Odesa I.I.Mechnikov
National University

University of L'Aquila

Double-Degree Master's Programme - InterMaths
Applied and Interdisciplinary Mathematics

Applied Mathematics	Master of Science Mathematical Engineering
ODESA I.I.MECHNIKOV NATIONAL UNIVERSITY ()	UNIVERSITY OF L'AQUILA (UAQ)

Master's Thesis

Quality classification of molded products via Machine Learning and Control Theory

Supervisor
Alessandro D'Innocenzo

Candidate
Danylo Savchak

Student ID (UAQ): 280160
Student ID (): 5157759

Academic Year 2022/2023

Contents

0.1 Introduction 1
0.2 Quality classification in injection molding process 6
0.3 Machine Learning techniques for regression and classification 9
0.4 Control theory techniques 32
0.5 Experimental validation 36
0.6 Conclusion and future work 79

0.1 Introduction

The general problem

The general problem of quality classification of molded products refers to the task of evaluating and categorizing the quality of products that have been manufactured using a molding process, such as injection molding, blow molding, or rotational molding.

Molded products are manufactured by injecting molten material into a mold cavity and allowing it to solidify into the desired shape. Quality classification is crucial to ensure that these molded products meet the required standards, specifications, and customer expectations.

Several challenges are associated with the quality classification of molded products:

1. Defect detection: Detecting defects in molded products can be challenging due to the complexity of the molded shapes and the potential presence of hidden or internal defects. Common defects include surface imperfections, such as flash (excess material), sink marks, warping, air traps, or incomplete filling of the mold cavity. These defects can impact the product's functionality, aesthetics, and structural integrity.

2. Variation in dimensions and tolerances: Molded products can have varying dimensions and tolerances due to factors like material shrinkage, mold wear, or process variations. Classifying the quality of molded products involves assessing whether they meet the specified dimensional requirements and tolerance limits.

3. Material properties: The quality of molded products is influenced by the material properties used in the molding process, such as the melt flow index, viscosity, hardness, or color. Variations in material properties can affect the product's performance and appearance, requiring careful evaluation during the classification process.

4. Sample representation: Determining the quality of molded products requires representative samples from production batches. Ensuring that the samples

accurately represent the entire batch and provide a comprehensive overview of the quality is essential for accurate classification.

To address these challenges, various approaches can be employed for quality classification of molded products:

1. Visual inspection: Human inspectors or automated systems can visually examine the molded products for surface defects, dimensional accuracy, and overall appearance. This inspection can be aided by cameras, sensors, or machine vision algorithms to detect defects and assess quality parameters.

2. Non-destructive testing: Techniques like X-ray or ultrasound scanning can be used to inspect molded products for internal defects without damaging or disassembling them. These methods can reveal hidden defects and provide insights into the structural integrity of the products.

3. Statistical process control: By monitoring and analyzing process data, such as temperature, pressure, or cycle time, statistical process control methods can help identify process variations and trends that may affect the quality of molded products. This enables manufacturers to take corrective actions and maintain consistent quality.

4. Mechanical and functional testing: Testing the mechanical properties and functional performance of molded products can provide insights into their quality. This may involve conducting tests like tensile strength, impact resistance, or functionality checks to ensure that the products meet the required standards and specifications.

Literature review

Early research works for quality prediction in plastic injection molding are dated back to the early 2000s. For example, Sadeghi proposed a neural network and a multi-layer perceptron (MLP) using four features (melt flow rate, injection pressure, mold temperature, and melt temperature) to estimate actual injection pressure,

filling time, and part soundness, with a hidden layer composed of two neurons. The author tested the proposed methodology on simulated data.

Similarly, Chen et al. used a neural network-based approach to estimate the weight of the samples. They combined self-organizing maps (SOM), using three features as inputs (injection stroke curve, injection velocity curve, and pressure curve), and an MLP, using six features as inputs (injection time, VP switch position, packing pressure, injection velocity, packing time, and injection stroke), on 160 real samples (120 for training, 40 for testing), achieving a root mean squared error of 0.0017.

Ribeiro proposed the use of support vector machine (SVM) classifiers to predict faults in automotive parts produced with injection molding. They used six features as the input (cycle time, metering time, injection time, barrel temperature before nozzle, cushion, and injection velocity) and compared their methodology with a neural network to predict six types of faults, obtaining an accuracy between 87% and 100% (depending on the type of fault to be classified).

Liu et al. proposed an ensemble of neural networks to predict the shrinkage of thin shell parts as a measure of their quality. They used mold temperature, holding pressure, and holding time as inputs for the neural networks, taking 70 samples for training, 15 samples for validation, and 15 samples for testing, achieving a top root mean squared error of 0.008.

S. Ameli et al. worked on defect classification in injection-molded parts using machine learning techniques. They developed a system to classify different types of defects in injection-molded parts using machine learning algorithms such as support vector machines (SVM) and random forests. They achieved over 90% accuracy in classifying the defects.

M. Amiri et al. worked on machine learning-based quality prediction in injection molding. They utilized an ensemble of machine learning models, including decision trees and artificial neural networks, to predict the quality attributes of injection-molded parts. They achieved approximately 95% accuracy in predicting the quality attributes.

M. Yuan et al. worked on surface defect detection in molded plastic parts using

convolutional neural networks. They employed convolutional neural networks (CNNs) to detect and classify surface defects in molded plastic parts. They achieved around 92% accuracy in detecting and classifying the surface defects.

H. Park et al. worked on quality control of blow-molded bottles using machine vision and machine learning techniques. They developed a system combining machine vision and machine learning algorithms, such as k-nearest neighbors (KNN) and support vector machines (SVM), to classify bottles into different quality categories. They achieved approximately 90% accuracy in the bottle classification.

Y. Liu et al. worked on fault diagnosis of rotational molding process based on machine learning algorithms. They developed a system using machine learning algorithms, including decision trees and support vector machines (SVM), to detect and classify various faults in rotational molding processes. They achieved approximately 88% accuracy in detecting and classifying the faults.

These works demonstrate the achievements of different researchers in using machine learning and deep learning techniques for quality classification of molded products and the corresponding accuracies they were able to reach.

Work organization:

Next parts of work are organized in the next way:

Quality classification in injection molding process section is defining injection molded problems and explains problem formulation, what do we understand as the quality of the product. In *Machine Learning techniques for regression and classification* section, we delve into the machine learning techniques employed for regression and classification tasks in the context of quality classification in injection molding. Different algorithms, including decision trees, support vector machines (SVM), random forests, and others, are explored. The section provides an overview of each technique. *Experimental validation* section describes the methodology and setup used to evaluate the effectiveness of machine learning techniques for quality classification in injection molding. Details of the dataset used, including the number of samples, features, and ground truth labels, are provided. The section also outlines the preprocessing steps employed, such as data cleaning, feature

selection, and normalization. Furthermore, the evaluation metrics used to assess the performance of the machine learning models, such as accuracy, precision, recall, and F1 score, are discussed. In the *Conclusion and future work*, the findings of the study are summarized, highlighting the effectiveness of machine learning techniques for quality classification in injection molding. The section discusses the achieved results, including the accuracy and performance of the developed models. Moreover, potential areas for improvement and future research directions are outlined. This includes exploring advanced machine learning algorithms, incorporating additional features, and considering real-time implementation of the proposed models.

0.2 Quality classification in injection molding process

Plastic injection molding is a manufacturing process widely used in the industry, as it allows for the production of plastic objects characterized by complex geometries with high precision and productivity. Specifically, the process includes four phases, i.e., plasticization, injection, packing, and cooling, during which the polymer is melted, injected and compressed in the mold, and, eventually, cooled. Given that the quality of a molded product depends on the process parameters, as optimal values reduce the cycle time and increase the quality of the final product, extensive research has been devoted to predict the quality of products from the value of the production process parameters.

In this regard, this paper investigates the effectiveness of ML techniques to predict the quality of molded plastic products from the production process parameters, using real data from in-line industrial measurements. Specifically, the data comes from the production of plastic road lenses of "iGuzzini Illuminazione", a company located in Recanati, Italy, which specializes in providing plastic components for lighting devices. In fact, road lenses are regulated by the standard "UNI EN 13201-2:2016", which sets threshold values for the lighting uniformity (U_0). Currently, controlling the quality of the lenses, i.e., their lighting uniformity, requires a lens-by-lens photometric analysis conducted by specialized personnel in lab settings, demanding significant resources. Therefore, our aim is to understand if this control can be automated or, at least, integrated by using ML methods to estimate the level of the lighting uniformity of a lens from its process parameters collected by the sensors of the injection molding machine during the production.

The experimental evaluation consists of tests executed on a dataset composed of 1451 feature vectors corresponding to 1451 road lenses produced by iGuzzini Illuminazione. The data was collected from the sensors of the "Engel E-MAC 310/100" injection machine (https://www.engelglobal.com/en/us/index.html, accessed on the 15 May 2022), with the "TIG" Manufacturing Execution System (MES) (https://www.tig-mes.com/en/, accessed on the 15 May 2022), during five different production days, namely, the 18th, 19th, and 20th of September 2019, the 7th of February 2020, and the 20th of May 2020. Each feature vector includes 13

injection molding process parameters (Table 0.1), namely, the melt temperature ($°C$), the mold temperature ($°C$), the filling time (s), the plasticizing time (s), the cycle time (s), the closing force (N), the clamping force peak value (N), the torque peak value ($N \cdot m$), the torque mean value ($N \cdot m$), the back pressure peak value (bar), the injection pressure peak value (bar), the screw position at the end of the hold pressure (cm), and the shot volume (cm^3).

Table 0.1. The process parameters included in the feature vector of each sample in the dataset.

Feature	Unit	Description
Melt temperature	$°C$	Temperature of polymer before the injection in the mold
Molt temperature	$°C$	Temperature of the mold
Filling time	s	Time to fill the mold
Plasticizing time	s	Time to plasticize the product how
Cycle time	s	Time to complete the entire process of the product
Closing force	N	Closing force of the mold
Clamping force peak value	N	Peak value of the closing force of the mold
Torque peak value current cycle	$N \cdot m$	Peak value of the torque on the injection screw
Torque mean value current cycle	$N \cdot m$	Mean value of the torque on the injection screw
Specific back pressure peak value	Bar	Peak value of the resistance of the injection screw
Specific injection pressure peak value	Bar	Peak value of the injection pressure
Screw position at the end of hold pressure	cm	Position of the injection screw at the end of the holding cycle
Shot Volume	cm^3	Injection volume

Given that the standard “UNI EN 13201-2:2016” imposes a general uniformity

U0 greater than 0.4 for M1 lenses in motorized roads, we defined the following four quality classes for the samples of the dataset:

- Waste, with $U_0 < 0.4$. All the samples that exhibit a general uniformity less than 0.4 in the photometric analysis should be discarded, as they are not compliant to the standard. The label for this class is 1;
- Acceptable, with $0.4 <= U_0 < 0.45$. All the samples with a uniformity greater than or equal to 0.4 and less than 0.45 are considered acceptable by iGuzzini Illuminazione, as they comply with the standard. However, the target for the iGuzzini production is a higher quality. The label for this class is 2;
- Target, with $0.45 <= U_0 <= 0.5$. All the samples with a uniformity greater than or equal to 0.45 and less than or equal to 0.5 are considered optimal by the company. The label for this class is 3;
- Inefficient, with $U_0 > 0.5$. Even if the general uniformity is far greater than the standard threshold, producing lenses with a uniformity greater than 0.5 would result in the molding machine using more resources; therefore, this quality should be avoided, given that it is not required. The label for this class is 4.

Table 0.2 describes the division of the 1451 samples included in the dataset into the four classes: 370 lenses are labeled as Waste, 406 lenses as Acceptable, 310 as Target, and 360 as Inefficient.

Table 0.2. The number of samples for each class in the proposed dataset.

	Waste	**Acceptable**	**Target**	**Inefficient**
Label	1	2	3	4
#Samples	370	406	310	360

0.3 Machine Learning techniques for regression and classification

Linear Regression

Linear regression is used to study the linear relationship between a dependent variable (y) and one or more independent variables (X). The linearity of the relationship between the dependent and independent variables is an assumption of the model. The relationship is modeled through a random disturbance term (or, error variable) ϵ. The disturbance is primarily important because we are not able to capture every possible influential factor on the dependent variable of the model. To capture all the other factors, not included as independent variable, that affect the dependent variable, the disturbance term is added to the linear regression model.

In this way, the linear regression model takes the following form:

$$y = X\beta + \epsilon = [1\ x_1\ x_2\ x_3\ ...\ x_K] \cdot [\beta_0\ ...\ \beta_K]^T + \epsilon = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_K x_K + \epsilon \quad (0.1)$$

where

$$\beta_i \in R, i = 1, 2...K$$

are the **regression coefficients** of the model (which we want to estimate!), and K is the number of independent variables included. The equation is called the regression equation.

Example

Consider the following example of a linear regression model:

$$y = 2 + 1.5x + \epsilon$$

In the plot above, the blue dots represent the observed data points, and the red line represents the fitted linear regression model.

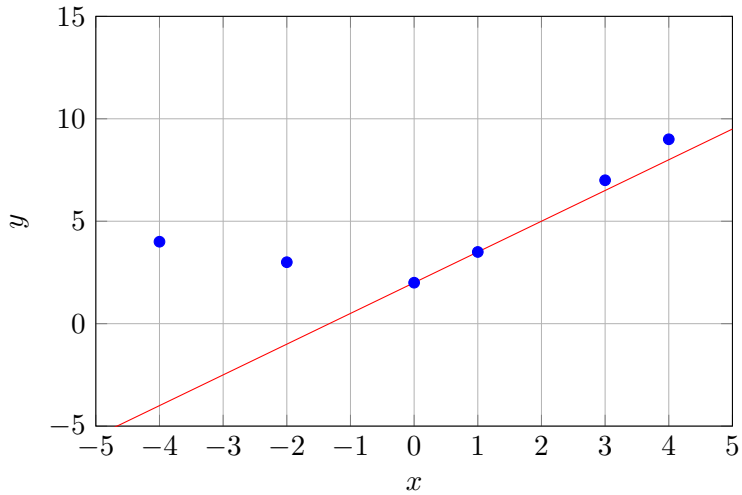


Figure 0.1. Example of a linear regression model

Prediction

Once the linear regression model is fitted to the data, it can be used to make predictions for new values of the independent variable x. By plugging in the new values into the linear equation, we can obtain the predicted values of the dependent variable y.

Polynomial Regression

Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and the dependent variable y is modeled as an n-degree polynomial. It is useful when the relationship between the variables cannot be accurately captured by a simple linear model.

Model Representation

The polynomial regression model can be represented as:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$$

where y is the dependent variable, x is the independent variable, $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients, n is the degree of the polynomial, and ϵ represents the error term.

Fitting the Polynomial

To fit a polynomial regression model to the data, we need to estimate the coefficients $\beta_0, \beta_1, \ldots, \beta_n$ that best represent the relationship between x and y. This can be done by minimizing the sum of squared errors between the predicted values and the actual values of the dependent variable.

Example

Consider the following example of a polynomial regression model with degree $n = 2$:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$

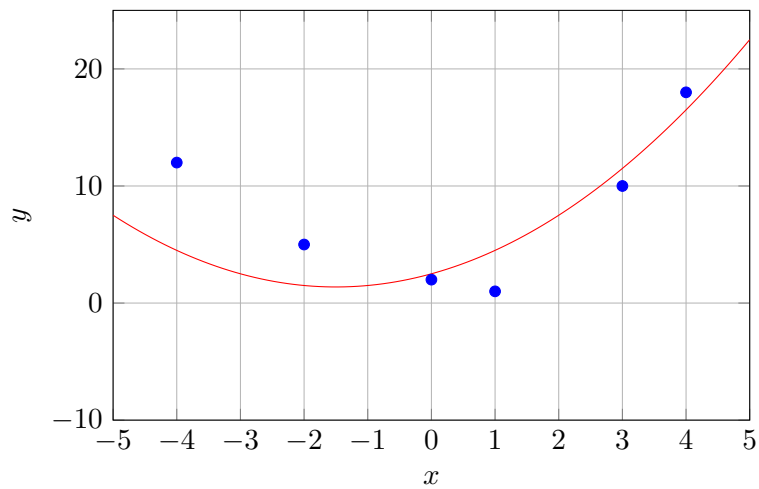


Figure 0.2. Example of a polynomial regression model with degree $n = 2$

In the plot above, the blue dots represent the observed data points, and the red curve represents the fitted polynomial regression model.

