, Y. (2013). *Imbalanced learning: foundations, algorithms, and applications.* Wiley-IEEE Press.

Hearst, M., Dumais, S.T., Osman, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. *Intelligent Systems and their Applications, 13,* 18 - 28. doi : 10.1109/5254.708428.

Kass, M., Witkin, A., and Terzopolous, D. (1988). Snakes: Active Contour Models. *International Journal of Computer Vision 1*(4), 321-331

Koklu, M., Cinar, I., and Taspinar, Y. S. (2021). Classification of Rice Varieties with Deep Learning Methods. *Computers and Electronics in Agriculture 187*(2021), 1-8.

López, V., Triguero, I., Carmona, C. J., García, S., and Herrera, F. (2014). Addressing imbalanced classification with instance generation techniques: IPADE-ID. *Neurocomputing, 126*, 15-28.

Madzarov, G., Gjorgjevikj, D., and Chorbev, I. (2009). Multi-class Classification using Support Vector Machines in Binary Tree Architecture. *IEEE EUROCON 2009*(pp. 288-295). Russia, doi: 10.1109/EURCON.2009.5167645.

McHugh, M. (2012). *Interrater reliability: the kappa statistic*. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/.

Saritas, M., and Yasar, A. (2019). Performance Analysis of ANN and Naïve Bayes Classification Algorithm for Data Classification. *International Journal of Intelligent Systems and Applications in Engineering 7*(2), 88–91.

Natekin, A., and Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7, 21. doi : 10.3389/fnbot.2013.00021

Neapolitan, R. E., and Jiang, X. (2018). *Artificial Intelligence: With an Introduction to Machine Learning.* (2 ed.) CRC Press.

Nikhil, G. S. (2023). *Histogram Equalization — Everything you need to know.* Retrived from https://nikhilgandhudi.medium.com/histogram-equalization-everything-you-need-to-know-dd5e41a47da8.

NV5 Geospatial Software, (2023). *Apply Laplacian Filters*, Retrived from https://www.nv5geospatialsoftware.com/docs/LaplacianFilters.html.

Quinlan, J. (1986). Indroduction of decision trees machine learning. *Boston (NL): Kluwer Acad. Publ, 1*(86-106), 650.

Quinlan, J. R. (1993). *C4. 5: programs for machine learning*: Morgan Kaufmann Publishers Inc.

Reddy, R., Nagaraju, C., and Reddy, I. (2016). Canny Scale Edge Detection. *International Journal of Engineering Trends and Technology*. doi : 10.14445/22315381/IJETT-ICGTETM-N3/ICGTETM-P121.

Rexce, J., and Usha Kingsly Devi, K. (2017). Classification of Milled Rice Using Image Processing. *International Journal of Scientific & Engineering Research*.

Rokach, L., and Maimon, O. (2005). *Data mining and knowledge discovery handbook*. Springer Science+Business Media.

Sá, A., Almeida, A., Rocha, B., Mota, M., Souza, J., and Dentel, L. (2011). *Lightning forecast using data mining techniques on hourly evolution of the convective available potential energy*. Paper presented at the Brazilian Congress on Computational Intelligence, Fortaleza, November.

scikit-learn. (2022). *Precision-Recall.* Retrieved from https://scikit-learn.org/stable/auto-examples/model_selection/plot_precision_recall.html

Shah, A., (2018), *Through The Eyes of Gabor Filter*. Retrieved from https://medium.com/anuj_shah/@through-the-eyes-of-gabor-filter-17d1fdb3ac97.

Shokouh, G. S., Baptiste, M., Xu, B., and Montesinos, P. (2021). Ridge Detection by Image Filtering Techniques: A Review and an Objective Analysis. *Pattern Recognition and Image Analysis. 31*, 551-570. doi : 10.1134/S1054661821030226.

Singh, S., and Gupta, P. (2014). Comparative study ID3, cart and C4. 5 decision tree algorithm: a survey. *International Journal of Advanced Information Science and Technology (IJAIST), 27*(27), 97-103.

Taunk, K., De, S., Verma, S., and Swetapadma, A. (2019). *A brief review of nearest neighbor algorithm for learning and classification*. Paper presented at the 2019 International Conference on Intelligent Computing and Control Systems (ICCS).

Thai Rice Exporters Association (2024), *F.O.B. Prices*. Retrieved from http://www.thairiceexporters.or.th/

Wakefield, K. (2022). *A guide to the types of machine learning algorithms and their applications.* Retrieved from https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html

Wang, Z., Xu, H., Zhou, P. and Xiao, G. (2023). An Improved Multilabel k-Nearest Neighbor Algorithm Based on Value and Weight. *Computation 2023, 11*, 32. doi : 10.3390/computation11020032

Sobel operator. (2024). Retrieved February 5, 2024 from Wikipedia: https://en.wikipedia.org/wiki/Sobel_operator

Zareiforoush, H., Minaei, S., Alizadeh, M. R., and Banaka, A. (2016). Qualitative Classification of Milled Rice Grains Using Computer Vision and Metaheuristic Techniques. *Journal of Food Science Technology, 53*(1). 118-131.

APPENDICES

APPENDIX A

THE RESULTS TABLE OF THE PERFORMANCE OF MACHINE LEARNING MODELS

A.1 The Results of the Performance of Canny edge detection

The classification results using the feature dataset from Canny edge detection and various machine learning techniques are shown in tables A.1-A.5.

Table A.1 Decision Tree with Canny edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13908	35	15	149	898	92.40%	92.69%	92.54
Basmati	35	14533	0	431	1	95.09%	96.89%	95.98
Ipsala	27	0	14883	88	2	98.90%	99.22%	99.06
Jasmine	196	715	150	13930	9	95.40%	92.87%	94.06
Karacadake	885	0	0	4	14111	93.94%	94.07%	94.01
Accuracy	95.15%							
Kappa	93.93%							
Time	168.96s							

Table A.2 Naïve Bayes with Canny edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	12933	85	0	295	1687	86.69%	86.22%	86.45%
Basmati	317	13348	26	1308	1	86.64%	88.99%	87.80%
Ipsala	59	9	14874	58	0	92.85%	99.16%	95.90%
Jasmine	498	1965	1120	11079	338	86.80%	73.86%	79.81%
Karacadake	1112	0	0	24	13864	87.25%	92.43%	89.76%
Accuracy	88.13%							
Kappa	85.16%							
Time	13.68s							

Table A.3 K-NN with Canny edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14061	1	0	104	834	96.12%	93.74%	94.92%
Basmati	12	14742	0	246	0	97.08%	98.28%	97.67%
Ipsala	15	1	14922	62	0	99.73%	99.48%	99.60%
Jasmine	63	442	41	14449	5	97.19%	96.33%	96.76%
Karacadake	477	0	0	6	14517	94.54%	96.78%	95.65%
Accuracy	96.92%							
Kappa	96.15%							
Time	32.6705s							

Table A.4 Gradient Boost Tree with Canny edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14219	9	7	137	628	95.73%	94.79%	95.26%
Basmati	43	14697	1	259	0	97.67%	97.98%	97.82%
Ipsala	11	0	14924	65	0	99.67%	99.49%	99.58%
Jasmine	93	142	41	14522	21	96.87%	96.81%	96.84%
Karacadake	488	0	0	8	14504	96.69%	96.69%	96.26%
Accuracy	97.15%							
Kappa	96.44%							
Time	5886.94s							

Table A.5 Support Vector Machine with Canny edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14357	1	0	69	573	96.64%	95.71%	96.17%
Basmati	5	14709	0	286	0	98.03%	98.06%	98.05%
Ipsala	6	0	14956	38	0	99.83%	99.71%	99.77%
Jasmine	54	294	25	14624	3	97.38%	97.49%	97.43%
Karacadake	434	0	0	0	14565	96.20%	97.10%	96.65%
Accuracy	97.61%							
Kappa	97.02%							
Time	413.03s							

A.2 The Results of the Performance of Sobel edge detection

The classification results using the feature dataset from Sobel edge detection and various machine learning techniques are shown in tables A.6-A.10.