Prediction

Once the polynomial regression model is fitted to the data, it can be used to make predictions for new values of the independent variable x. By plugging in the new values into the polynomial equation, we can obtain the predicted values of the dependent variable y.

For do it we need to transform our data. For example lets see quadratic case:

$$y = \beta_0 + \beta_1 x_1 \rightarrow y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 \tag{0.2}$$

Ridge Regression

Ridge Regression is a regularized version of linear regression that introduces a penalty term to the loss function. It is used to mitigate the problem of multicollinearity in the data and helps to stabilize the model by reducing the impact of high-variance features.

Model Representation

The Ridge Regression model can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

where y is the dependent variable, $x_1, x_2, \ldots, x_n$ are the independent variables, $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients, and ϵ represents the error term.

Loss Function

The Ridge Regression loss function includes a penalty term that encourages the model to keep the coefficients β small. It is defined as:

$$\text{Loss} = \sum_{i=1}^{m} \left(y_i - \left(\beta_0 + \sum_{j=1}^{n} \beta_j x_{ij} \right) \right)^2 + \lambda \sum_{j=1}^{n} \beta_j^2$$

where m is the number of training samples, n is the number of features, y_i is the actual value of the dependent variable for sample i, x_{ij} is the value of feature j for sample i, and λ is the regularization parameter that controls the strength of the penalty.

Regularization

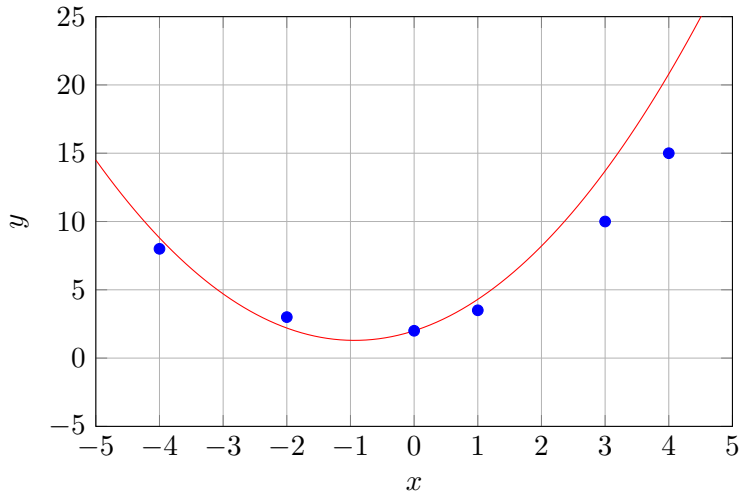
The penalty term in the Ridge Regression loss function helps to shrink the coefficients towards zero. The λ parameter determines the amount of regularization

applied to the model. A larger value of λ leads to greater regularization and smaller coefficients.

Example

Consider the following example of Ridge Regression:

$$y = 2 + 1.5x + 0.8x^2 + \epsilon$$



Lasso Regression

Lasso Regression is a regularized version of linear regression that introduces a penalty term to the loss function. It is used for feature selection and can set the coefficients of less important features to zero, effectively performing feature elimination.

Model Representation

The Lasso Regression model can be represented as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

where y is the dependent variable, $x_1, x_2, \ldots, x_n$ are the independent variables, $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients, and ϵ represents the error term.

Loss Function

The Lasso Regression loss function includes a penalty term that encourages the model to have sparse coefficients. It is defined as:

$$\text{Loss} = \sum_{i=1}^{m} \left(y_i - \left(\beta_0 + \sum_{j=1}^{n} \beta_j x_{ij} \right) \right)^2 + \lambda \sum_{j=1}^{n} |\beta_j|$$

where m is the number of training samples, n is the number of features, y_i is the actual value of the dependent variable for sample i, x_{ij} is the value of feature j for sample i, and λ is the regularization parameter that controls the strength of the penalty.

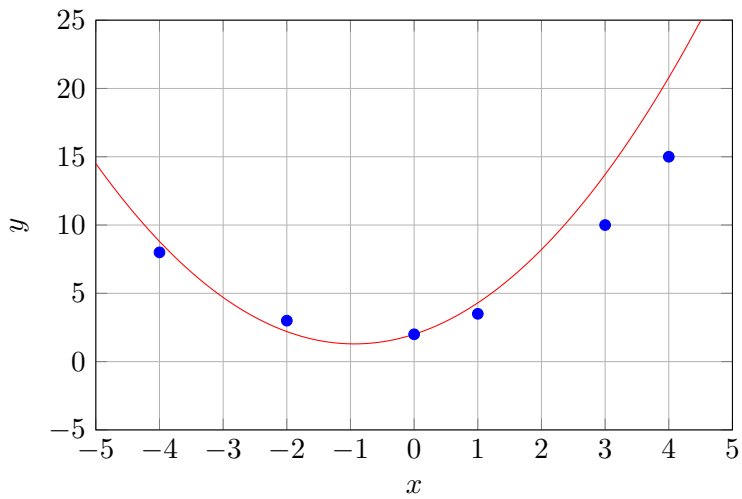
Regularization

The penalty term in the Lasso Regression loss function promotes sparsity in the coefficients. It encourages some coefficients to be exactly zero, effectively performing feature selection. The λ parameter controls the amount of regularization applied to the model.

Example

Consider the following example of Lasso Regression:

$$y = 2 + 1.5x + 0.8x^2 + \epsilon$$



In the plot above, the blue dots

Regression Trees

Regression Trees are a type of decision tree algorithm used for solving regression problems. They partition the feature space into regions and assign a constant value to each region, which represents the predicted value for samples falling within that region.

Model Representation

The Regression Tree model can be represented as a binary tree, where each internal node represents a splitting criterion on a specific feature, and each leaf node represents a predicted value. The prediction for a given sample is obtained by traversing the tree from the root to a leaf node based on the feature values.

Splitting Criterion

To create a Regression Tree, the algorithm searches for optimal splitting points in the feature space. The splitting criterion is typically based on minimizing the sum of squared errors within each region. This is often measured using metrics like the mean squared error (MSE) or the sum of squared differences (SSD).

Example

Consider the following example of a Regression Tree:

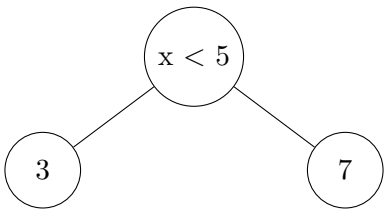


Figure 0.3. Example of a Regression Tree

In the example above, the tree has a single splitting criterion based on the feature x. If the value of x is less than 5, the predicted value is 3. Otherwise, the predicted value is 7.

Prediction

To make predictions using a Regression Tree, we start at the root node and follow the appropriate branches based on the feature values of the sample until we reach a leaf node. The predicted value for the sample is the constant value associated with that leaf node.

Random Forest Regressor

Ensemble learning is the process of using multiple models, trained over the same data, averaging the results of each model ultimately finding a more powerful predictive/classification result. Our hope, and the requirement, for ensemble learning is that the errors of each model (in this case decision tree) are independent and different from tree to tree. Bootstrapping is the process of randomly sampling subsets of a dataset over a given number of iterations and a given number of variables. These results are then averaged together to obtain a more powerful result. Bootstrapping is an example of an applied ensemble model. The bootstrapping Random Forest algorithm combines ensemble learning methods with the decision tree framework to create multiple randomly drawn decision trees from the data, averaging the results to output a new result that often leads to strong predictions/classifications.

Random Forest Regressor is an ensemble learning method that combines multiple decision trees to make predictions for regression problems. It reduces overfitting and improves the accuracy and robustness of the model by averaging the predictions of individual trees.

Model Representation

The Random Forest Regressor model can be represented as an ensemble of decision trees, where each tree independently predicts the target variable. The final prediction is obtained by averaging the predictions of all the trees.

Training Process

The Random Forest Regressor algorithm randomly selects subsets of the training data, known as bootstrap samples, to train each individual decision tree. Additionally,

at each node split, a random subset of features is considered for splitting. These two sources of randomness introduce diversity among the trees, reducing correlation and improving the overall performance.

Prediction

To make predictions using a Random Forest Regressor, each tree in the ensemble independently predicts the target variable based on the input features. The final prediction is obtained by averaging the predictions of all the trees.

Example

Consider the following example of a Random Forest Regressor with three decision trees:

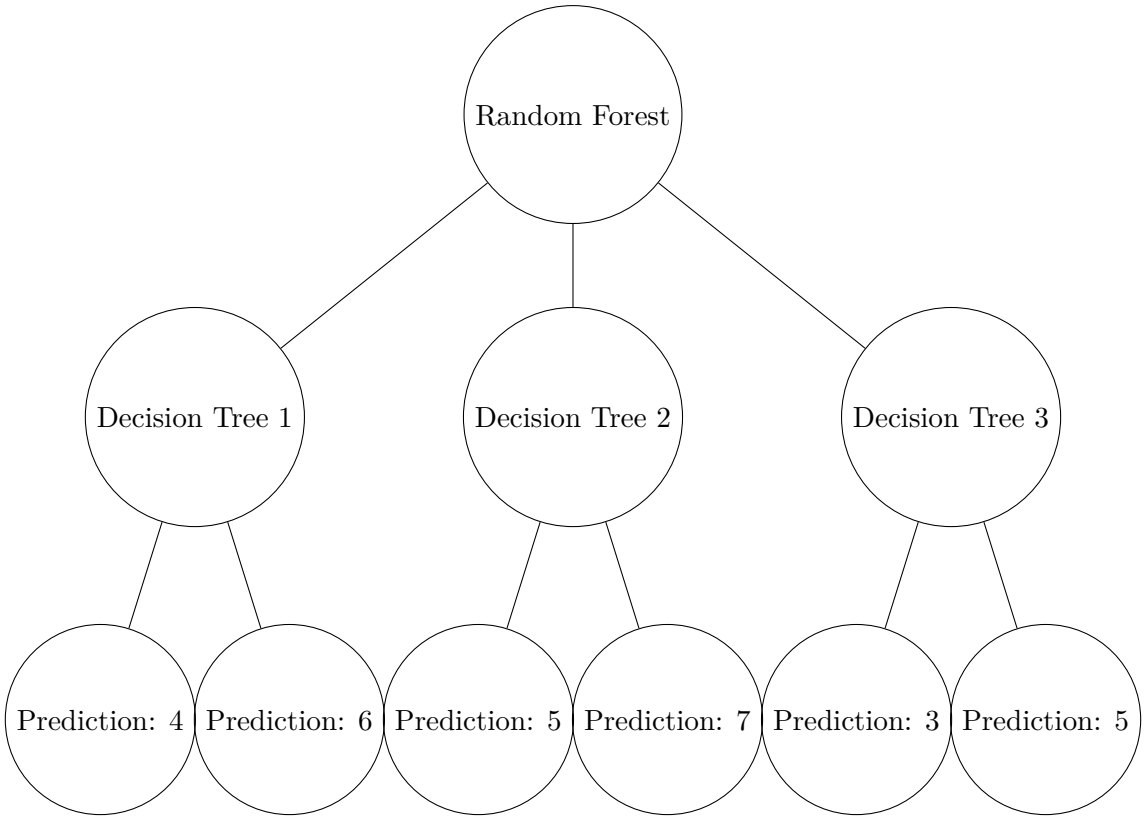


Figure 0.4. Example of a Random Forest Regressor

In the example above, the Random Forest Regressor consists of three decision trees. Each decision tree independently predicts a target variable for a given input,

and the final prediction is obtained by averaging the predictions of all the trees.

Gradient Boosting Regressor

Gradient Boosting Regressor is an ensemble learning method that combines multiple weak learners, typically decision trees, to make predictions for regression problems. It builds the model in a stage-wise manner by minimizing the loss function using gradient descent.

Model Representation

The Gradient Boosting Regressor model can be represented as a sum of weak learners:

$$\hat{y} = \sum_{i=1}^{N} \alpha_i h_i(x)$$

where $\hat{y}$ is the predicted value, N is the number of weak learners, α_i are the learning rates, and $h_i(x)$ are the weak learners, typically decision trees.

Training Process

The training process of Gradient Boosting Regressor involves adding weak learners to the model sequentially. Each weak learner is trained to minimize the residual error from the previous iteration. The learning rates α_i control the contribution of each weak learner to the final prediction.

Loss Function

The loss function used in Gradient Boosting Regressor is typically the mean squared error (MSE):

$$\text{Loss} = \frac{1}{2} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

where m is the number of training samples, y_i is the actual value of the target variable, and $\hat{y}_i$ is the predicted value.

Prediction

To make predictions using a Gradient Boosting Regressor, the weak learners are added sequentially, and each weak learner contributes to the final prediction based on its learning rate. The final prediction is the sum of the predictions from all the weak learners.

Example

Consider the following example of a Gradient Boosting Regressor with three decision trees:

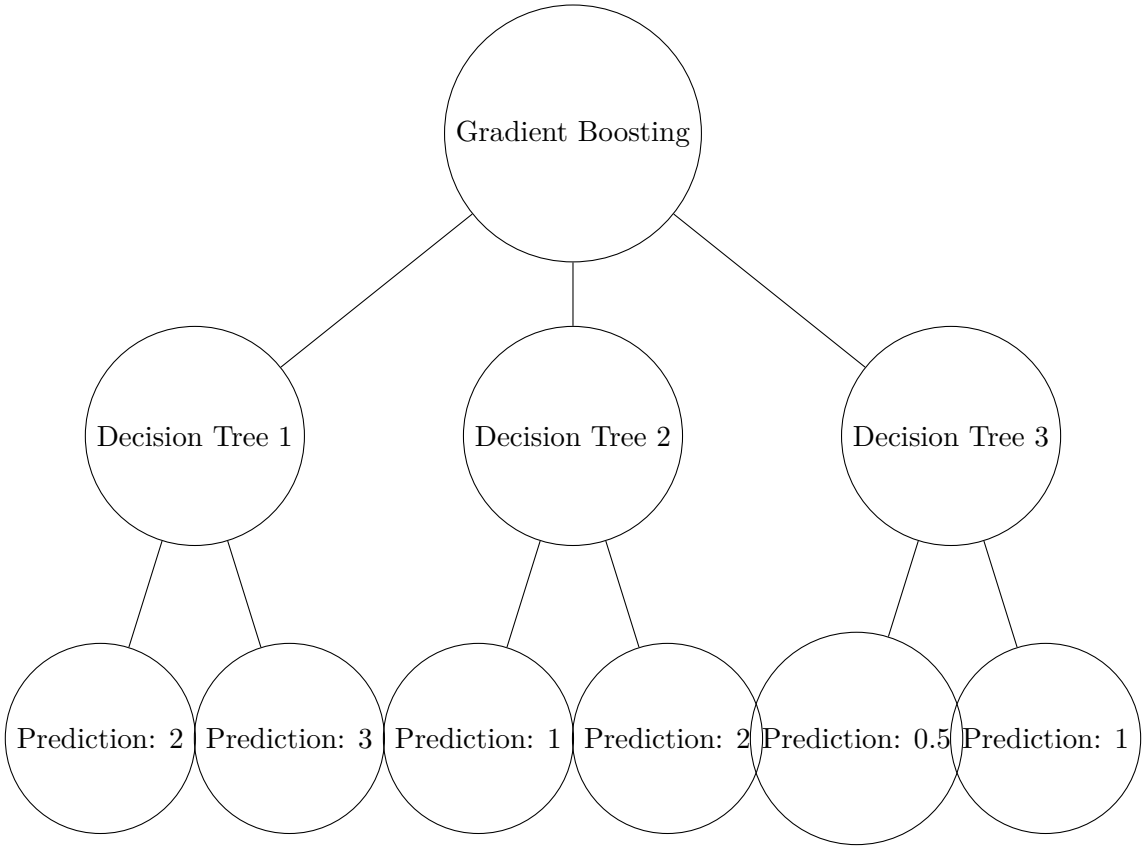


Figure 0.5. Example of a Gradient Boosting Regressor

In the example above, the Gradient Boosting Regressor consists of three decision trees. Each decision tree is trained to minimize the residual error from the previous iteration, and the final prediction is obtained by summing the predictions of all the decision trees.

SVM Regressor

Support Vector Machine (SVM) Regressor is a supervised learning algorithm used for regression tasks. It finds a hyperplane that best fits the training data, maximizing the margin between the hyperplane and the support vectors.

Model Representation

The SVM Regressor model can be represented as:

$$y = w^T x + b$$

where y is the predicted value, x is the input vector, w is the weight vector, and b is the bias term.

Loss Function

The objective of SVM Regressor is to find the hyperplane that minimizes the loss function, which is typically defined as:

$$\text{Loss} = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\Big(\max(0, |y_i - (w^T x_i + b)| - \epsilon)\Big)$$

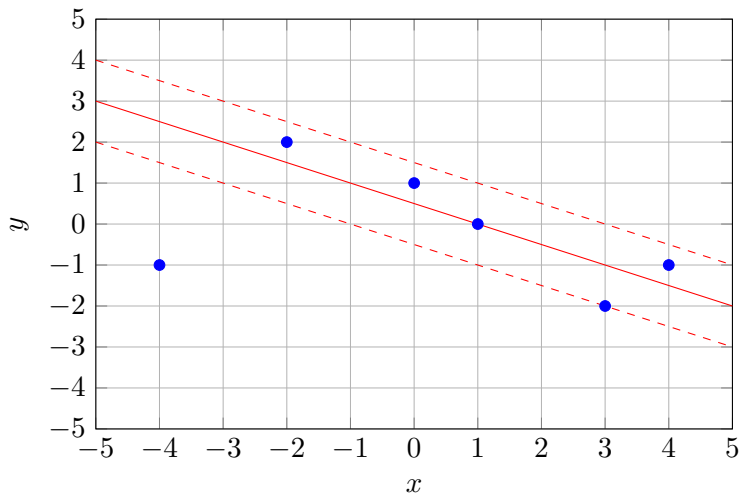
where $\|w\|^2$ is the squared norm of the weight vector, C is the regularization parameter that controls the trade-off between the margin and the training error, m is the number of training samples, y_i is the actual value of the target variable for sample i, x_i is the input vector for sample i, and ϵ is the parameter that determines the width of the insensitive zone around the hyperplane.

Margin and Support Vectors

In SVM Regressor, the margin is the distance between the hyperplane and the support vectors. The support vectors are the data points that lie on the margin or violate the margin. These points have the most influence on determining the hyperplane.

Example

Consider the following example of an SVM Regressor:



In the plot above, the blue dots represent the training samples, and the red line represents the SVM Regressor hyperplane. The dashed lines indicate the margin

Multinomial Logistic Regression

Multinomial Logistic Regression, also known as Softmax Regression, is a popular supervised learning algorithm used for multi-class classification problems. It extends the binary Logistic Regression to handle multiple classes by using the softmax function.

Model Representation

The Multinomial Logistic Regression model assigns probabilities to each class using the softmax function:

$$P(y = k|x) = \frac{e^{w_k^T x}}{\sum_{i=1}^{K} e^{w_i^T x}}$$

where $P(y = k|x)$ is the probability of the input vector x belonging to class k, w_k is the weight vector for class k, and K is the total number of classes.

Decision Boundary

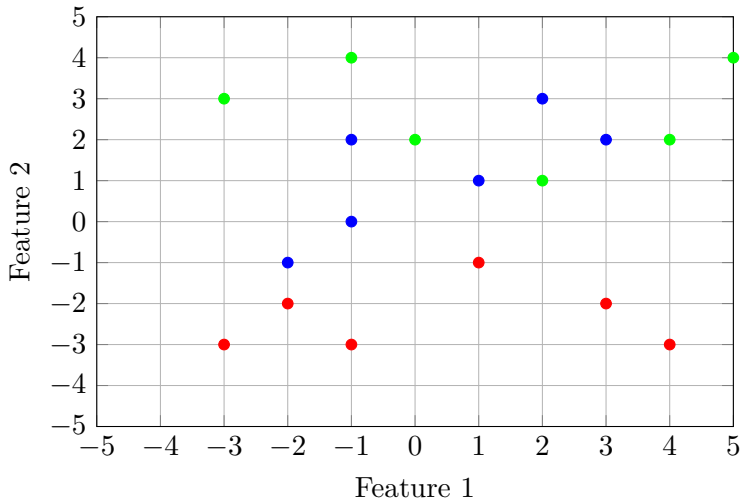
In Multinomial Logistic Regression, there is no direct decision boundary as in binary Logistic Regression. Instead, the class with the highest predicted probability is assigned as the predicted class.

Training Process

The training process of Multinomial Logistic Regression involves finding the optimal values of the weight vectors w_k. This is typically done by minimizing a loss function, such as the cross-entropy loss, using optimization algorithms like gradient descent.

Example

Consider the following example of Multinomial Logistic Regression with three classes:



In the plot above, the blue dots represent class 1 samples, the red dots represent class 2 samples, and the green dots represent class 3 samples. The decision boundaries for Multinomial Logistic Regression are not explicitly represented in the plot, but the algorithm would assign a class label based on the highest predicted probability.

Ridge Classifier for Multiple Classifications

Ridge Classifier is a supervised learning algorithm used for multi-class classification problems. It extends Ridge Regression to handle multiple classes by using the one-vs-all approach.

Model Representation

The Ridge Classifier model represents each class as a binary classification problem. For each class k, a binary classifier is trained to distinguish samples of class k from samples of all other classes. The final prediction is based on the classifier with the highest confidence score.

Binary Classifier

Each binary classifier in the Ridge Classifier is trained using Ridge Regression. The model for the binary classifier can be represented as:

$$y = \text{sign}(w_k^T x + b_k)$$

where y is the predicted class label, x is the input vector, w_k is the weight vector for class k, and b_k is the bias term for class k.

Regularization

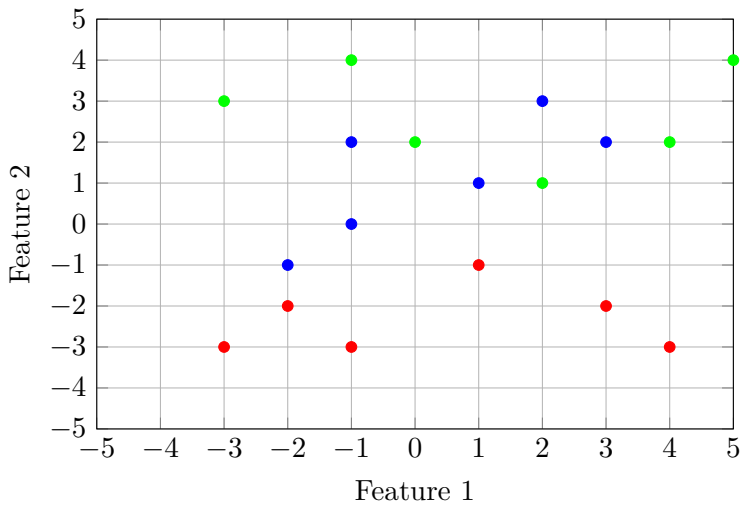
Ridge Classifier uses the L2 regularization term to control the complexity of the model and avoid overfitting. The regularization term is added to the loss function, which penalizes large values of the weight vectors:

$$\text{Loss} = \sum_{i=1}^{m} \Big(\max(0, 1 - y_i(w_{y_i}^T x_i + b_{y_i}))\Big) + \alpha \sum_{k=1}^{K} \|w_k\|^2$$

where m is the number of training samples, y_i is the actual class label for sample i, w_{y_i} is the weight vector for the true class label of sample i, b_{y_i} is the bias term for the true class label of sample i, K is the total number of classes, and α is the regularization parameter.

Example

Consider the following example of Ridge Classifier with three classes:



In the plot above, the blue dots represent class 1 samples, the red dots represent class 2 samples, and the green dots represent class 3 samples. The decision boundaries for the Ridge Classifier are not explicitly represented in the plot, but the algorithm would assign a class label based on the highest confidence score.

KNN

The k-Nearest Neighbors (k-NN) algorithm is a non-parametric classification algorithm that assigns a label to a test sample based on the labels of its k nearest neighbors in the training data. It works on the principle of similarity: if a majority of the k nearest neighbors belong to a certain class, the test sample is assigned to that class.

Algorithm Steps

The steps involved in the k-NN algorithm are as follows:

1. Compute the distance between the test sample and all training samples using a distance metric such as Euclidean distance or Manhattan distance.

2. Select the k nearest neighbors based on the smallest distances.

3. Assign the class label to the test sample based on the majority class among the k nearest neighbors.

Distance Metrics

The choice of distance metric plays a crucial role in the k-NN algorithm. The most commonly used distance metrics are:

- Euclidean distance:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

- Manhattan distance:

$$d(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} |a_i - b_i|$$

Here, $\mathbf{a}$ and $\mathbf{b}$ represent two feature vectors of length n, and a_i and b_i represent the i-th components of the feature vectors.

Choosing the Value of k

The choice of the value of k is an important consideration in the k-NN algorithm. A small value of k may lead to a noisy decision boundary, while a large value of k may result in oversmoothing and loss of local information. The value of k can be determined using cross-validation or other model selection techniques.

Prediction

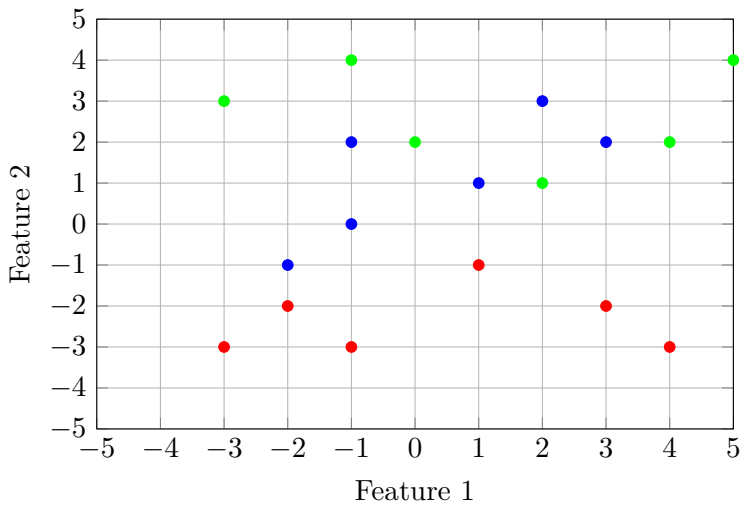
Once the k nearest neighbors are identified and their class labels are known, the prediction for the test sample can be made using different strategies:

- **Majority Voting**: Assign the class label that appears most frequently among the k nearest neighbors to the test sample.
- **Weighted Voting**: Assign the class label based on the weighted votes of the k nearest neighbors, where the weight is inversely proportional to the distance from the test sample.

The choice of the voting strategy depends on the specific problem and dataset.

Example

Consider the following example of KNN with two classes:



SVM Classifier

The Support Vector Machine (SVM) algorithm is a powerful supervised learning algorithm used for classification tasks. It finds an optimal hyperplane that separates the classes while maximizing the margin between the classes.

Model Representation

In the case of binary classification, the SVM algorithm aims to find the optimal hyperplane defined by:

$$w^T x + b = 0$$

where w is the weight vector perpendicular to the hyperplane, x is the input vector, and b is the bias term.

The predicted class label for a new input sample x is determined by:

$$\hat{y} = \text{sign}(w^T x + b)$$

where $\hat{y}$ is the predicted class label.

Margin and Support Vectors

The margin is defined as the distance between the hyperplane and the nearest training samples from both classes. The SVM algorithm aims to maximize this margin.

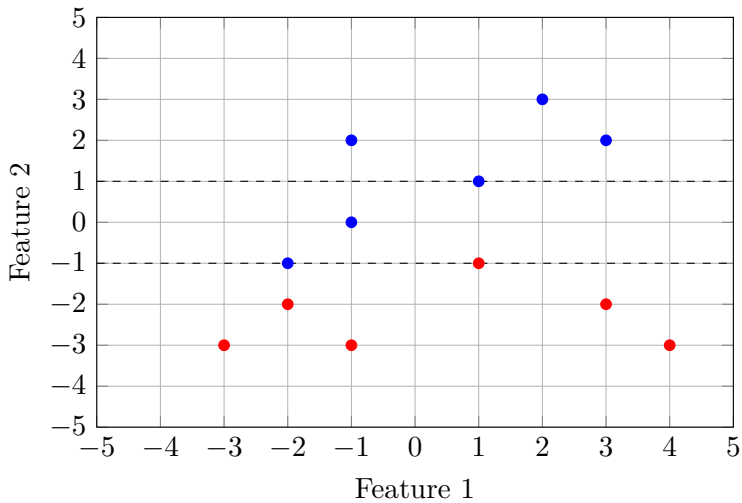
The support vectors are the training samples that lie on the margin or are misclassified. They are the critical points that determine the position of the hyperplane.

Kernel Trick

In cases where the classes are not linearly separable, the SVM algorithm can use the kernel trick. The kernel function maps the input samples into a higher-dimensional feature space where the classes can be linearly separable.

Example

Consider the following example of SVM with two classes:



In the plot above, the blue dots represent class 1 samples, and the red dots represent class 2 samples. The dashed lines represent the decision boundary (hyperplane) and the margins. The SVM algorithm finds the optimal hyperplane that separates the classes while maximizing the margin.

Decision Trees Classifier

The Decision Tree classifier is a popular machine learning algorithm used for both classification and regression tasks. It creates a tree-like model of decisions based on the features of the training data and uses this model to make predictions on new data.

Decision Tree Structure

A Decision Tree consists of internal nodes, branches, and leaf nodes. Internal nodes represent features or attributes, branches represent decision rules, and leaf nodes represent the class labels or regression values.

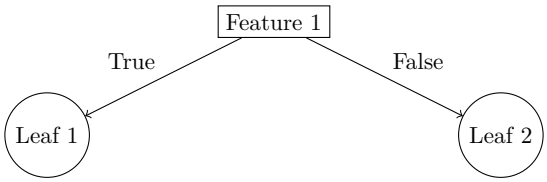


Figure 0.6. Example of a Decision Tree structure

Splitting Criterion

The Decision Tree algorithm uses a splitting criterion to determine the best feature and threshold for dividing the data at each internal node. Commonly used splitting criteria include Gini impurity, entropy, and information gain.

- **Gini impurity**: It measures the degree of impurity in a node and is defined as:

$$\text{Gini}(p) = 1 - \sum_{i=1}^{c} p_i^2$$

  where p_i is the probability of class i in the node, and c is the number of classes.

Random Forest

The Random Forest classifier is an ensemble learning algorithm that combines multiple decision trees to make predictions. It is a powerful and widely used machine learning algorithm known for its robustness and ability to handle complex datasets.

Ensemble of Decision Trees

A Random Forest consists of an ensemble of decision trees, where each tree is trained on a different subset of the training data and uses a random selection of features. The predictions of individual trees are then combined to make the final prediction.