Table A.6 Decision Tree with Sobel edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13920	10	6	181	883	92.65%	92.82%	92.72%
Basmati	16	14630	1	350	3	96.61%	97.53%	97.07%
Ipsala	6	1	14900	92	1	99.02%	99.33%	99.17%
Jasmine	221	501	141	14107	30	95.58%	94.05%	94.81%
Karacadake	862	2	0	29	14107	93.90%	94.05%	93.97%
Accuracy	95.55%							
Kappa	94.44%							
Time	60.29s							

Table A.7 Naïve Bayes with Sobel edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	12429	543	0	445	1583	86.43%	82.86%	84.61%
Basmati	781	13255	17	931	16	83.32%	88.37%	85.77%
Ipsala	1	26	14755	218	0	94.12%	98.37%	96.20%
Jasmine	220	2085	905	9459	2331	82.73%	63.06%	71.57%
Karacadake	949	0	0	380	13671	77.67%	91.14%	83.87%
Accuracy	84.76%							
Kappa	80.95%							
Time	4.91s							

Table A.8 K-NN with Sobel edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13822	0	1	87	1090	95.75%	92.15%	93.91%
Basmati	23	14730	1	245	1	96.75%	98.20%	97.47%
Ipsala	12	5	14876	107	0	99.25%	99.17%	99.21%
Jasmine	130	490	111	14245	24	97.00%	94.97%	95.97%
Karacadake	449	0	0	2	14549	92.88%	96.99%	94.89%
Accuracy	96.30%							
Kappa	95.37%							
Time	13.15s							

Table A.9 Gradient Boost Tree with Sobel edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14336	2	0	113	549	97.30%	95.57%	96.43%
Basmati	26	14770	0	204	0	98.02%	98.47%	98.42%
Ipsala	4	0	14928	68	0	99.75%	99.52%	99.63%
Jasmine	78	25	38	14579	10	97.39%	97.19%	97.29%
Karacadake	288	1	0	5	14706	96.34%	98.04%	97.18%
Accuracy	97.76%							
Kappa	97.20%							
Time	9168.98s							

Table A.10 Support Vector Machine with Sobel edge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14703	1	2	40	254	98.62%	98.02%	98.32%
Basmati	0	14758	0	242	0	98.66%	98.39%	98.52%
Ipsala	3	0	14966	31	0	99.87%	99.77%	99.82%
Jasmine	39	200	17	14743	1	97.92%	98.29%	98.10%
Karacadake	163	0	0	0	14837	98.31%	98.91%	98.61%
Accuracy	98.68%							
Kappa	98.35%							
Time	136.21s							

A.3 The Results of the Performance of Ridge detection

The classification results using the feature dataset from Ridge detection and various machine learning techniques are shown in tables A.11-A.15.

Table A.11 Decision Tree with Ridge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13822	36	44	200	898	91.60%	92.15%	91.87%
Basmati	40	14442	2	516	0	95.45%	96.28%	95.86%
Ipsala	45	1	14855	99	0	98.61%	99.03%	98.82%
Jasmine	258	653	163	13908	19	94.39%	92.72%	93.55%
Karacadake	925	0	0	12	14063	93.88%	93.75%	93.82%
Accuracy	94.79%							
Kappa	93.48%							
Time	35.72s							

Table A.12 Naïve Bayes with Ridge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	12404	9	1	948	1638	88.22%	82.69%	85.37%
Basmati	781	13255	17	931	16	87.22%	93.15%	90.09%
Ipsala	1	26	14755	218	0	95.50%	98.28%	96.87%
Jasmine	220	2085	905	9459	2331	84.63%	77.93%	81.14%
Karacadake	949	0	0	380	13671	88.06%	91.99%	89.98%
Accuracy	88.81%							
Kappa	88.81%							
Time	5.24s							

Table A.13 K-NN with Ridge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13628	4	9	167	1192	93.36%	90.85%	92.09%
Basmati	19	14658	0	323	0	96.42%	97.72%	97.06%
Ipsala	65	1	14850	84	0	99.59%	99.00%	99.29%
Jasmine	91	540	52	14311	6	96.08%	95.41%	95.74%
Karacadake	795	0	0	10	14195	92.22%	94.63%	93.41%
Accuracy	95.52%							
Kappa	94.40%							
Time	11.21s							

Table A.14 Gradient Boost Tree with Ridge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14144	22	15	159	660	95.22%	94.29%	94.75%
Basmati	44	14661	0	295	0	97.28%	97.74%	97.51%
Ipsala	35	0	14893	71	1	99.54%	99.29%	99.41%
Jasmine	129	388	54	14424	5	96.41%	96.16%	96.29%
Karacadake	502	0	0	12	14486	95.60%	96.57%	96.09%
Accuracy	96.81%							
Kappa	96.01%							
Time	2639.67s							

Table A.15 Support Vector Machine with Ridge detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14081	4	13	245	657	95.07%	93.87%	94.47%
Basmati	5	14493	0	502	0	97.26%	96.62%	96.94%
Ipsala	52	1	14883	64	0	99.65%	99.22%	91.87%
Jasmine	203	403	40	148352	2	94.63%	95.68%	95.15%
Karacadake	470	0	0	4	14526	95.66%	96.84%	96.25%
Accuracy	96.45%							
Kappa	95.56%							
Time	207.60s							

A.4 The Results of the Performance of Texture detection

The classification results using the feature dataset from Texture detection and various machine learning techniques are shown in tables A.16-A.20.

Table A.16 Decision Tree with Texture detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13411	95	12	232	1205	89.13%	89.41%	89.27%
Basmati	79	14270	6	638	7	93.41%	95.13%	94.27%
Ipsala	12	2	14813	171	3	98.26%	98.82%	98.54%
Jasmine	315	901	243	13493	48	92.56%	89.95%	91.24%
Karacadake	1230	8	1	54	13707	91.29%	91.38%	91.33%
Accuracy	92.94%							
Kappa	91.17%							
Time	42.02s							

Table A.17 Naïve Bayes with Texture detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	11806	561	0	559	2074	82.43%	78.71%	80.53%
Basmati	995	12943	2	953	107	82.60%	86.29%	84.40%
Ipsala	23	19	14772	186	0	95.19%	98.48%	96.81%
Jasmine	212	2118	745	11267	658	84.78%	75.11%	79.65%
Karacadake	1268	28	0	325	13361	82.48%	91.14%	85.65%
Accuracy	85.53%							
Kappa	81.92%							
Time	4.19s							

Table A.18 K-NN with Texture detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13067	12	0	163	1758	93.18%	87.11%	90.04%
Basmati	48	14546	0	400	6	96.04%	96.97%	96.50%
Ipsala	20	5	14888	87	0	99.03%	99.25%	99.14%
Jasmine	191	583	146	14020	60	95.35%	93.47%	94.40%
Karacadake	698	0	0	99	14269	88.67%	95.13%	91.78%
Accuracy	94.39%							
Kappa	92.98%							
Time	9.64s							

Table A.19 Gradient Boosted Tree with Texture detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13909	13	0	176	982	94.97%	92.73%	93.38%
Basmati	42	14616	0	340	2	97.88%	97.44%	97.26%
Ipsala	5	0	14905	90	0	99.45%	99.37%	99.41%
Jasmine	129	425	82	14348	16	95.72%	95.65%	95.69%
Karacadake	561	2	0	36	14401	94.00%	96.01%	94.99%
Accuracy	96.24%							
Kappa	95.30%							
Time	4845.29s							

Table A.20 Support Vector Machine with Texture detection.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13746	23	0	174	1057	93.68%	91.64%	92.65%
Basmati	35	14579	0	383	3	95.94%	97.19%	96.56%
Ipsala	1	4	14917	78	0	99.45%	99.45%	99.45%
Jasmine	134	588	83	14137	58	95.37%	94.25%	94.80%
Karacadake	758	2	0	52	14188	92.70%	94.59%	93.63%
Accuracy	95.42%							
Kappa	94.28%							
Time	276.62s							

A.5 The Results of the Performance of Histogram equalization

The classification results using the feature dataset from Histogram Equalization and various machine learning techniques are shown in tables A.21-A.25.