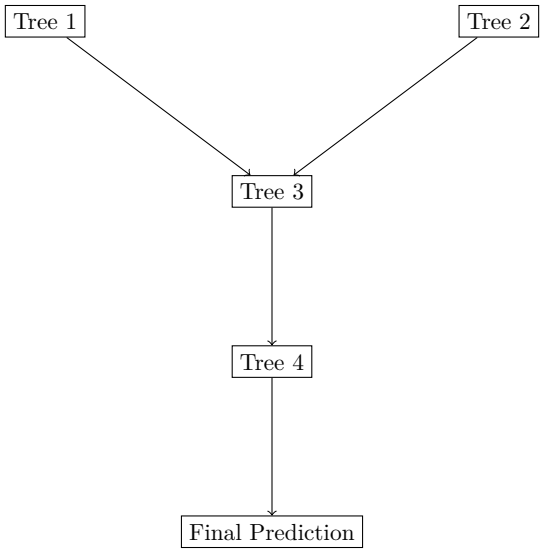


Figure 0.7. Example of a Random Forest with four decision trees

Random Feature Selection

To introduce randomness and reduce overfitting, each decision tree in the Random Forest is trained on a random subset of features. This means that at each node of a tree, only a subset of features is considered for the split. The number of features selected is typically the square root of the total number of features.

Bagging

The Random Forest also uses a technique called bagging (bootstrap aggregating) to create different subsets of the training data for each tree. Bagging involves sampling the training data with replacement, creating multiple subsets of the same size as the original dataset. Each decision tree is then trained on one of these subsets.

Prediction

To make a prediction using the Random Forest, each decision tree in the ensemble independently predicts the class label or regression value for the input data point. The final prediction is then determined by majority voting for classification or averaging for regression. For example, in classification, the class label with the highest number of votes is assigned as the predicted label.

Gradient Boosting Classifier

The Gradient Boosting classifier is an ensemble learning algorithm that combines multiple weak learners (usually decision trees) to create a strong predictive model. It sequentially builds a series of models, each one correcting the mistakes made by the previous models.

Boosting Process

The Gradient Boosting algorithm builds the model in an iterative manner. At each iteration, it fits a weak learner to the negative gradient of the loss function associated with the current ensemble. The new model is then added to the ensemble, adjusting the weights of the previously fitted models.

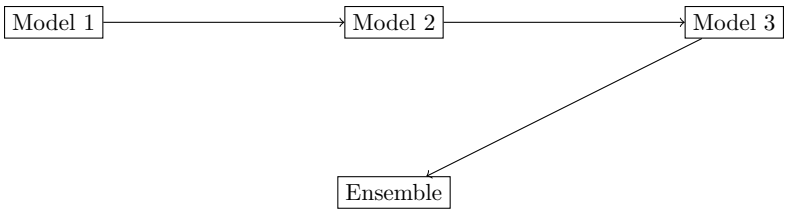


Figure 0.8. Example of a Gradient Boosting ensemble with three models

Loss Function

The loss function used in Gradient Boosting depends on the problem being solved. For classification problems, commonly used loss functions include the logistic loss and the exponential loss. For regression problems, the mean squared error (MSE) is often used.

Model Fitting

At each iteration, the weak learner is fitted to the negative gradient of the loss function. This is done by minimizing the loss function with respect to the predictions of the current ensemble. The weak learner is typically a shallow decision tree, often referred to as a "stump".

Weighted Combination

The predictions of the weak learners are combined to form the final prediction of the Gradient Boosting classifier. Each model is assigned a weight that determines its contribution to the ensemble. The weights are usually determined through gradient descent optimization.

Prediction

To make a prediction using the trained Gradient Boosting classifier, the predictions of all the weak learners in the ensemble are combined according to their weights. For classification, a probability score or class label is generated, while for regression, a continuous value is predicted.

0.4 Control theory techniques

PCA-based Algorithm with Kalman Filter

The PCA-based algorithm with Kalman filter is a method used for classification tasks that combines Principal Component Analysis (PCA) with the Kalman filter.

PCA is a dimensionality reduction technique that transforms high-dimensional data into a lower-dimensional space while retaining important information. It aims to find a set of orthogonal variables called principal components, which capture the maximum variance in the data. By discarding less significant principal components, PCA effectively reduces the dimensionality of the data.

The Kalman filter is an algorithm employed for state estimation in dynamic systems. It proves useful in the presence of uncertainty or noise in measurements. This filter recursively estimates the state of a system by combining measurements with predictions based on a mathematical model of the system.

Algorithm

Training Phase:

- Perform PCA on the training dataset to reduce its dimensionality.
- Calculate the principal components and select the most significant ones.
- Apply the Kalman filter to the transformed data.
- Model the dynamics of the data and estimate its state using the Kalman filter.
- Associate the training samples with their corresponding class labels.

Classification Phase:

- Apply the same PCA transformation used during training to reduce the dimensionality of the test data.
- Use the trained Kalman filter to estimate the state of the transformed test data.
- Classify the test data based on its estimated state using a suitable classification algorithm, such as Random Forest Classifier.

Entropy-based Feature Selection

Also we can use entropy-based feature selection, which is a technique used to identify the most informative features in a dataset by measuring their relevance to the target variable. The concept of entropy, borrowed from information theory, quantifies the uncertainty or randomness in a random variable.

Information gain is a commonly used measure in entropy-based feature selection. It measures the reduction in entropy of the target variable when a particular feature is known. The higher the information gain, the more relevant the feature is to the target variable. Entropy is a measure of the impurity or randomness in a set of data. In the context of feature selection, it represents the uncertainty in the target variable. A high entropy value indicates a greater diversity of target variable values, while a low entropy value indicates more homogeneity.

Entropy-based Feature Selection Process

The process of entropy-based feature selection typically involves the following steps:

1. Calculate the entropy of the target variable using its distribution in the dataset.
2. For each feature in the dataset, calculate the information gain by considering its distribution and the target variable's distribution.
3. Select the features with the highest information gain as the most informative features.
4. Use the selected features for further analysis or modeling.

RT - based algorithm

The RT-based algorithm is an approach for classification tasks that combines decision trees and regression (we used SVM Regression technique). The algorithm follows the following steps:

1. Tree Construction:
    - Initially, a decision tree is grown using the training dataset.
    - The tree construction process involves recursively splitting the data based on the feature values that optimize a certain criterion, such as information gain or Gini index.
2. Leaf Dataset Partition:
    - Once the tree is constructed, each leaf node contains a subset of the training dataset that falls within its region.
    - This partitioning allows us to work with localized datasets for subsequent analysis.
3. SVM Regression:
    - For each leaf node, an SVM regressor is trained using the dataset partition within that leaf.
    - The SVM regressor aims to find an optimal hyperplane that best separates the classes and maximizes the margin.
    - It utilizes a kernel function to map the data into a higher-dimensional space if necessary.
4. Classification:
    - To classify new instances, they traverse down the tree until reaching a leaf node.
    - Then, the corresponding SVM regressor for that leaf is used to predict the class label.

- The decision is typically made by thresholding the SVM prediction at a certain confidence level.

The RT-based algorithm leverages decision trees to partition the data and then applies SVM regression on the dataset partitions at the leaf nodes.

0.5 Experimental validation

Data Description

As described before, our data consist of 1451 samples. There are no duplicates and NaN values. The data is distributed as follows:

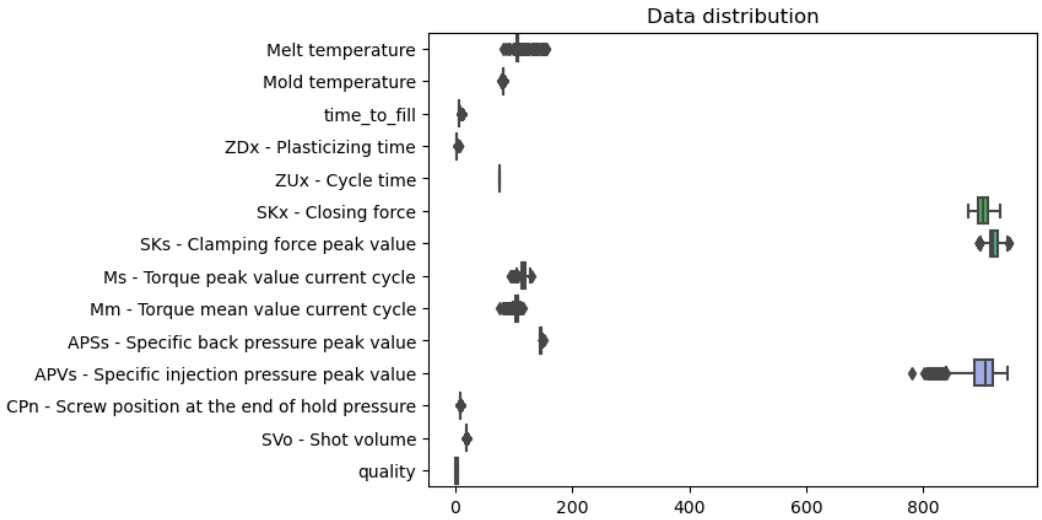


Figure 0.9. Data Distribution

The dataset under consideration, as referenced in Figure 0.9, exhibits variations in the dimensions of each feature. In order to ensure compatibility with machine learning techniques, it is necessary to normalize the data. After applying normalization and standardization processes, the dataset was transformed to meet the following specifications:

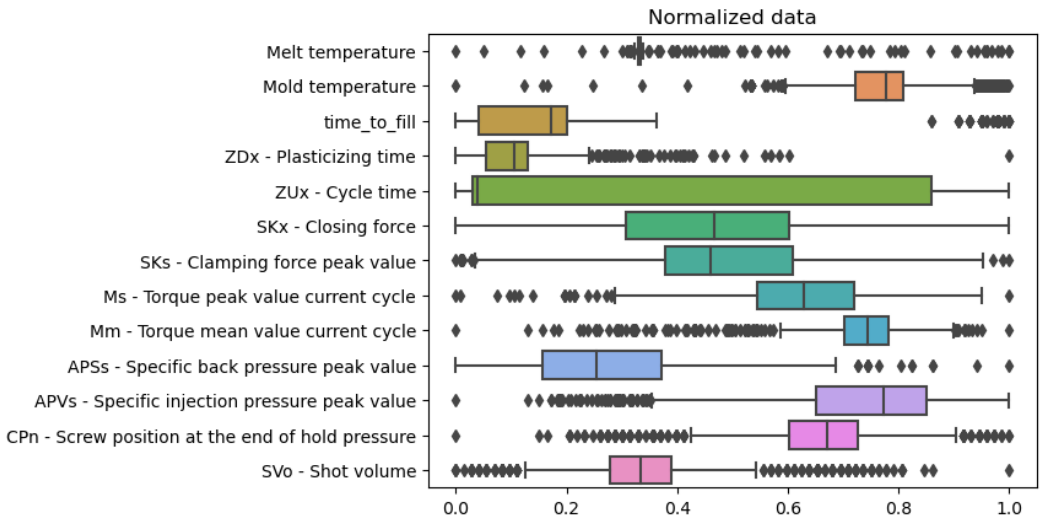


Figure 0.10. Normalized Data Distribution

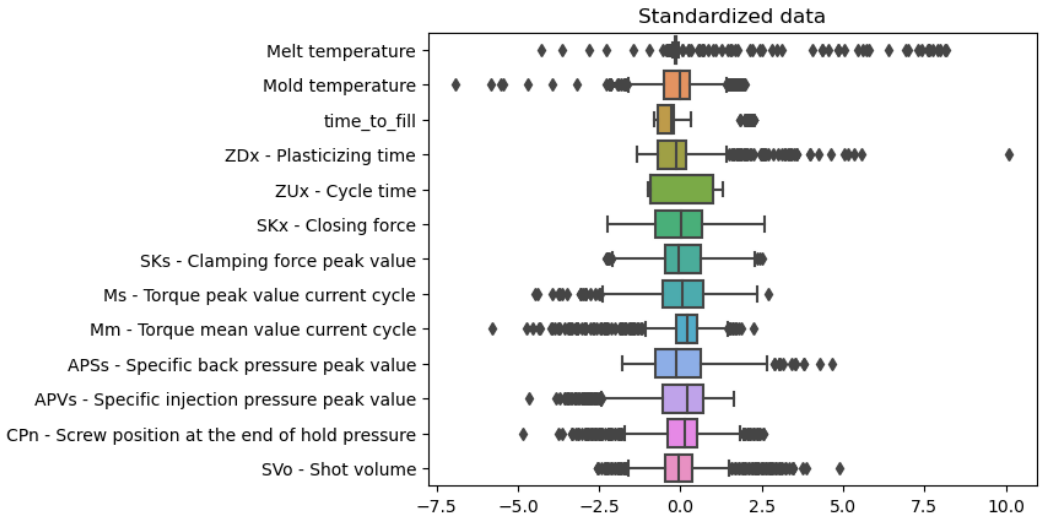
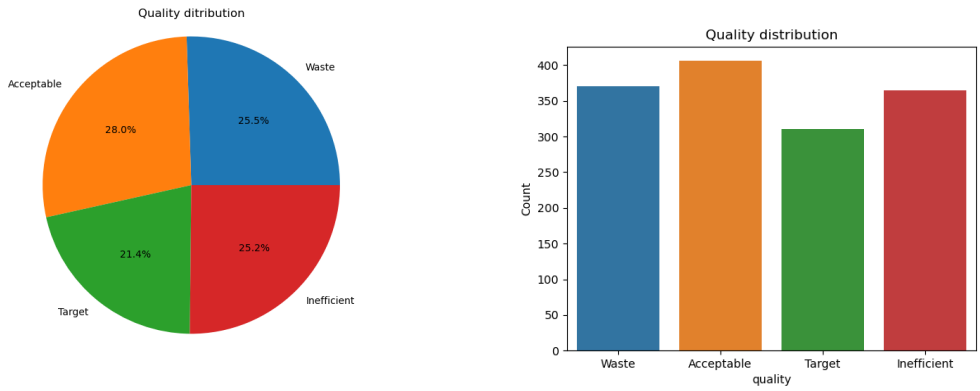


Figure 0.11. Standardized Data Distribution

Based on the analysis of the graphics, it can be observed that our dataset exhibits a balanced distribution. The figures provide visual representations that indicate an approximately equal distribution of samples across different quality classes.



(a) Pie Chart **(b)** Bar Plot

Figure 0.12. Quality Variable Distribution

To explore the correlations between variables, we can construct trajectories or paths that depict the relationships and patterns between the variables. By analyzing these trajectories, we can gain insights into the dependencies, associations, and trends among the variables under investigation:

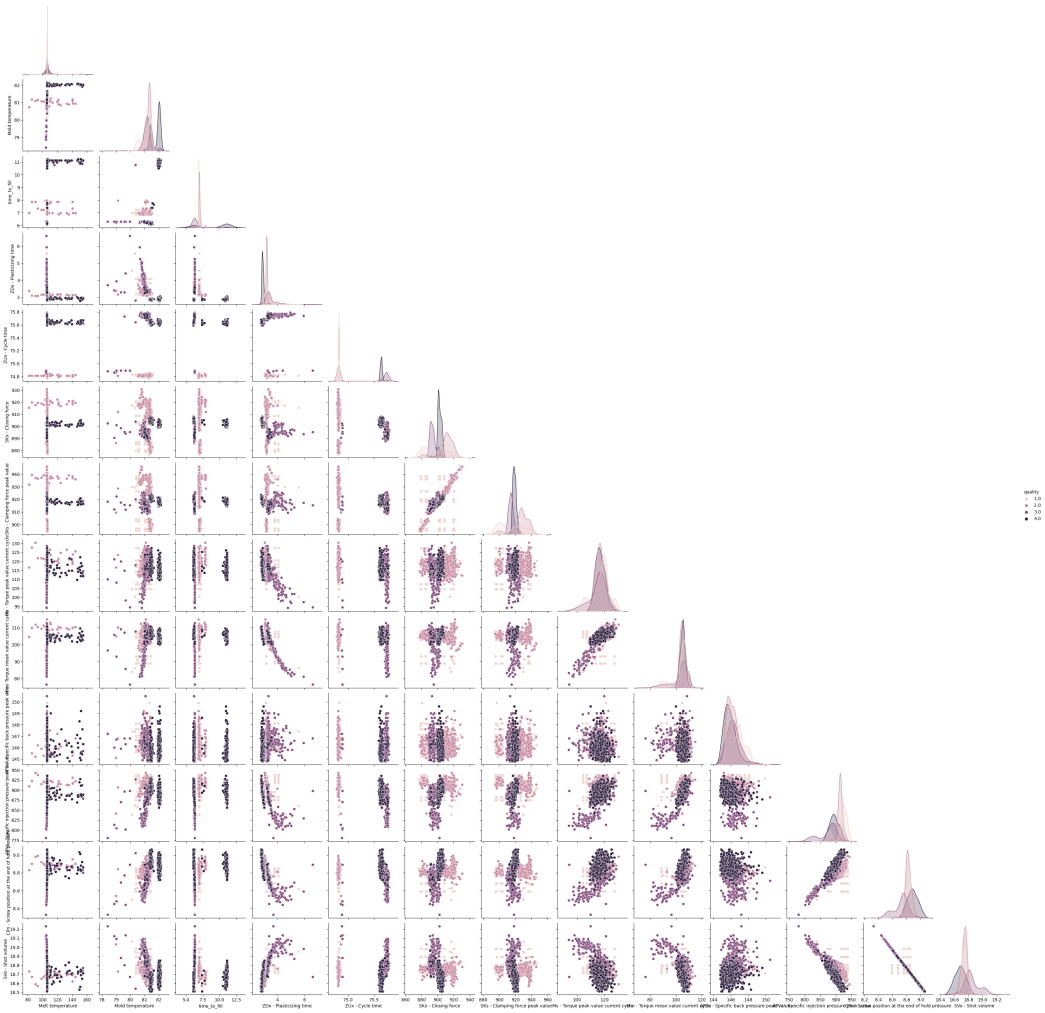


Figure 0.13. Trajectories

Based on the analysis of the trajectories, the following pairs demonstrate linear relationships:

- Mold temperature and time to fill
- SKs - Clamping force peak value and SKx - Closing force
- Ms - Torque peak value current cycle and Mm - Torque mean value current cycle
- APVs - Specific injection pressure peak value and CPn - Screw position at the end of hold pressure

- SVo - Shot volume and APVs - Specific injection pressure peak value
- SVo - Shot volume and CPn - Screw position at the end of hold pressure

In order to determine the strength of the linear correlation between the pairs of variables mentioned, Pearson's correlation coefficient can be computed.

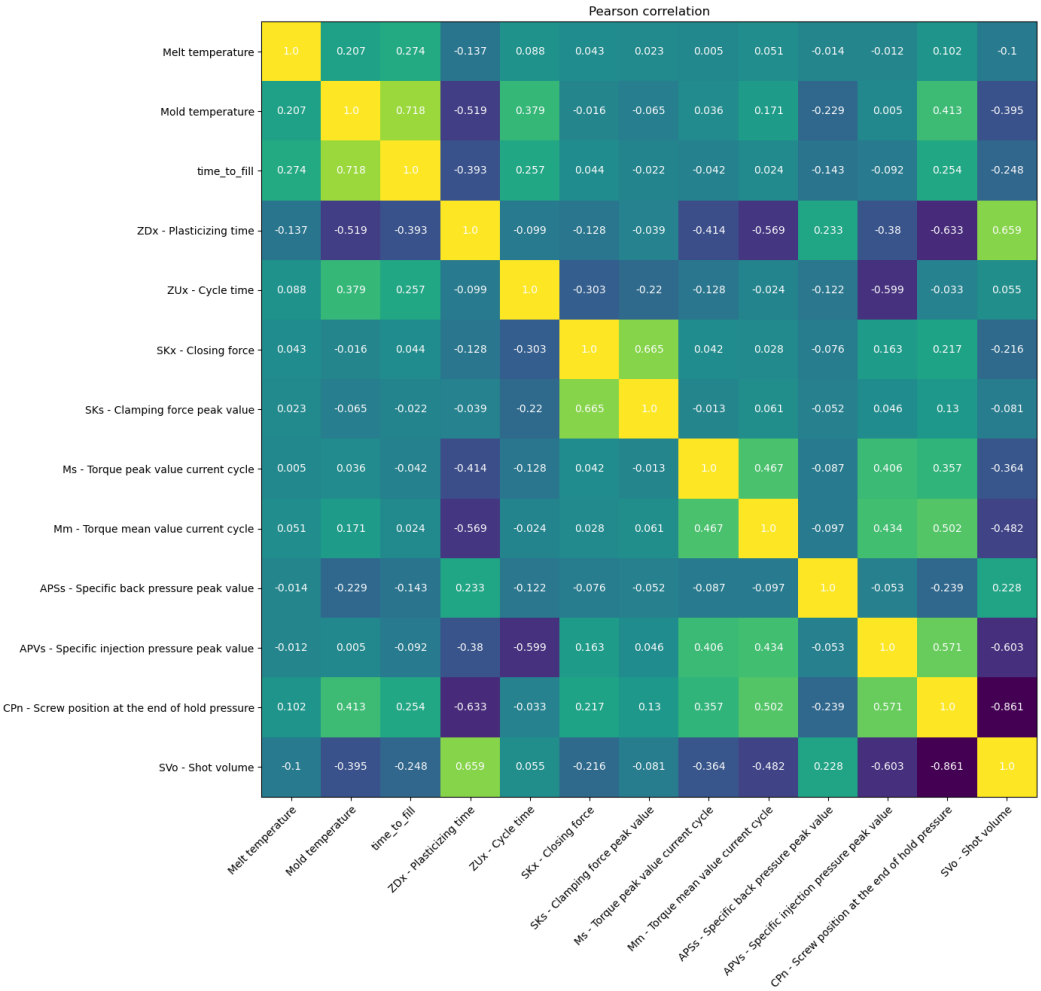


Figure 0.14. Pearson's correlation coefficients

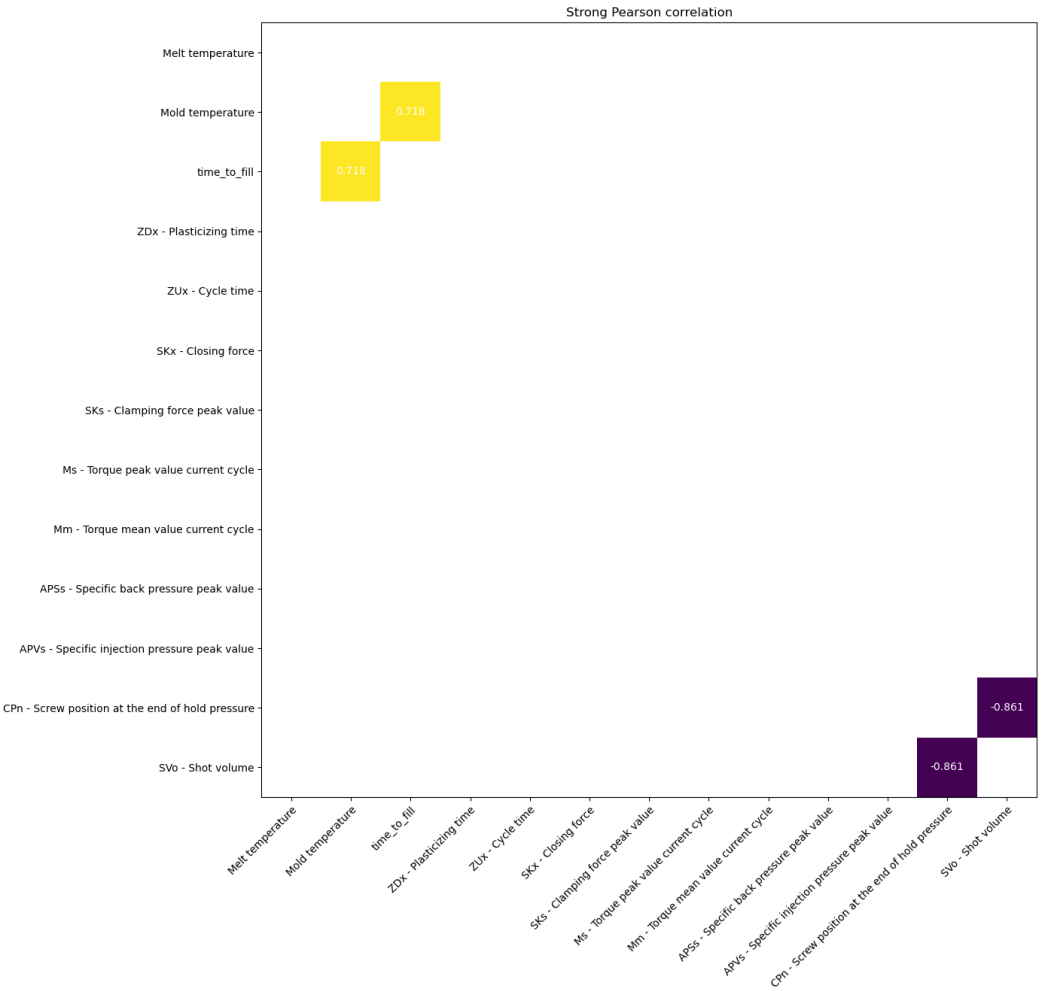


Figure 0.15. Strong Pearson's correlation coefficients

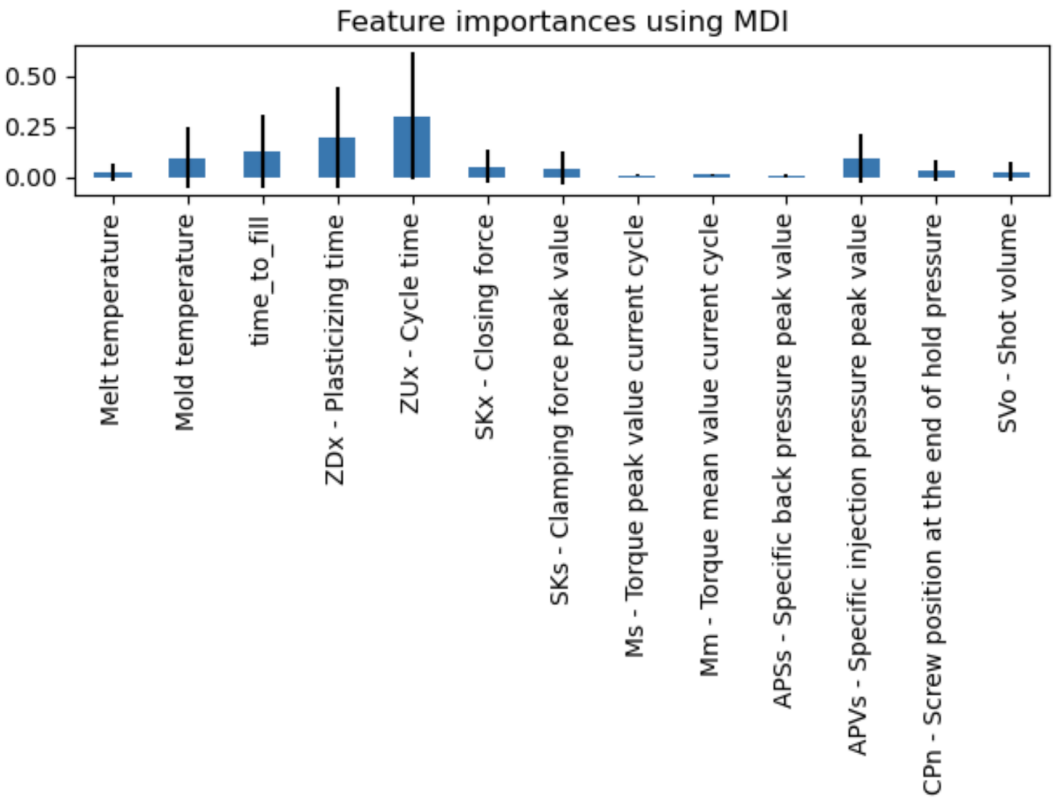
We can conclude that the pairs: *Mold Temperature and time to fill* and *CPn - Screw position at the end of hold pressure and SVo - Shot Volume* exhibit a strong linear correlation.

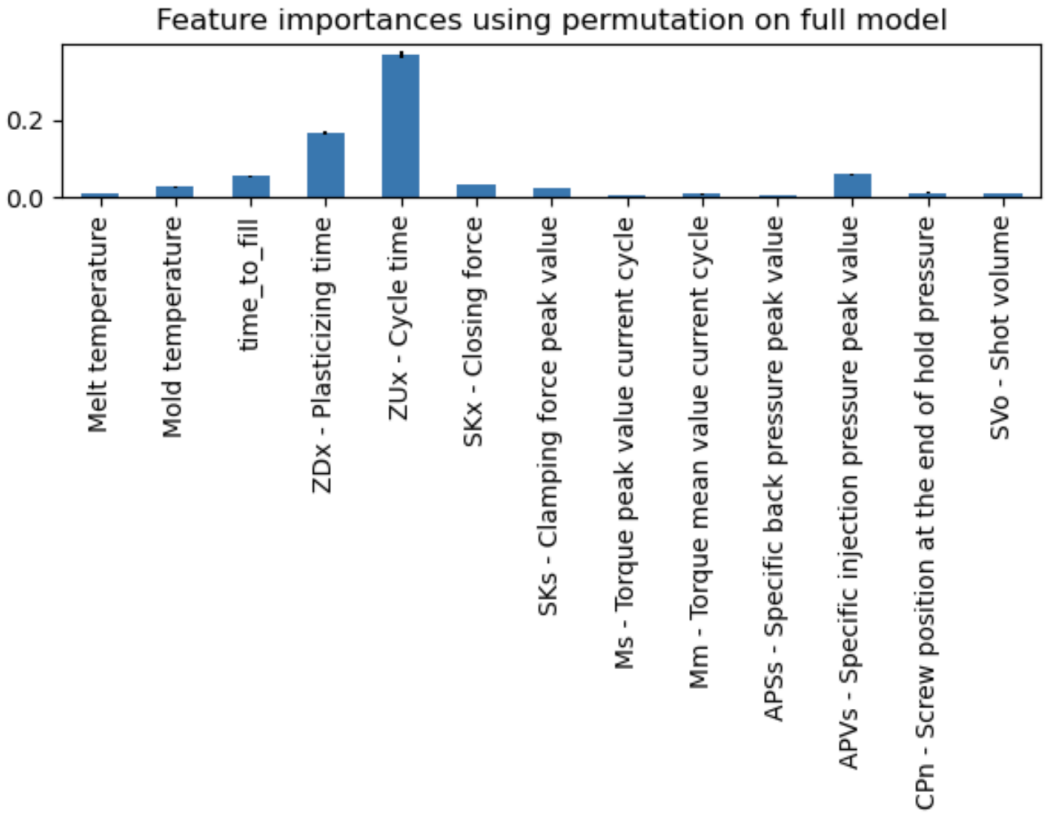
The strong linear correlation between *Mold Temperature* and *time to fill* indicates that there is a consistent relationship between the temperature of the mold and the time required to fill it during the injection molding process. As the mold temperature increases, the time to fill the mold also tends to increase, or vice versa.

Similarly, the strong linear correlation between *CPn - Screw position at the end of hold pressure* and *SVo - Shot Volume* suggests a consistent relationship between

the screw position at the end of the hold pressure phase and the shot volume. As the screw position increases or decreases, the shot volume also exhibits a corresponding increase or decrease.

Using the Random Forest algorithm, we conducted a comprehensive analysis to assess the importance of variables in the prediction process. The results revealed that the variables *Ms - Torque peak value current cycle* and *APSs - Specific back pressure peak value* exhibit relatively low importance when considering tree-based algorithms.





Based on the findings of our analysis, we can consider creating additional datasets that exclude certain variables based on their characteristics. Specifically, we can generate the following datasets:

- Dataset without strong linear correlations.
- Dataset without not important variables for tree-based algorithms.
- Dataset without both strong linear correlations and not important variables.

ML techniques

In this section, we present and discuss the results obtained from applying the techniques mentioned earlier to the datasets. The results provide insights into the performance and effectiveness of these techniques in addressing the problem at hand.