Table A.21 Decision Tree with Histogram Equalization.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13483	132	5	317	1063	89.24%	89.89%	89.56%
Basmati	116	14417	7	409	51	95.37%	96.11%	95.74%
Ipsala	15	3	14795	184	3	97.51%	98.63%	98.07%
Jasmine	417	520	364	13617	82	93.18%	90.78%	91.96%
Karacadake	1078	45	2	87	13788	92.00%	91.20%	91.96%
Accuracy	93.47%							
Kappa	91.83%							
Time	60.40s							

Table A.22 Naïve Bayes with Histogram Equalization.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	12325	255	0	572	1848	80.56%	82.17%	82.17%
Basmati	1192	12304	6	1084	414	90.09%	82.03%	85.87%
Ipsala	23	13	14790	174	0	95.09%	98.60%	96.81%
Jasmine	681	1086	758	12141	334	85.68%	80.94%	83.24%
Karacadake	1078	0	0	199	13723	84.09%	91.49%	87.63%
Accuracy	87.04%							
Kappa	83.80%							
Time	4.25s							

Table A.23 K-NN with Histogram Equalization.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13132	41	2	248	1577	92.27%	87.55%	89.85%
Basmati	228	14477	0	171	124	96.84%	96.51%	96.68%
Ipsala	16	11	14894	79	0	98.75%	99.29%	99.02%
Jasmine	332	410	187	13983	88	96.42%	93.22%	94.79%
Karacadake	524	10	0	21	14445	88.98%	96.30%	92.50%
Accuracy	94.57%							
Kappa	93.22%							
Time	9.77s							

Table A.24 Gradient Boosted Tree with Histogram Equalization.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13993	15	0	206	786	95.15%	93.29%	94.21%
Basmati	77	14716	1	179	27	98.20%	98.11%	98.16%
Ipsala	4	2	14897	97	0	99.45%	99.31%	99.38%
Jasmine	157	247	81	14490	25	96.64%	96.60%	96.62%
Karacadake	475	5	0	22	14498	94.54%	96.65%	95.58%
Accuracy	96.79%							
Kappa	95.99%							
Time	4869.08s							

Table A.25 Support Vector Machine with Histogram Equalization.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13955	40	0	175	830	94.48%	93.03%	93.75%
Basmati	58	14646	2	2227	67	97.35%	97.64%	97.49%
Ipsala	0	6	14907	87	0	99.49%	99.38%	99.44%
Jasmine	155	314	74	14336	121	96.00%	95.57%	95.78%
Karacadake	683	39	0	109	14249	93.33%	94.99%	94.16%
Accuracy	96.12%							
Kappa	95.16%							
Time	258.16s							

A.6 The Results of the Performance of Enhancement by Laplacian filter

The classification results using the feature dataset from Enhancement by Laplacian filter and various machine learning techniques are shown in tables A.26-A.30.

Table A.26 Decision Tree with Enhancement by Laplacian filter.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13464	56	10	322	1148	88.96%	89.76%	89.36%
Basmati	66	14623	10	258	43	96.52%	97.49%	97.00%
Ipsala	11	7	14787	194	1	97.58%	98.58%	98.08%
Jasmine	436	407	346	13724	87	94.17%	91.49%	92.81%
Karacadake	1158	58	1	75	13708	91.47%	91.39%	91.43%
Accuracy	93.74%							
Kappa	92.18%							
Time	60.86s							

Table A.27 Naïve Bayes with Enhancement by Laplacian filter.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	12471	54	0	738	1737	81.05%	83.14%	82.08%
Basmati	846	12871	6	818	459	91.27%	85.81%	88.45%
Ipsala	39	19	14789	153	0	95.03%	98.59%	96.78%
Jasmine	994	1158	768	11674	406	85.78%	77.83%	81.61%
Karacadake	1037	0	0	237	13736	84.07%	91.57%	87.66%
Accuracy	87.39%							
Kappa	84.23%							
Time	4.21s							

Table A.28 K-NN with Enhancement by Laplacian filter.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13260	16	2	233	1489	93.07%	88.40%	90.68%
Basmati	183	14659	2	141	95	98.24%	97.73%	97.98%
Ipsala	31	10	14885	74	0	98.80%	99.23%	99.02%
Jasmine	370	233	177	14137	83	96.79%	94.25%	95.50%
Karacadake	483	3	0	21	14493	89.68%	96.62%	93.02%
Accuracy	95.25%							
Kappa	94.06%							
Time	9.25s							

Table A.29 Gradient Boosted Tree with Enhancement by Laplacian filter.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13920	15	0	208	857	94.98%	92.00%	93.88%
Basmati	26	14811	0	147	16	87.88%	98.74%	98.81%
Ipsala	4	2	14902	91	1	99.55%	99.35%	99.45%
Jasmine	175	142	67	14594	22	96.85%	97.29%	97.07%
Karacadake	530	9	0	29	14432	94.15%	96.21%	95.17%
Accuracy	96.88%							
Kappa	96.10%							
Time	5177.58s							

Table A.30 Support Vector Machine with Enhancement by Laplacian filter.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13604	18	0	176	1202	93.04%	90.69%	91.85%
Basmati	60	14663	4	223	50	98.36%	97.75%	98.06%
Ipsala	0	8	14930	62	0	99.55%	99.53%	99.54%
Jasmine	162	197	63	14458	120	96.26%	96.39%	96.32%
Karacadake	796	21	0	101	14082	91.12%	93.88%	92.48%
Accuracy	95.65%							
Kappa	94.56%							
Time	256.39s							

A.7 The Results of the Performance of Enhancement by Gaussian blur

The classification results using the feature dataset from Enhancement by Gaussian blur and various machine learning techniques are shown in tables A.31-A.35.