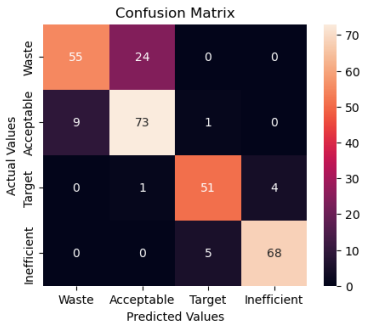
In the context of regression techniques applied to a classification problem, it is common for the output to be in the form of continuous float numbers rather than discrete integers. However, in order to associate these predicted values with specific qualification classes such as "Waste," "Acceptable," "Target," and "Inefficient," a mapping process is required to approximate the predicted values to the nearest integer representing the respective class.

This mapping allows for the interpretation of the regression predictions within the framework of the predefined qualification classes. By considering the closest integer value, the predictions can be effectively assigned to the appropriate qualification class. This approximation facilitates the alignment of the continuous output of the regression technique with the discrete classification labels used to identify the qualification classes.

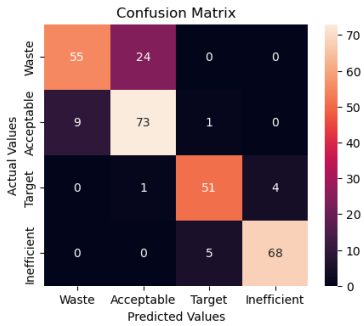
Linear Regression

Name of data	Accuracy
Normalized Data without not important features	86.94%
Standardized Data without not important features	86.94%
Fully Clean Standardized data	85.57%
Fully Clean Normalized data	85.57%
Standardized data	84.88%
Normalized data	84.88%
Normalized Data without strong correlations	83.85%
Standardized Data without strong correlations	83.85%

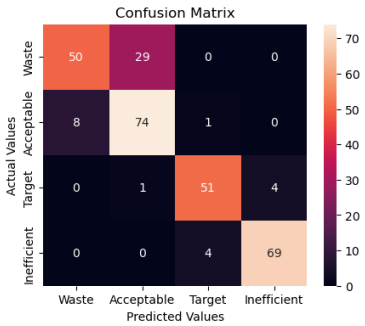
The analysis of the results obtained from the linear regression technique reveals that the best performance, with an accuracy of 86.94%, is achieved when using either the standardized or normalized dataset without the inclusion of not important features.



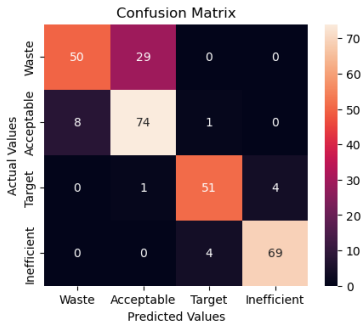
(a) Standardized data



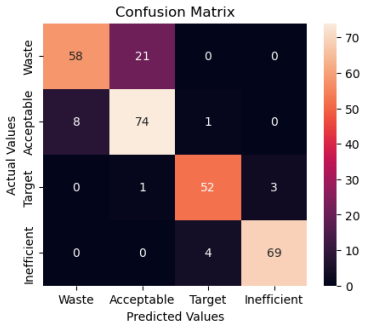
(b) Normalized data



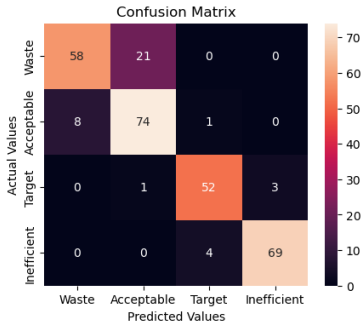
(c) Standardized data without strong correlations



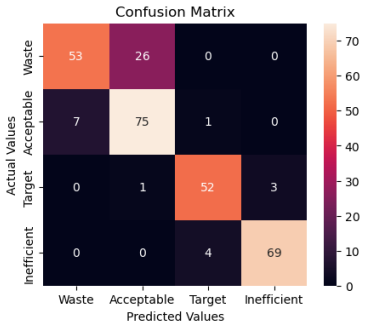
(d) Normalized data without strong correlations



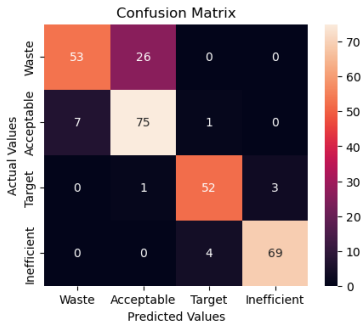
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.16. Linear Regression Confusion Matrices

Polynomial Regression

Name of data	Accuracy
Normalized Data without strong correlations	89.69%
Standardized Data without strong correlations	89.69%
Standardized data	89.35%
Normalized data	89.35%
Normalized Data without not important features	89.00%
Standardized Data without not important features	89.00%
Fully Clean Standardized data	87.63%
Fully Clean Normalized data	87.63%

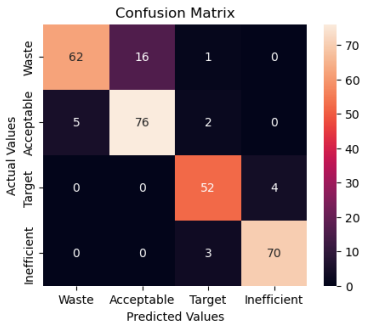
(a) 2 degree

Name of data	Accuracy
Fully Clean Normalized data	89.69%
Fully Clean Standardized data	89.69%
Normalized Data without not important features	88.32%
Standardized Data without not important features	88.32%
Normalized Data without strong correlations	84.19%
Standardized Data without strong correlations	84.19%
Standardized data	81.44%
Normalized data	81.44%

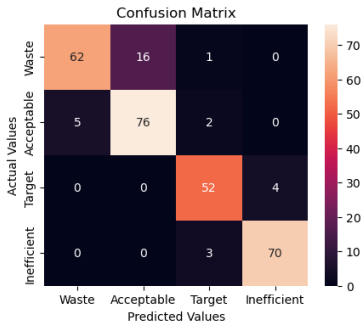
(b) 3 degree

The comparative analysis of regression models reveals interesting findings regarding the dataset characteristics and model performance. Specifically, when examining two different degrees of polynomial regression (2nd and 3rd degree) in relation to a linear regression model, it is observed that both polynomial regression models outperform the linear regression model in terms of accuracy, with a best-case accuracy of 89.69%.

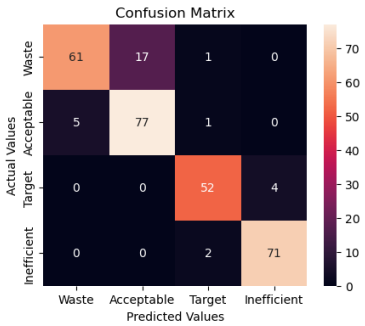
Upon closer examination, it is evident that the 2nd degree polynomial regression model demonstrates superior performance on datasets without strong correlations between the variables. In contrast, the 3rd degree polynomial regression model performs best on dataset that not only doesnt have strong correlations but also not important features.



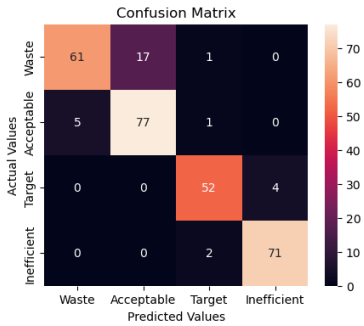
(a) Standardized data



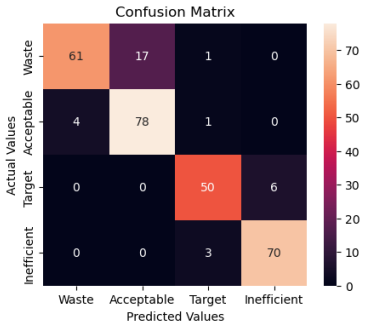
(b) Normalized data



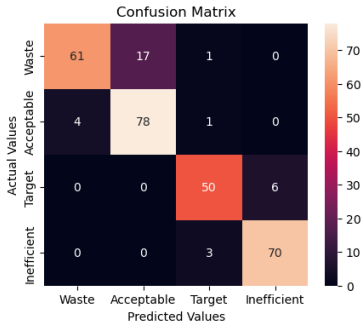
(c) Standardized data without strong correlations



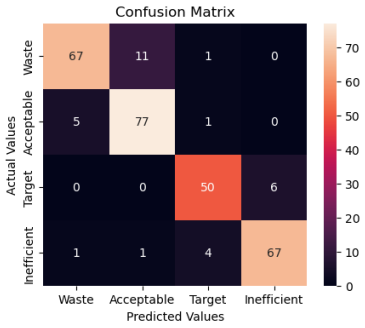
(d) Normalized data without strong correlations



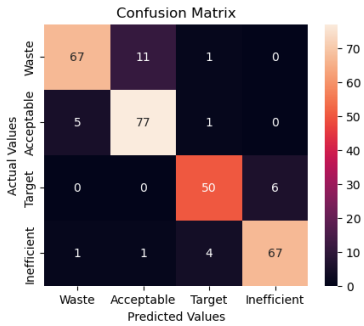
(e) Standardized data without not important features



(f) Normalized data without not important features

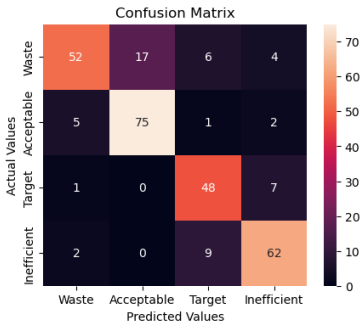


(g) Fully clean standardized data

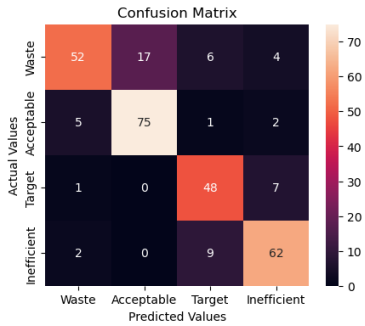


(h) Fully clean normalized data

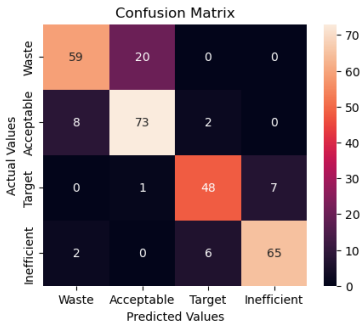
Figure 0.18. Polynomial Regression Degree 2 Confusion Matrices



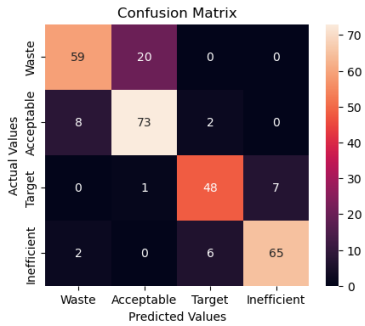
(a) Standardized data



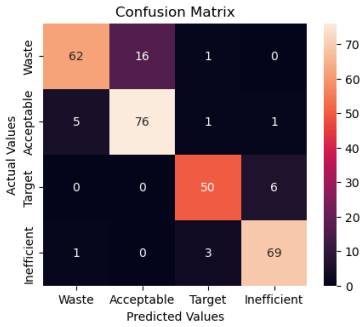
(b) Normalized data



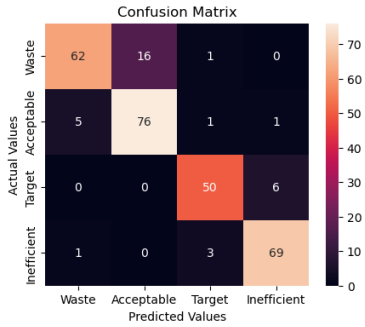
(c) Standardized data without strong correlations



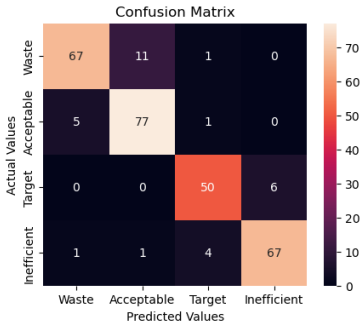
(d) Normalized data without strong correlations



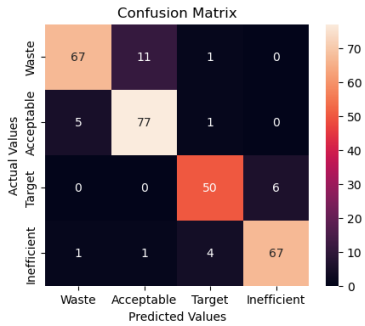
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.19. Polynomial Regression Degree 3 Confusion Matrices

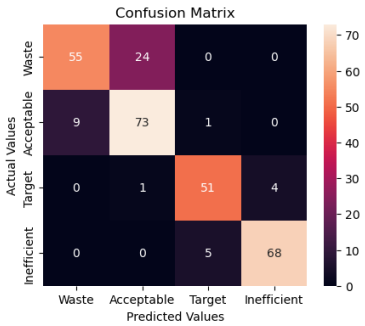
Ridge Regression

Name of data	Accuracy
Standardized Data without not important features	86.94%
Normalized Data without not important features	85.91%
Fully Clean Standardized data	85.57%
Normalized data	85.22%
Standardized data	84.88%
Fully Clean Normalized data	84.54%
Normalized Data without strong correlations	83.85%
Standardized Data without strong correlations	83.85%

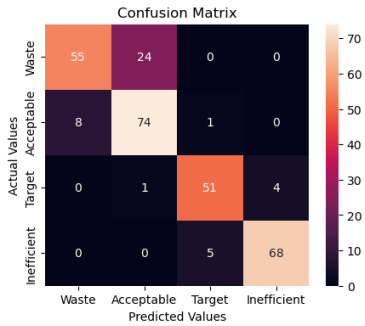
Ridge Regression technique achieves the highest performance among the evaluated models, attaining an accuracy of 86.94%. This performance is exactly same as the Linear Regression model but falls short of the accuracy achieved by the Polynomial Regression model.

Specifically, the best result of 86.94% is obtained when applying Ridge Regression to the standardized dataset without including not important features. It is worth noting that this accuracy is identical to the best accuracy achieved by Linear Regression.

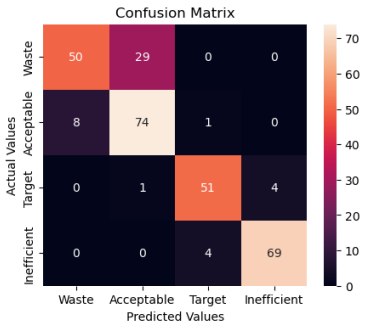
However, in terms of predictive performance, the Polynomial Regression model outperforms both Ridge Regression and Linear Regression, as it achieves higher accuracy. This indicates that the polynomial regression models, with their ability to capture nonlinear relationships, provide improved predictive capabilities compared to linear-based approaches.



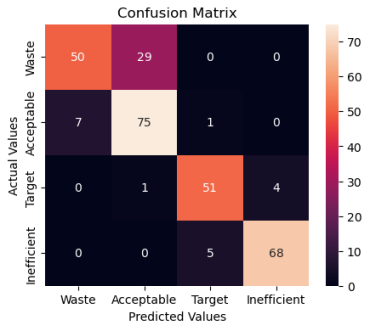
(a) Standardized data



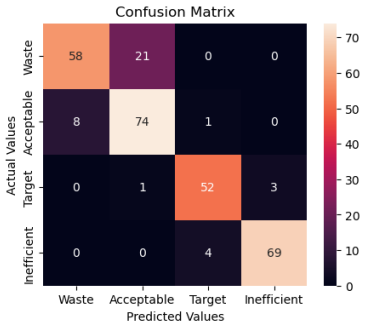
(b) Normalized data



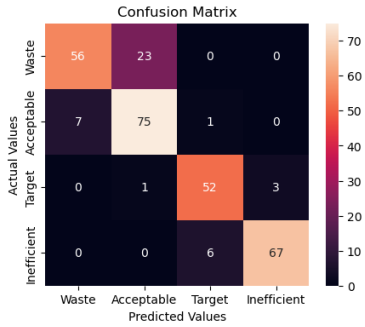
(c) Standardized data without strong correlations



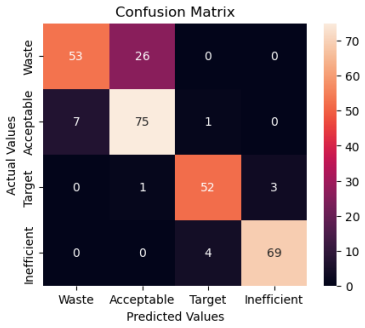
(d) Normalized data without strong correlations



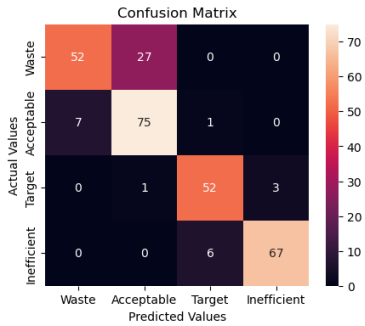
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.20. Ridge Regression Confusion Matrices

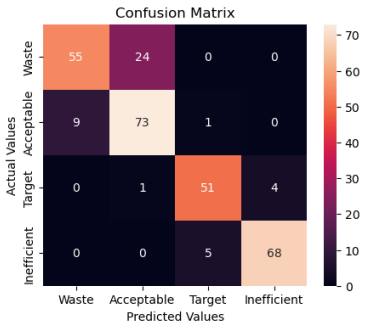
Lasso Regression

Name of data	Accuracy
Normalized Data without not important features	86.94%
Standardized Data without not important features	86.94%
Fully Clean Standardized data	85.57%
Fully Clean Normalized data	85.57%
Standardized data	84.88%
Normalized data	84.88%
Normalized Data without strong correlations	83.85%
Standardized Data without strong correlations	83.85%

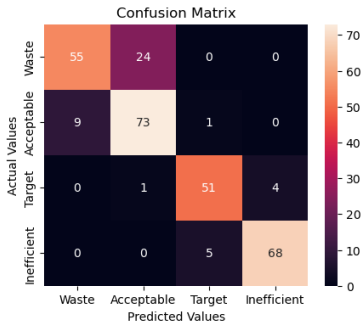
Lasso Regression technique achieves a similar performance to the Linear Regression model and Ridge Regression model, all of them attaining an accuracy of 86.94%. However, it falls short of the accuracy achieved by the Polynomial Regression model.

Specifically, the best result of 86.94% is obtained when applying Lasso Regression to the standardized or normalized data without including not important features. It is worth noting that this accuracy is identical to the best accuracy achieved by Linear Regression and Ridge Regression.

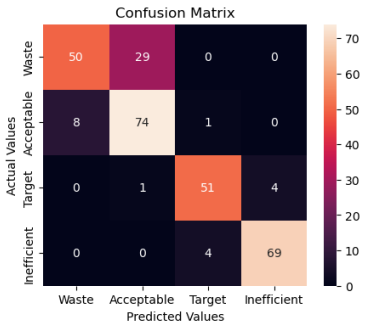
While Lasso Regression performs similarly to Linear Regression in terms of accuracy, the Polynomial Regression model outperforms both Lasso Regression and Linear Regression. This indicates that the polynomial regression models, with their ability to capture nonlinear relationships, provide improved predictive capabilities compared to linear-based approaches.



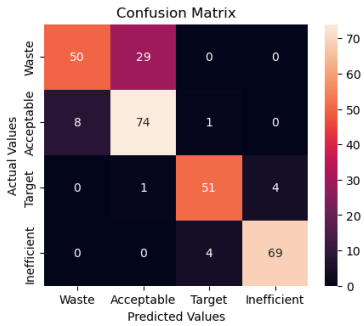
(a) Standardized data



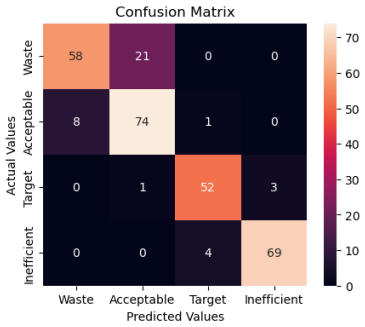
(b) Normalized data



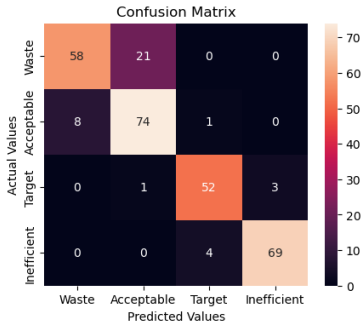
(c) Standardized data without strong correlations



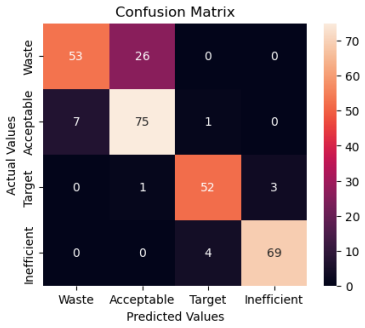
(d) Normalized data without strong correlations



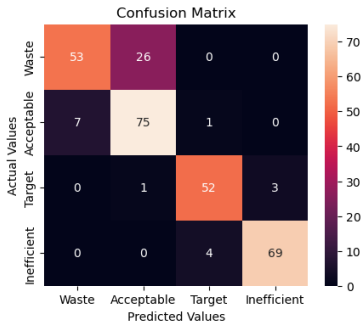
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.21. Lasso Regression Confusion Matrices

Decision Tree Regression

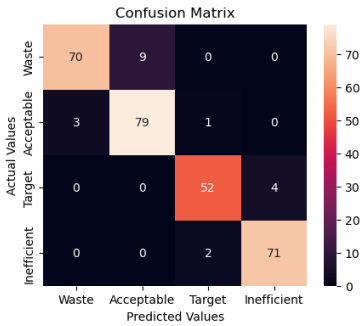
Name of data	Accuracy
Standardized data	93.81%
Fully Clean Standardized data	93.47%
Standardized Data without strong correlations	93.47%
Fully Clean Normalized data	93.13%
Normalized Data without strong correlations	93.13%
Normalized data	92.78%
Normalized Data without not important features	92.44%
Standardized Data without not important features	92.44%

The Decision Tree Regression algorithm consistently outperforms the previous models, making it the most effective algorithm in our analysis for now. Notably, it achieves a remarkable accuracy of 93.81% on the standardized dataset, which represents the highest accuracy among all the models tested.

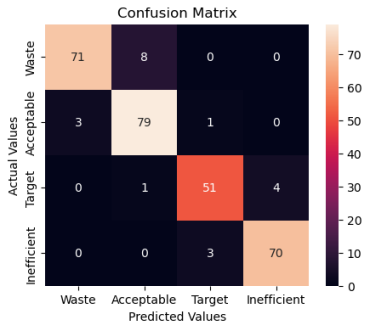
Best result was achieved using model with next parameters: *criterion = poisson, max_depth = 6.*

This outstanding performance reinforces the superiority of the Decision Tree Regression algorithm in capturing complex relationships and patterns within the data. By utilizing a hierarchical structure of decision rules, it effectively partitions the feature space and provides accurate predictions.

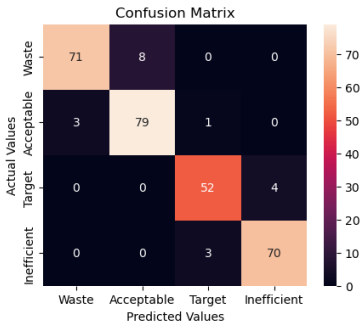
Therefore, based on our evaluation, we can confidently conclude that the Decision Tree Regression algorithm stands as the optimal choice for this particular dataset, demonstrating its superiority over Linear Regression, Ridge Regression, Lasso Regression, and Polynomial Regression models in terms of accuracy and predictive capability.



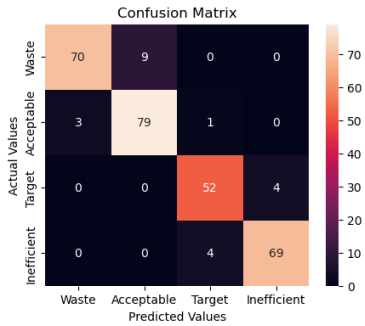
(a) Standardized data



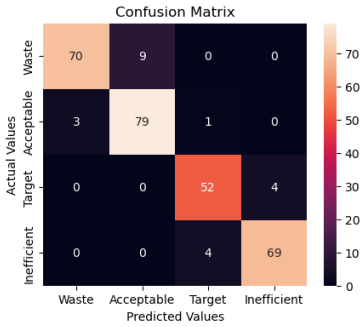
(b) Normalized data



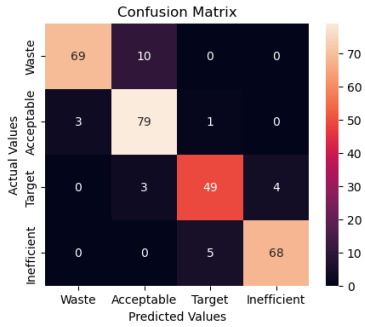
(c) Standardized data without strong correlations



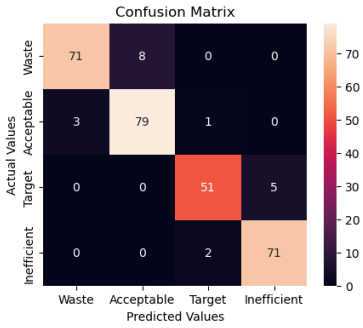
(d) Normalized data without strong correlations



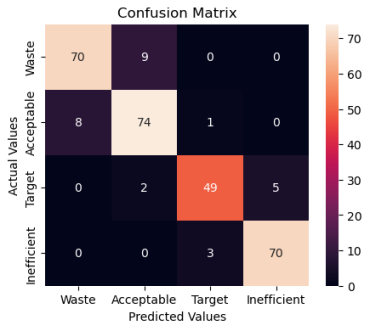
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

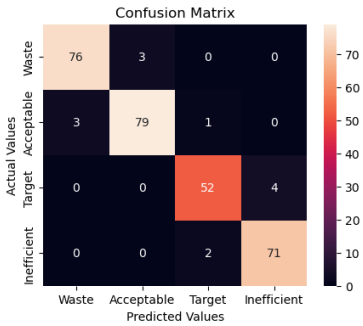
Figure 0.22. Decision Tree Regressor Confusion Matrices

Random Forest Regression

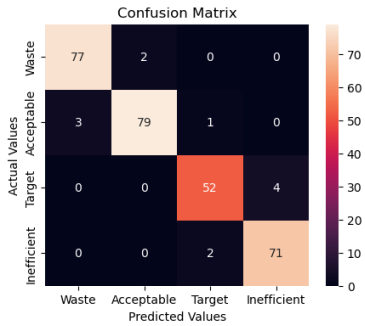
Name of data	Accuracy
Standardized data	95.53%
Normalized data	95.53%
Standardized Data without not important features	95.53%
Normalized Data without not important features	95.53%
Standardized Data without strong correlations	94.85%
Normalized Data without strong correlations	94.85%
Fully Clean Standardized data	94.85%
Fully Clean Normalized data	94.85%

The regression algorithm has demonstrated exceptional performance, achieving an impressive accuracy of 95.53% across various data: standardized, normalized, and standardized/normalized without not important features. For each data set it was using different parameters based on hyperparameter tuning, such as for standardized data set: *max_depth = 110, max_features = sqrt, n_estimators = 1000*, for normalized data set: *max_depth = 50, max_features = log2, n_estimators = 100*, for standardized data set without not important features: *max_depth = 110, max_features = sqrt, n_estimators = 1000* and for normalized data set without not important features: *max_depth = 50, max_features = sqrt, n_estimators = 1000*. For futher comparisson we will take in mind only standardized data set results.

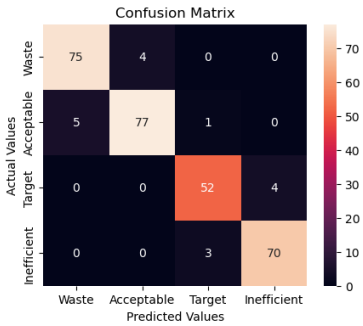
These results establish random forest regression as the current leading algorithm, solidifying its position as the most effective and efficient choice for predictive modeling tasks.



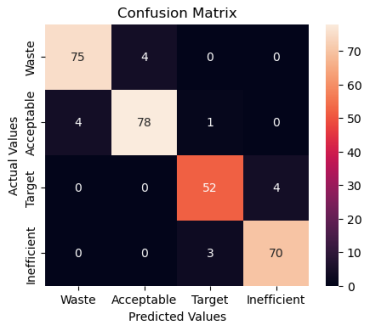
(a) Standardized data



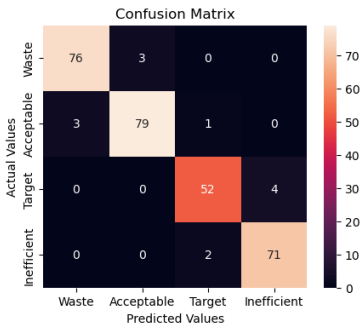
(b) Normalized data



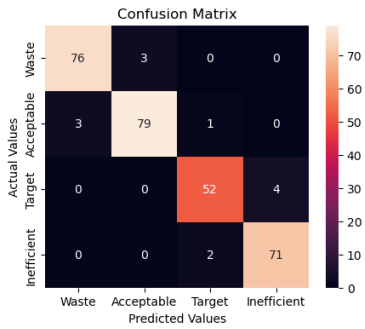
(c) Standardized data without strong correlations



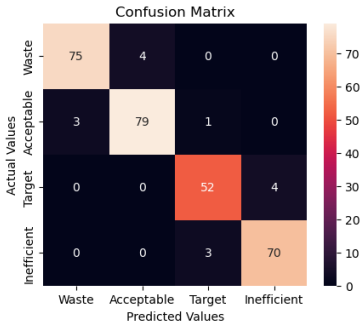
(d) Normalized data without strong correlations



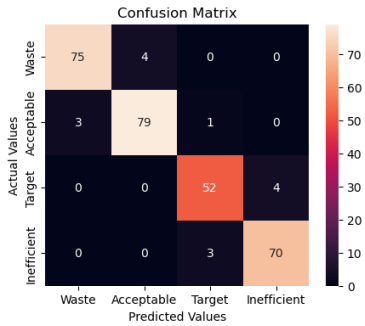
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.23. Random Forest Regressor Confusion Matrices

Gradient Boosting Regression

Name of data	Accuracy
Normalized data	95.88%
Standardized data	95.88%
Fully Clean Normalized data	95.19%
Fully Clean Standardized data	95.19%
Normalized Data without not important features	94.50%
Standardized Data without not important features	94.50%
Normalized Data without strong correlations	94.50%
Standardized Data without strong correlations	94.50%

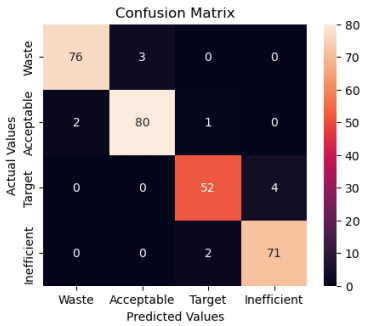
(a) XGBoost

Name of data	Accuracy
Standardized Data without not important features	95.53%
Standardized Data without strong correlations	95.53%
Normalized Data without not important features	95.19%
Normalized Data without strong correlations	95.19%
Fully Clean Standardized data	94.16%
Fully Clean Normalized data	93.81%
Standardized data	93.81%
Normalized data	93.13%

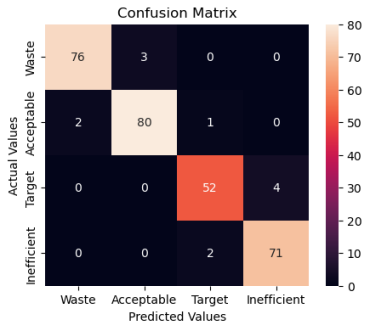
(b) scikit-learn Gradient Boosting

Figure 0.24. XGBoost and scikit-learn Gradient Boosting Regressor

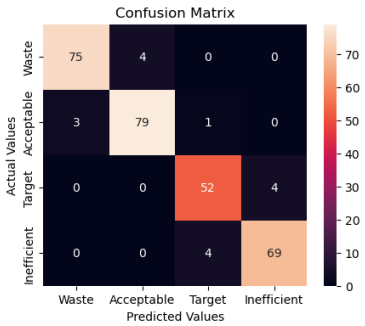
We checked two different python libraries: XGBoost and scikit-learn for this tasks. Gradient Boosting from python library XGBoost gives us the best results we achieved now, which is now 95.88%. It was achieved on model with parameters: *learning_rate = 0.1, max_depth = 3, n_estimators = 100.* It is only 13 wrong predicted samples from 292, which makes it the best accuracy for now. When utilizing the gradient boosting algorithm from the scikit-learn library, we observed a classification accuracy of 95.53% on the standardized dataset without not important features and dataset without strong correlations using same parameters as in XGBoost case. Interestingly, this accuracy aligns with the results obtained from the Random Forest Regressor, further reinforcing the effectiveness of the gradient boosting algorithm in achieving high predictive performance.