Table A.31 Decision Tree with Enhancement by Gaussian blur.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	13486	55	3	475	981	89.76%	89.91%	89.93%
Basmati	53	14566	12	260	109	96.98%	97.11%	97.05%
Ipsala	0	10	14788	201	1	97.69%	98.59%	98.14%
Jasmine	535	267	334	13609	255	92.11%	90.73%	91.42%
Karacadake	950	121	0	229	13700	91.05%	91.33%	91.19%
Accuracy	93.53%							
Kappa	91.92%							
Time	49.53s							

Table A.32 Naïve Bayes with Enhancement by Gaussian blur.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	12243	55	0	686	2016	78.97%	81.62%	80.27%
Basmati	711	11965	3	1825	496	93.99%	79.77%	86.30%
Ipsala	45	34	14766	155	0	95.54%	98.44%	96.97%
Jasmine	1182	661	687	11995	475	79.58%	79.97%	79.77%
Karacadake	1322	15	0	412	13251	81.60%	88.34%	84.84%
Accuracy	85.63%							
Kappa	82.03%							
Time	3.99s							

Table A.33 K-NN with Enhancement by Gaussian blur.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	11088	663	1	731	2517	68.77%	73.92%	71.25%
Basmati	2088	11901	1	679	331	87.37%	79.34%	83.16%
Ipsala	8	11	14717	264	0	97.98%	98.11%	98.04%
Jasmine	761	934	302	12088	915	84.17%	80.59%	82.34%
Karacadake	2179	112	0	600	12109	76.29%	80.73%	78.45%
Accuracy	82.54%							
Kappa	78.17%							
Time	9.45s							

Table A.34 Gradient Boosted Tree with Enhancement by Gaussian blur.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14068	19	1	245	667	95.16%	93.79%	94.47%
Basmati	25	14752	1	179	43	98.83%	98.35%	98.59%
Ipsala	1	5	14900	94	0	99.38%	99.33%	99.36%
Jasmine	268	110	91	14429	102	96.10%	96.19%	96.15%
Karacadake	421	41	0	68	14470	94.69%	96.47%	95.57%
Accuracy	96.83%							
Kappa	96.03%							
Time	4992.54s							

Table A.35 Support Vector Machine with Enhancement by Gaussian blur.

Varieties of rice grains	Arborio	Basmati	Ipsala	Jasmine	Karacadag	Precision	Recall	F1-Score
Arborio	14165	5	0	224	606	95.61%	94.43%	95.02%
Basmati	7	14790	3	176	24	98.96%	98.60%	98.79%
Ipsala	0	2	14934	64	0	99.51%	99.56%	99.53%
Jasmine	214	143	71	14425	147	96.19%	96.17%	96.18%
Karacadake	429	6	0	107	14458	94.90%	96.39%	95.64%
Accuracy	97.03%							
Kappa	96.29%							
Time	213.79s							

A.8 Performance Evaluation of Machine Learning Models Using Image Processing Datasets

Table A.36 The Performance of Image Processing Datasets with Decision Tree.

Dataset	Accuracy	Precision	Recall	F1-Score	Kappa	Time (sec)
Canny edge detection	95.15%	95.15%	95.15%	95.14%	93.93%	168.97
Sobel edge detection	**95.55%**	**95.55%**	**95.55%**	**95.55%**	**94.44%**	60.29
Ridge detection	94.79%	94.78%	94.78%	94.78%	93.48%	**35.72**
Texture detection	92.94%	92.93%	92.93%	92.93%	91.17%	42.02
Histogram equalization	93.47%	93.46%	93.46%	93.45%	91.83%	60.40
Enhancement by Laplacian filter	93.74%	93.74%	93.73%	93.73%	92.18%	60.86
Enhancement by Gaussian blur	93.53%	93.51%	93.52%	93.52%	91.92%	49.53

Table A.37 The Performance of Image Processing Datasets with Naïve Bayes.

Dataset	Accuracy	Precision	Recall	F1-Score	Kappa	Time (sec)
Canny edge detection	88.13%	88.04%	88.13%	87.94%	85.16%	13.68
Sobel edge detection	84.76%	84.75%	84.76%	84.40%	80.95%	4.91
Ridge detection	**88.81%**	**88.73%**	**88.81%**	**88.69%**	**86.01%**	5.24
Texture detection	85.53%	85.50%	85.53%	85.41%	81.92%	4.16
Histogram equalization	87.04%	87.10%	87.04%	86.98%	83.80%	4.25
Enhancement by Laplacian filter	87.39%	87.44%	87.39%	87.32%	84.23%	4.21
Enhancement by Gaussian blur	85.63%	85.94%	85.63%	85.63%	82.03%	**3.99**

Table A.38 The Performance of Image Processing Datasets with K-Nearest Neighbors.

Dataset	Accuracy	Precision	Recall	F1-Score	Kappa	Time (sec)
Canny edge detection	**96.92%**	**96.93%**	**96.92%**	**96.92%**	**96.15%**	32.05
Sobel edge detection	96.30%	96.32%	96.30%	96.29%	95.37%	13.15
Ridge detection	95.52%	95.53%	95.52%	95.52%	94.40%	11.21
Texture detection	94.39%	94.45%	94.39%	94.37%	92.98%	9.64
Histogram equalization	95.47%	94.65%	94.57%	94.57%	93.22%	9.77
Enhancement by Laplacian filter	95.25%	95.32%	95.25%	95.24%	94.06%	9.25
Enhancement by Gaussian blur	82.54%	82.91%	82.54%	82.65%	78.17%	**9.45**

Table A.39 The Performance of Image Processing Datasets with Gradient Boosted Tree.

Dataset	Accuracy	Precision	Recall	F1-Score	Kappa	Time (sec)
Canny edge detection	97.15%	97.15%	97.15%	97.15%	96.44%	5886.94
Sobel edge detection	**<u>97.76%</u>**	**<u>97.76%</u>**	**<u>97.75%</u>**	**<u>97.75%</u>**	**<u>97.20%</u>**	9168.98
Ridge detection	96.81%	96.81%	96.81%	96.81%	96.01%	**<u>2639.67</u>**
Texture detection	96.24%	96.24%	96.24%	96.23%	95.30%	4845.29
Histogram equalization	96.79%	96.79%	96.79%	96.79%	95.99%	4869.08
Enhancement by Laplacian filter	96.88%	96.88%	96.87%	96.87%	96.10%	5177.58
Enhancement by Gaussian blur	96.83%	96.82%	96.82%	96.82%	96.03%	4992.54

Table A.40 The Performance of Image Processing Datasets with Support Vector Machine.

Dataset	Accuracy	Precision	Recall	F1-Score	Kappa	Time (sec)
Canny edge detection	97.61%	97.61%	97.61%	97.61%	97.02%	413.03
Sobel edge detection	**<u>98.68%</u>**	**<u>98.67%</u>**	**<u>98.67%</u>**	**<u>98.67%</u>**	**<u>98.35%</u>**	**<u>136.21</u>**
Ridge detection	96.45%	96.45%	96.44%	96.44%	95.65%	207.60
Texture detection	95.42%	95.42%	95.42%	95.41%	9428%	276.62
Histogram equalization	96.12%	96.12%	96.12%	96.12%	95.16%	258.16
Enhancement by Laplacian filter	95.65%	95.66%	95.70%	95.65%	94.56%	256.39
Enhancement by Gaussian blur	97.03%	97.03%	97.03%	97.03%	96.29%	213.79

Remark: Bold and underlined text indicates the highest values of accuracy, precision, recall, F1-score, and Cohen's kappa with the fastest classification time (second).

APPENDIX B

APPLICATION OF PYTHON CODE IN IMAGE PROCESS, FEATURE EXTRACTION AND MACHINE LEARNING MODELING

This chapter presents some Python code using in this thesis.

B.1 Cropped Rice Grain Images By Python Code in Jupyter Notebook

```
import os
import cv2
import numpy as np

# Define the input folder and output folder paths
input_folder = "E:/Rice_Image_Dataset"
output_root = "C:/Users/Administrator/Desktop/Cropped_Objects_All"

# Function to crop the largest object in an image
def crop_largest_object(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, thresholded = cv2.threshold(gray, 128, 255, cv2.THRESH_BINARY)
    contours, _ = cv2.findContours(thresholded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) > 0:
        largest_contour = max(contours, key=cv2.contourArea)
        x, y, w, h = cv2.boundingRect(largest_contour)
        cropped_object = image[y:y+h, x:x+w]
        return cropped_object
    else:
        return None

# Recursive function to process subfolders
def process_subfolders(input_folder, output_root):
    for root, _, files in os.walk(input_folder):
        for file in files:
            if file.lower().endswith((".jpg", ".jpeg", ".png")):
                image_path = os.path.join(root, file)
                image = cv2.imread(image_path)

                if image is not None:
                    cropped = crop_largest_object(image)
```

```
                if cropped is not None:
                    relative_path = os.path.relpath(root, input_folder)
                    output_subfolder = os.path.join(output_root, relative_path)
                    os.makedirs(output_subfolder, exist_ok=True)
                    output_path = os.path.join(output_subfolder, file)
                    cv2.imwrite(output_path, cropped)

# Call the recursive function to process subfolders
process_subfolders(input_folder, output_root)

print("Cropped objects saved in:", output_root)
```

B.2 Processed Crop Rice Grain Images by Python Code in Jupyter Notebook

```
from PIL import Image
import os

# Source directory containing the images
source_dir = r'C:\Users\Administrator\Desktop\Cropped_Objects_All'
# Destination directory to save the processed images
destination_dir = r'C:\Users\Administrator\Desktop\Processed_Crop_Images'
# Create the destination directory if it doesn't exist
if not os.path.exists(destination_dir):
    os.makedirs(destination_dir)

# Iterate through subfolders in the source directory
for root, dirs, files in os.walk(source_dir):
    for file in files:
        if file.lower().endswith(('.jpg', '.jpeg', '.png', '.gif', '.bmp')):
            # Load the source image
            source_image = Image.open(os.path.join(root, file))

            # Create a blank black image of size 250x250
            new_image = Image.new('RGB', (250, 250), (0, 0, 0))

            # Calculate the position to paste the image to center it
            paste_x = (250 - source_image.width) // 2
            paste_y = (250 - source_image.height) // 2

            # Paste the source image onto the new image
            new_image.paste(source_image, (paste_x, paste_y))

            # Save the pasted image in the destination directory
            new_image.save(os.path.join(destination_dir, file))

print("Image processing and saving complete.")
```

B.3 Processed Image using Canny Edge Detection by Python Code in Jupyter Notebook

```
import cv2
import os

def apply_canny(image_path, output_path):
    img = cv2.imread(image_path, 0)
    edges = cv2.Canny(img, 100, 200)
    cv2.imwrite(output_path, edges)

root_dir = 'C:/Users/Administrator/Desktop/Rice_Image_Dataset'
output_folder = 'C:/Users/Administrator/Desktop/Rice_Image_Dataset_Canny'
# Specify the new folder path

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

for root, dirs, files in os.walk(root_dir):
    for file in files:
        if file.endswith(".jpg"):
            img_path = os.path.join(root, file)
            out_path = os.path.join(output_folder, "canny_" + file)
            apply_canny(img_path, out_path)
```

B.4 Processed Image using Sobel Edge Detection By Python code in Jupyter Notebook

```
import cv2
import os

path = "C:/Users/Administrator/Desktop/Rice_Image_Dataset"
output_folder = "C:/Users/Administrator/Desktop/Rice_Image_Dataset_Sobel"
# Specify the new folder path

# Create the output folder if it doesn't exist
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# Loop through all subdirectories and files in the given path
for root, dirs, files in os.walk(path):
    for file in files:
        if file.lower().endswith(".jpg") or file.lower().endswith(".png"):

            # Read the image
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path)

            # Apply Sobel edge detection
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            edges_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
            edges_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)
            edges = cv2.magnitude(edges_x, edges_y)
            edges = cv2.normalize(edges, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

            # Save the result in the output folder
            output_path = os.path.join(output_folder, "sobel_" + file)
            cv2.imwrite(output_path, edges)
```

B.5 Processed Image using Ridge Detection By Python code in Jupyter Notebook

```
import cv2
import os
import matplotlib.pyplot as plt
from skimage import filters
from tqdm import tqdm

path = ”C:/Users/Administrator/Desktop/Rice_Image_Dataset”

def plot_images(*images):
    images = list(images)
    n = len(images)
    fig , ax = plt.subplots(ncols=n, sharey=True)
    for i , img in enumerate(images):
        ax[i].imshow(img, cmap=’gray’)
        ax[i].axis(’off’)
    plt.subplots_adjust(left=0.03, bottom=0.03, right=1.97, top=1.97)
    plt.show()

# Loop through all subdirectories and files in the given path
for root, dirs, files in tqdm(os.walk(path)):
    for file in files:
        if file.lower().endswith(”.jpg”) or file.lower().endswith(”.png”):

            # Read the image
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
            img2 = cv2.imread(img_path)
            img3 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

            # Apply Adaptive Threshold detection
            edges2= filters.sobel(img)
            low2 = 0.07
```

```
high2 = 0.08
hyst2= filters.apply_hysteresis_threshold(edges2, low2, high2)
#adaptive_thresh = cv2.adaptiveThreshold(img, 255,
 cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)

# Save the result
output_path = os.path.join(root, "adaptive_" + file)
plot_images(img3,hyst2)
```

B.6 Processed Image using Texture Detection By Python code in Jupyter Notebook

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def apply_gabor_filter(image, ksize=31, sigma=5.0, theta=0.0, lambd=10.0, gamma=0.5):
    gabor_kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lambd, gamma, 0, ktype=cv2.CV_32F)
    filtered_image = cv2.filter2D(image, cv2.CV_8UC3, gabor_kernel)
    return filtered_image

def process_image(image_path, output_folder):
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

    mx = (0, 0, 0, 0)
    mx_area = 0
    for cont in contours:
        x, y, w, h = cv2.boundingRect(cont)
        area = w * h
        if area > mx_area:
            mx = x, y, w, h
```

```
            mx_area = area
        x, y, w, h = mx
        crop_img = image[y:y+h, x:x+w]

        # Apply histogram equalization
        equalized_image = cv2.equalizeHist(crop_img)
        texture_image = apply_gabor_filter(equalized_image)

        # Create output folder if it doesn't exist
        os.makedirs(output_folder, exist_ok=True)

        # Save the processed image
        result_path = os.path.join(output_folder, os.path.basename(image_path))
        cv2.imwrite(result_path, texture_image)

        return result_path

# Specify input and output folders
input_folder = 'C:/Users/Administrator/Desktop/Rice_Image_Dataset'
output_folder = 'C:/Users/Administrator/Desktop/Texture_Images'

# Process all images in the subfolders
for root, dirs, files in os.walk(input_folder):
    for file in files:
        if file.lower().endswith(('.png', '.jpg', '.jpeg')):
            image_path = os.path.join(root, file)
            process_image(image_path, output_folder)

print("Processing complete.")
```

B.7 Processed Image using Histogram Equalization, By Python code in Jupyter Notebook

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def equalize_and_save(image_path, save_path):
    # Load the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Thresholding
    thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    mx = (0, 0, 0, 0)
    # biggest bounding box so far
    mx_area = 0
    for cont in contours:
        x, y, w, h = cv2.boundingRect(cont)
        area = w * h
        if area > mx_area:
            mx = x, y, w, h
            mx_area = area
    x, y, w, h = mx
    crop_img = image[y:y + h, x:x + w]

    # Apply histogram equalization
    equalized_image = cv2.equalizeHist(crop_img)

    # Save the equalized image
    save_image_path = os.path.join(save_path, os.path.basename(image_path))
    cv2.imwrite(save_image_path, equalized_image)

# Folder path containing subfolders with images
```

```
main_folder_path = "C:/Users/Administrator/Desktop/Rice_Image_Dataset"

# Path for the new folder to save the result images
result_folder_path = "C:/Users/Administrator/Desktop/Equalized_Images"
os.makedirs(result_folder_path, exist_ok=True)

# Loop through all subfolders
for subfolder_name in os.listdir(main_folder_path):
    subfolder_path = os.path.join(main_folder_path, subfolder_name)

    if os.path.isdir(subfolder_path):
        # Loop through all images in the subfolder
        for filename in os.listdir(subfolder_path):
            if filename.endswith(".jpg") or filename.endswith(".png"):
                image_path = os.path.join(subfolder_path, filename)
                equalize_and_save(image_path, result_folder_path)
```

B.8 Processed Image using Laplacian Filter (Image Enhancement) by Python code in Jupyter Notebook

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def enhance_texture(image):
    laplacian = cv2.Laplacian(image, cv2.CV_64F)
    sharpened = np.uint8(np.clip(image - laplacian, 0, 255))
    return sharpened

def process_and_save(image_path, save_folder):
    # Load the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
```

```
    # Thresholding
    thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    mx = (0, 0, 0, 0)
    mx_area = 0
    for cont in contours:
        x, y, w, h = cv2.boundingRect(cont)
        area = w * h
        if area > mx_area:
            mx = x, y, w, h
            mx_area = area
    x, y, w, h = mx
    crop_img = image[y:y + h, x:x + w]

    # Apply histogram equalization
    equalized_image = cv2.equalizeHist(crop_img)

    # Enhance the texture of the cropped image
    enhanced_texture = enhance_texture(equalized_image)

    # Save the enhanced image
    save_path = os.path.join(save_folder, os.path.basename(image_path))
    cv2.imwrite(save_path, enhanced_texture)

# Folder path containing subfolders with images
main_folder_path = "C:/Users/Administrator/Desktop/Rice_Image_Dataset"

# Create a subfolder named 'enhance1' on the desktop
save_folder_path = "C:/Users/Administrator/Desktop/enhance1"
os.makedirs(save_folder_path, exist_ok=True)

# Loop through all subfolders
for subfolder_name in os.listdir(main_folder_path):
    subfolder_path = os.path.join(main_folder_path, subfolder_name)

    if os.path.isdir(subfolder_path):
```

```
    # Loop through all images in the subfolder
    for filename in os.listdir(subfolder_path):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(subfolder_path, filename)
            process_and_save(image_path, save_folder_path)
```

B.9 Processed Image using Gaussian blur (Image Enhancement) by Python code in Jupyter Notebook

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

def enhance_texture2(image):
    # Apply Laplacian filter for sharpening
    laplacian = cv2.GaussianBlur(image, (25, 25), 0)
    sharpened = np.uint8(np.clip(image - laplacian, 0, 500))

    return sharpened

def process_and_save(image_path, save_folder):
    # Load the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Thresholding
    thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    contours, hierarchy = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    mx = (0, 0, 0, 0)
    mx_area = 0
    for cont in contours:
        x, y, w, h = cv2.boundingRect(cont)
        area = w * h
        if area > mx_area:
```

```
            mx = x, y, w, h
            mx_area = area
    x, y, w, h = mx
    crop_img = image[y:y + h, x:x + w]

    # Apply histogram equalization
    equalized_image = cv2.equalizeHist(crop_img)

    # Enhance the texture of the cropped image
    enhanced_texture2 = enhance_texture2(equalized_image)

    # Save the enhanced image
    save_path = os.path.join(save_folder, os.path.basename(image_path))
    cv2.imwrite(save_path, enhanced_texture2)

# Folder path containing subfolders with images
main_folder_path = "C:/Users/Administrator/Desktop/Rice_Image_Dataset"

# Create a subfolder named 'enhance1' on the desktop
save_folder_path = "C:/Users/Administrator/Desktop/enhance2"
os.makedirs(save_folder_path, exist_ok=True)

# Loop through all subfolders
for subfolder_name in os.listdir(main_folder_path):
    subfolder_path = os.path.join(main_folder_path, subfolder_name)

    if os.path.isdir(subfolder_path):
        # Loop through all images in the subfolder
        for filename in os.listdir(subfolder_path):
            if filename.endswith(".jpg") or filename.endswith(".png"):
                image_path = os.path.join(subfolder_path, filename)
                process_and_save(image_path, save_folder_path)
```

B.10 Example of Shape Feature Extraction by Python Code in Jupyter Notebook

```
from PIL import Image
import numpy as np
import os
from skimage import measure, morphology, filters
import pandas as pd
import scipy.stats as stats
from skimage.measure import shannon_entropy

# Define the path to your dataset folder
dataset_path = 'C:/Users/Administrator/Desktop/enhance1'

# Create empty lists to store images, labels, and shape features
images = []
labels = []
shape_features = []

# Loop through each subdirectory (each class)
for subdir in os.listdir(dataset_path):
    subdir_path = os.path.join(dataset_path, subdir)
    if os.path.isdir(subdir_path):
        for image_file in os.listdir(subdir_path):
            image_path = os.path.join(subdir_path, image_file)
            if image_file.endswith(('.jpg', '.png', '.jpeg')):
# Check if it's an image file
                # Open and resize the image
                img = Image.open(image_path).resize((224, 224))
                images.append(np.array(img))
                labels.append(subdir)  # You can assign labels based on the subdirectory name

                # Convert the image to grayscale (2D)
                grayscale_image = np.array(img.convert('L'))
```

```python
# Compute shape features using scikit-image's regionprops
props = measure.regionprops(grayscale_image)

# Calculate standard deviation of pixel values
std_dev = np.std(grayscale_image)

# Calculate peak value (maximum pixel value)
peak_value = np.max(grayscale_image)

# Calculate minimum and maximum gray values
min_gray_value = np.min(grayscale_image)
max_gray_value = np.max(grayscale_image)

# Calculate edginess using Sobel filter
edge_image = filters.sobel(grayscale_image)
edginess = np.mean(edge_image)

# Calculate normalized center of mass
com = props[0].local_centroid
normalized_com = (com[0] / grayscale_image.shape[0], com[1] / grayscale_image.shape[1])

# Calculate eccentricity
eccentricity = props[0].eccentricity

# Calculate solidity
solidity = props[0].solidity

# Calculate compactness
compactness = (props[0].perimeter ** 2) / (4 * np.pi * props[0].area)

# Calculate shape factor
shape_factor = (props[0].perimeter ** 2) / (props[0].area)

# Calculate equivalent diameter
equivalent_diameter = props[0].equivalent_diameter
```

```
        # Calculate entropy
        entropy = shannon_entropy(grayscale_image)

        shape_feature = {
            "Image": image_file,
            "Label": subdir,
            "Area": props[0].area,
            "Perimeter": props[0].perimeter,
            "Extent": props[0].extent,
            "ConvexArea": props[0].convex_area,
            "AspectRatio": props[0].minor_axis_length / props[0].major_axis_length,
            "Kurtosis": stats.kurtosis(grayscale_image.ravel()),
            "Skewness": stats.skew(grayscale_image.ravel()),
            "MajorAxis": props[0].major_axis_length,
            "MinorAxis": props[0].minor_axis_length,
            "StdDev": std_dev,
            "PeakValue": peak_value,
            "MinGrayValue": min_gray_value,
            "MaxGrayValue": max_gray_value,
            "Edginess": edginess,
            "NormalizedCOM_X": normalized_com[0],
            "NormalizedCOM_Y": normalized_com[1],
            "Eccentricity": eccentricity,  # Eccentricity feature
            "Solidity": solidity,  # Solidity feature
            "Compactness": compactness,  # Compactness feature
            "ShapeFactor": shape_factor,  # Shape factor feature
            "EquivalentDiameter": equivalent_diameter,
# Equivalent diameter feature
            "Entropy": entropy,  # Entropy feature
            # Add more shape features here
        }
        shape_features.append(shape_feature)

# Create a DataFrame to store the shape features
shape_df = pd.DataFrame(shape_features)
```

```
# Save shape features to CSV and XLSX files
shape_csv_path = 'C:/Users/Administrator/Desktop/shape_features_EH1.csv'
shape_xlsx_path = 'C:/Users/Administrator/Desktop/shape_features_EH1.xlsx'

shape_df.to_csv(shape_csv_path, index=False)
shape_df.to_excel(shape_xlsx_path, index=False, engine='openpyxl')

print("Shape features saved as CSV:", shape_csv_path)
print("Shape features saved as XLSX:", shape_xlsx_path)
```

B.11 Example of Texture Feature Extraction by Python Code in Jupyter Notebook

```
import os
import cv2
import numpy as np
from skimage.feature import graycomatrix, graycoprops
import pandas as pd

# Function to extract texture attributes from an image
def extract_texture_attributes(image_path):
    # Read the image
    image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

    # Calculate the co-occurrence matrix
    co_occurrence_matrix = graycomatrix(image, [1], [0], symmetric=True, normed=True)

    # Calculate texture attributes from the co-occurrence matrix
    correlation = graycoprops(co_occurrence_matrix, 'correlation')[0, 0]
    dissimilarity = graycoprops(co_occurrence_matrix, 'dissimilarity')[0, 0]
    energy = graycoprops(co_occurrence_matrix, 'energy')[0, 0]
    entropy = -np.sum(co_occurrence_matrix * np.log(co_occurrence_matrix + np.finfo(float).eps))
    contrast = graycoprops(co_occurrence_matrix, 'contrast')[0, 0]
    homogeneity = graycoprops(co_occurrence_matrix, 'homogeneity')[0, 0]
```

```
    # Calculate gray level moments
    uniformity = np.sum(co_occurrence_matrix ** 2)
    mean = np.mean(image)
    variance = np.var(image)
    skewness = np.mean(((image - mean) ** 3) / (variance ** 1.5))
    kurtosis = np.mean(((image - mean) ** 4) / (variance ** 2))

    # Get the image name (file name without extension)
    image_name = os.path.splitext(os.path.basename(image_path))[0]

    return [correlation, dissimilarity, energy, entropy, contrast, homogeneity, uniformity, mean,
     variance, skewness, kurtosis, image_name]

# Define the folder containing the images
folder_path = "C:/Users/Administrator/Desktop/Processed_Crop_Images"

# Initialize lists to store attributes and class labels
data = []

# Iterate through subfolders and images
for subfolder in os.listdir(folder_path):
    subfolder_path = os.path.join(folder_path, subfolder)

    if os.path.isdir(subfolder_path):
        for image_file in os.listdir(subfolder_path):
            image_path = os.path.join(subfolder_path, image_file)

            # Extract texture attributes from the image
            attributes = extract_texture_attributes(image_path)

            # Append the class label (subfolder name) to the attributes
            attributes.append(subfolder)

            # Add the attributes to the data list
            data.append(attributes)
```

```python
# Create a Pandas DataFrame
columns = ["Correlation", "Dissimilarity", "Energy", "Entropy", "Contrast", "Homogeneity", "Uniformity",
           "Mean", "Variance", "Skewness", "Kurtosis", "ImageName",   "Class"]
df = pd.DataFrame(data, columns=columns)

# Save the DataFrame to an Excel file
output_file = "C:/Users/Administrator/Desktop/texture_attributes.xlsx"
df.to_excel(output_file, index=False)

print(f"Texture attributes saved to {output_file}")
```

B.12 Example of Data Normalization by Python code in Jupyter Notebook

```
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset from the Excel file
file_path = "F:/Thesis/shape_texture_features_Ridge.xlsx"
data = pd.read_excel(file_path)

# Separate the features (X) and labels (y)
X = data.drop(columns=['Image', 'Label'])
y = data['Label']

# Normalize the features using StandardScaler
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Create a new DataFrame with the normalized features and labels
normalized_data = pd.DataFrame(data=X_normalized, columns=X.columns)
normalized_data['Image'] = data['Image']
normalized_data['Label'] = y

# Save the normalized data to a new Excel file
normalized_file_path = "F:/Thesis/normalized_shape_texture_features_Ridge.xlsx"
normalized_data.to_excel(normalized_file_path, index=False)
```

B.13 Example of Decision Tree Modeling by Python Code in Jupyter Notebook

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score, cohen_kappa_score, precision_recall_fscore_support,
confusion_matrix
import time

# Load the dataset
data_path = "C:/Users/Administrator/Desktop/normalized_shape_texture_features_Canny.xlsx"
df = pd.read_excel(data_path)

# Split the dataset into features and labels
X = df.drop(['Image', 'Label'], axis=1)
y = df['Label']

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier()

# Measure the start time
start_time = time.time()

# Perform 10-fold cross-validation with predictions
y_pred = cross_val_predict(clf, X, y, cv=10)

# Calculate the performance metrics
accuracy = accuracy_score(y, y_pred)
kappa = cohen_kappa_score(y, y_pred)

# Calculate precision, recall, and F1 score for each class
precision, recall, fscore, support = precision_recall_fscore_support(y, y_pred)

# Calculate the confusion matrix
```

```
confusion = confusion_matrix(y, y_pred)

# Calculate the total time taken
end_time = time.time()
total_time = end_time - start_time

# Display the results
print(f"Total time taken: {total_time:.4f} seconds")
print(f"Accuracy: {accuracy:.4f}")
print(f"Cohen's Kappa: {kappa:.4f}")

# Display precision, recall, and F1 score for each class
for class_label, prec, rec, f1 in zip(range(len(precision)), precision, recall, fscore):
    print(f"Class {class_label}: Precision = {prec:.4f}, Recall = {rec:.4f},
                                                            F1 Score = {f1:.4f}")

print("Confusion Matrix:")
print(confusion)
```

B.14 Example of Naïve Bayes Modeling by Python code in Jupyter Notebook

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import accuracy_score, cohen_kappa_score, precision_score, recall_score,
confusion_matrix, f1_score
import time

# Load the dataset
data_path = "C:/Users/Administrator/Desktop/normalized_shape_texture_features_Canny.xlsx"
df = pd.read_excel(data_path)

# Split the dataset into features and labels
```

```
X = df.drop(['Image', 'Label'], axis=1)
y = df['Label']

# Initialize the Naive Bayes Classifier (GaussianNB)
clf = GaussianNB()

# Measure the start time
start_time = time.time()

# Perform 10-fold cross-validation with predictions
y_pred = cross_val_predict(clf, X, y, cv=10)

# Calculate the performance metrics
accuracy = accuracy_score(y, y_pred)
kappa = cohen_kappa_score(y, y_pred)
precision = precision_score(y, y_pred, average='weighted')
recall = recall_score(y, y_pred, average='weighted')
fscore = f1_score(y, y_pred, average='weighted')

# Calculate precision, recall, and F1 score for each class
precision_per_class = precision_score(y, y_pred, average=None)
recall_per_class = recall_score(y, y_pred, average=None)
fscore_per_class = f1_score(y, y_pred, average=None)

# Calculate the confusion matrix
confusion = confusion_matrix(y, y_pred)

# Calculate the total time taken
end_time = time.time()
total_time = end_time - start_time

# Display the results
print(f"Total time taken: {total_time:.4f} seconds")
print(f"Accuracy: {accuracy:.4f}")
print(f"Cohen's Kappa: {kappa:.4f}")
print(f"Precision: {precision:.4f}")
```

```
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {fscore:.4f}")

# Display precision, recall, and F1 score for each class
for class_label, prec, rec, f1 in zip(range(len(precision_per_class)),
                            precision_per_class, recall_per_class, fscore_per_class):
    print(f"Class {class_label}: Precision = {prec:.4f}, Recall = {rec:.4f},
                                                            F1 Score = {f1:.4f}")

print("Confusion Matrix:")
print(confusion)
```

B.15 Example of K-Nearest Neighbors Modeling by Python code in Jupyter Notebook

```
import pandas as pd
from sklearn.model_selection import cross_val_predict, StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, cohen_kappa_score, precision_score,
                                            recall_score, confusion_matrix, f1_score
import time

# Load the dataset
dataset_path = "C:/Users/Administrator/Desktop/normalized_shape_texture_features_Canny.xlsx"
df = pd.read_excel(dataset_path)

# Extract features and labels
X = df.drop(['Image', 'Label'], axis=1)
# Assuming 'Image' and 'Label' are the column names for ID and Class
y = df['Label']

# Initialize the K-NN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)
# You can adjust the number of neighbors as needed
```

```
# Perform 10-fold cross-validation with shuffling
start_time = time.time()
stratified_kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Perform predictions during cross-validation
y_pred = cross_val_predict(knn_classifier, X, y, cv=stratified_kfold)

# Calculate and print the time taken for training and cross-validation
total_time = time.time() - start_time
print(f"Total time taken: {total_time:.4f} seconds")

# Evaluate performance metrics
accuracy = accuracy_score(y, y_pred)
kappa = cohen_kappa_score(y, y_pred)
precision = precision_score(y, y_pred, average='weighted')
recall = recall_score(y, y_pred, average='weighted')
fscore = f1_score(y, y_pred, average='weighted')

print(f"Accuracy: {accuracy:.4f}")
print(f"Kappa: {kappa:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {fscore:.4f}")

# Display precision, recall, and F1 score for each class
precision_per_class = precision_score(y, y_pred, average=None)
recall_per_class = recall_score(y, y_pred, average=None)
fscore_per_class = f1_score(y, y_pred, average=None)

for class_label, prec, rec, f1 in zip(range(len(precision_per_class)),
                        precision_per_class, recall_per_class, fscore_per_class):
    print(f"Class {class_label}: Precision = {prec:.4f}, Recall = {rec:.4f},
                                                    F1 Score = {f1:.4f}")

# Display confusion matrix
```

```
conf_matrix = confusion_matrix(y, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

B.16 Example of Support Vector Machine Modeling by Python code in Jupyter Notebook

```
import pandas as pd
from sklearn.model_selection import cross_val_predict, KFold
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, cohen_kappa_score, precision_recall_fscore_support,
                                                                                  confusion_matrix
from sklearn.preprocessing import StandardScaler
import time

# Load the dataset
data = pd.read_excel("C:/Users/Administrator/Desktop/normalized_shape_texture_features_Canny.xlsx")

# Extract features (excluding 'Image' and 'Label' columns)
X = data.drop(['Image', 'Label'], axis=1)

# Extract labels
y = data['Label']

# Initialize the SVM classifier
classifier = SVC(kernel='linear')  # You can change the kernel type as needed

# Standardize features (optional but recommended for SVM)
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Start the timer
start_time = time.time()
```

```
# Perform 10-fold cross-validation with shuffling and get predicted labels
kf = KFold(n_splits=10, shuffle=True, random_state=42)
predicted_labels = cross_val_predict(classifier, X, y, cv=kf)

# Stop the timer
end_time = time.time()
total_time = end_time - start_time

# Calculate accuracy, precision, recall, and F1 score for each class
accuracy = accuracy_score(y, predicted_labels)
precision, recall, fscore, support = precision_recall_fscore_support(y, predicted_labels)

# Calculate Cohen's Kappa
kappa = cohen_kappa_score(y, predicted_labels)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y, predicted_labels)

# Display results with four decimal places
print(f"Total time taken: {total_time:.4f} seconds")
print(f"Accuracy: {accuracy:.4f}")
print(f"Cohen's Kappa: {kappa:.4f}")

# Display precision, recall, and F1 score for each class
for class_label, prec, rec, f1 in zip(range(len(precision)), precision, recall, fscore):
    print(f"Class {class_label}: Precision = {prec:.4f}, Recall = {rec:.4f},
                                                          F1 Score = {f1:.4f}")

print("Confusion Matrix:")
print(conf_matrix)
```

B.17 Example of Gradient Boosted Tree Modeling by Python code in Jupyter Notebook

```
import pandas as pd
from sklearn.metrics import cohen_kappa_score, accuracy_score, precision_score, recall_score,
                                                                f1_score, confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_predict, KFold
import time

# Load your dataset from the provided file path
file_path = "C:/Users/Administrator/Desktop/normalized_shape_texture_features_EH1.xlsx"
df = pd.read_excel(file_path)

# Use 'Image' and 'Label' for column names
X = df.drop(['Image', 'Label'], axis=1)

# Set 'Label' as the target variable
y = df['Label']

# Create a GradientBoostingClassifier
gradient_booster = GradientBoostingClassifier(n_estimators=50, learning_rate=0.1, max_depth=5)

# Start the timer
start_time = time.time()

# Use 10-fold cross-validation with shuffling
kf = KFold(n_splits=10, shuffle=True, random_state=42)
y_pred = cross_val_predict(gradient_booster, X, y, cv=kf)

# Stop the timer
end_time = time.time()
total_time = end_time - start_time

# Calculate performance metrics
```

```
accuracy = accuracy_score(y, y_pred)
kappa = cohen_kappa_score(y, y_pred)
precision_per_class = precision_score(y, y_pred, average=None)
recall_per_class = recall_score(y, y_pred, average=None)
fscore_per_class = f1_score(y, y_pred, average=None)

# Generate a confusion matrix
conf_matrix = confusion_matrix(y, y_pred)

# Print the precision, recall, and F1 score for each class
print(f"Total Time: {total_time:.2f} seconds")
print(f"Accuracy: {accuracy:.4f}")
print(f"Kappa: {kappa:.4f}")

# Display precision, recall, and F1 score for each class
for class_label, prec, rec, f1 in zip(range(len(precision_per_class)), precision_per_class,
                                      recall_per_class, fscore_per_class):
    print(f"Class {class_label}: Precision = {prec:.4f}, Recall = {rec:.4f},
                                      F1 Score = {f1:.4f}")

# Display confusion matrix
print("Confusion Matrix:")
print(conf_matrix)
```

CURRICULUM VITAE

NAME : Piyanart Boonramart **GENDER :** Female

EDUCATION BACKGROUND:

- Bachelor of Science (Mathematics), Suranaree University of Technology, Thailand, 2020

SCHOLARSHIP:

- Outstanding Academic Performance Suranaree University of Technology Scholarship

CONFERENCE:

- Boonramart, P., Koatborom, P., Rodjanadid, B., and Tanthanuch, J. (2022) An Application of Image Processing and Machine Learning for Rice Varieties Classification., *The Proceedings of The 10th Nonsi Isan National Academic Conference*, Kasetsart University Chalermphrakiat Campus, Sakon Nakhon, 26 November 2022, 711-721.

- Udomjetjamnong, K., Boonramart, P., and Tanthanuch, J. (2023) Leveraging Three Image Processing Techniques and Machine Learning for Milled Rice Variety Classification., *The Proceedings of The 20th International and National Conference on Applied Computer Technology and Information Systems (ACTIS)*. 25 August 2023, 56-60.

EXPERIENCE:

- Teaching assistant in Suranaree University of Technology, Calculus I, Calculus II, Calculus III, Differential Equations I and Differential Equations for Civil Engineers.