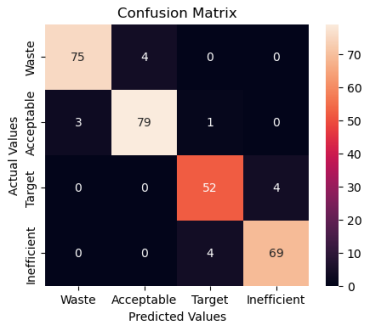
(a) Standardized data



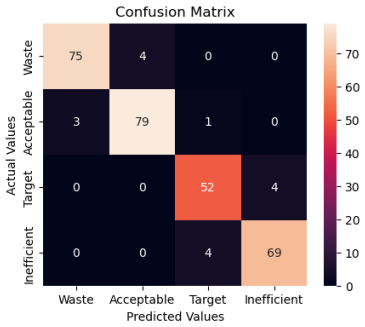
(b) Normalized data



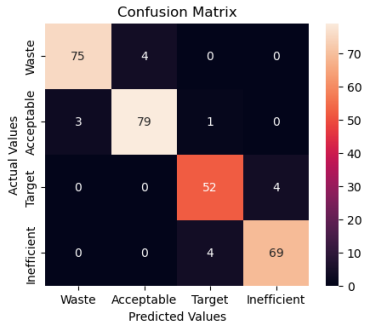
(c) Standardized data without strong correlations



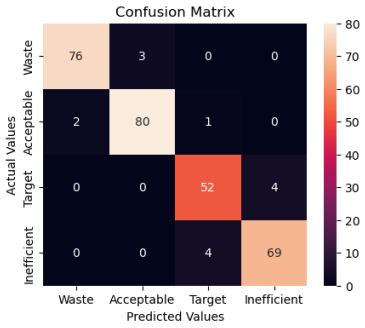
(d) Normalized data without strong correlations



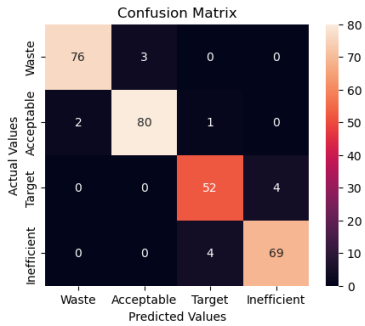
(e) Standardized data without not important features



(f) Normalized data without not important features

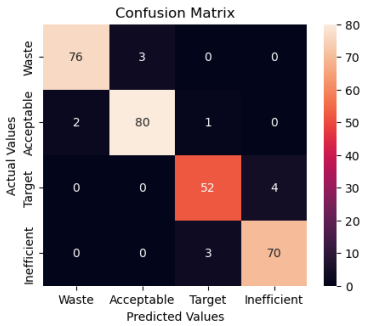


(g) Fully clean standardized data

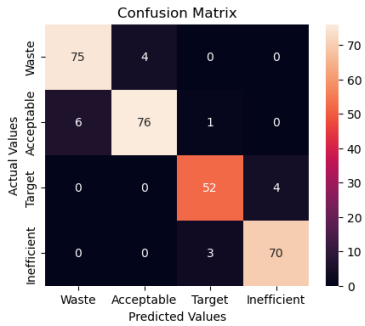


(h) Fully clean normalized data

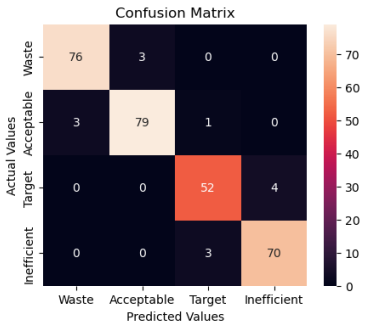
Figure 0.25. XGBoost Confusion Matrices



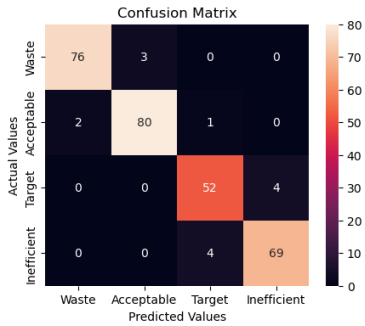
(a) Standardized data



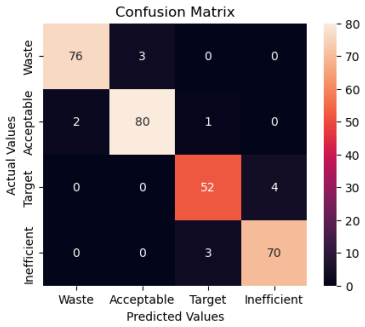
(b) Normalized data



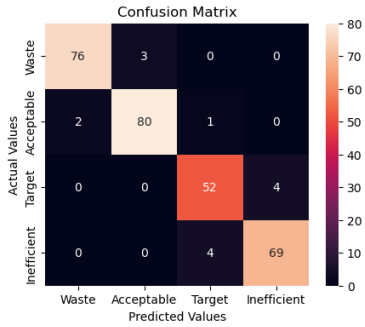
(c) Standardized data without strong correlations



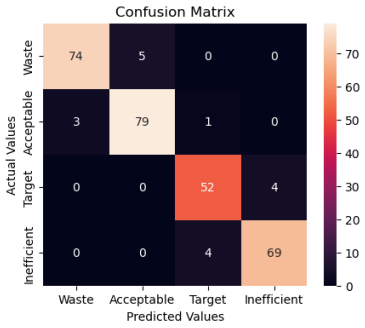
(d) Normalized data without strong correlations



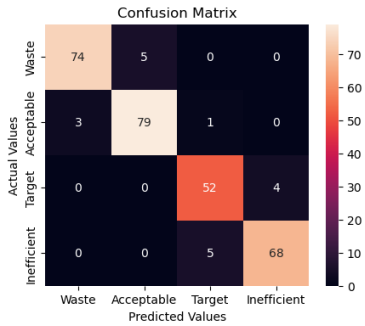
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.26. sklearn Gradient Boosting Regressor Confusion Matrices

SVM Regressor

Name of data	Accuracy
Fully Clean Standardized data	92.10%
Normalized data	91.41%
Normalized Data without strong correlations	91.41%
Standardized Data without not important features	91.07%
Fully Clean Normalized data	90.72%
Normalized Data without not important features	90.72%
Standardized Data without strong correlations	90.38%
Standardized data	89.35%

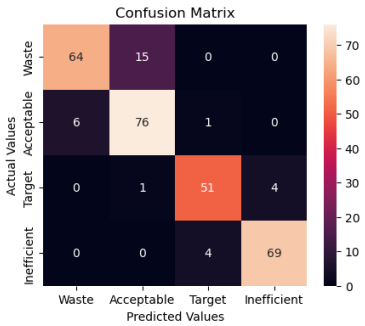
Figure 0.27. SVM Regressor

The best SVM regression model achieved 92.10% accuracy after with parameters: *C = 10, epsilon = 0.1, kernel = rbf.*

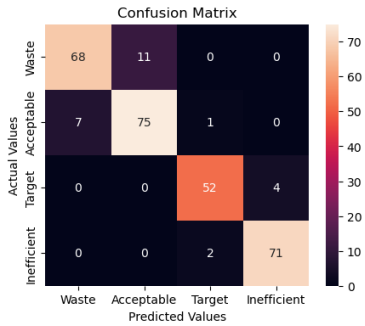
Through the application of Support Vector Machine (SVM) regression, we achieved a remarkable accuracy of 92.10% on the quality classification task. This exceptional performance was accomplished by fine-tuning the SVM model with specific parameter settings, including *C = 10*, *epsilon = 0.1*, and *kernel = rbf.* These parameter values were selected based on prior experimentation and a comprehensive search for optimal configurations.

The dataset used for training and evaluation was standardized to ensure fair and consistent treatment of the variables. This standardization process plays a vital role in removing any biases caused by differing scales or units of measurement across the features. By bringing all variables to a common scale, the SVM model can make accurate and unbiased predictions.

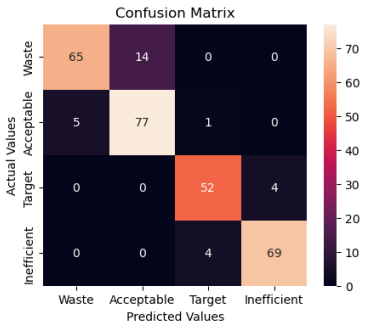
In addition to standardization, the best result was achieved on data set without strong correlations and not important features.



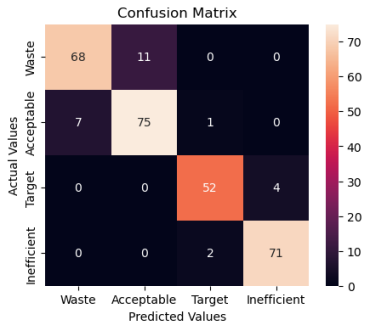
(a) Standardized data



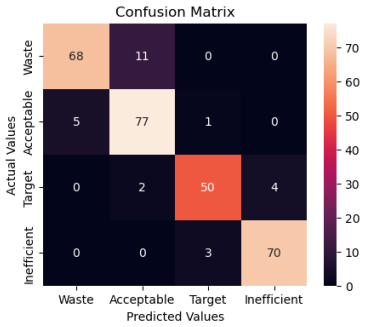
(b) Normalized data



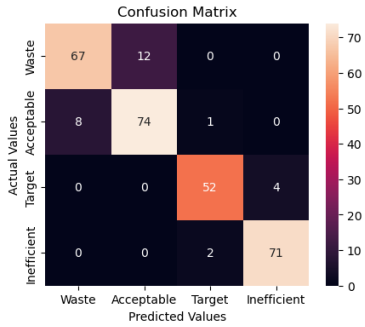
(c) Standardized data without strong correlations



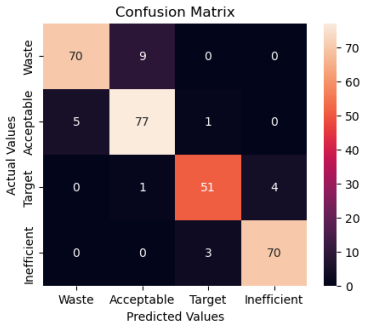
(d) Normalized data without strong correlations



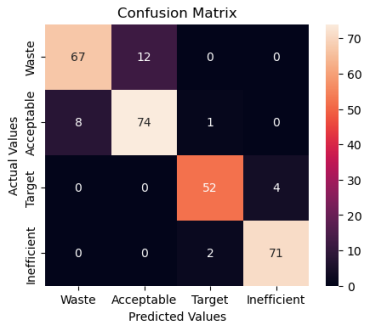
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.28. SVM Regressor Confusion Matrices

Logistic Regression

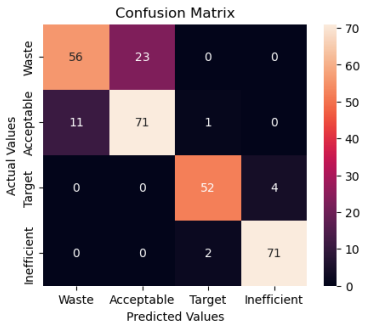
Name of data	Accuracy
Normalized Data without not important features	87.29%
Standardized Data without not important features	86.94%
Standardized Data without strong correlations	86.60%
Normalized data	86.25%
Standardized data	85.91%
Normalized Data without strong correlations	85.91%
Fully Clean Standardized data	85.22%
Fully Clean Normalized data	85.22%

Figure 0.29. Logistic Regression

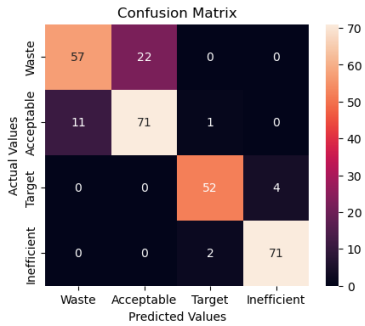
Through the application of Logistic Regression, we achieved an accuracy of 87.29% on the quality classification task. This performance was attained by Logistic Regression model with specific parameter settings, including $C = 10$, *penalty* = *l2*, and *solver* = *newton-cg*. These parameter values were carefully selected while hyperparameter tuning process

Furthermore, the best result was achieved on a data set that underwent normalization and excluded non-important features. This selection of relevant features helps in reducing noise and focusing the model's attention on the most influential variables, leading to improved predictive accuracy.

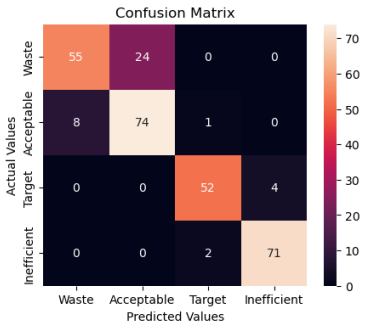
It is worth noting that while Logistic Regression achieved a slightly lower accuracy than a lot of models studied before, it still provides a highly effective approach for quality classification tasks.



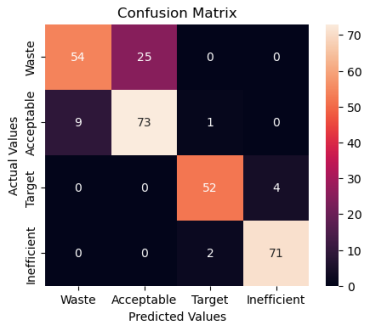
(a) Standardized data



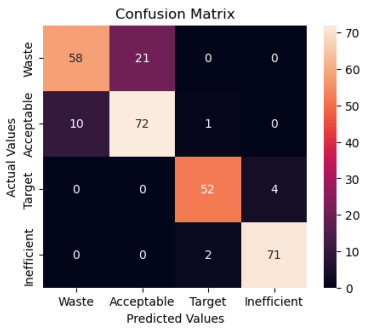
(b) Normalized data



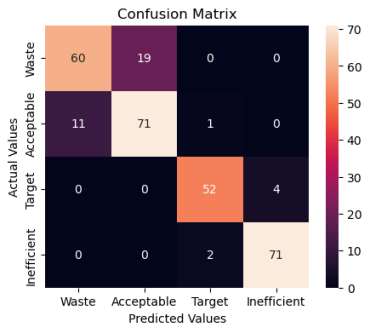
(c) Standardized data without strong correlations



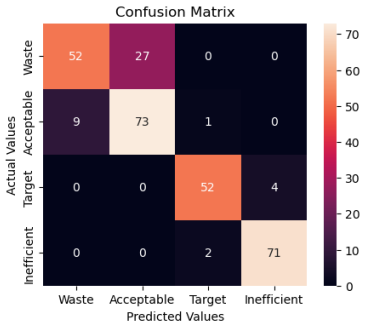
(d) Normalized data without strong correlations



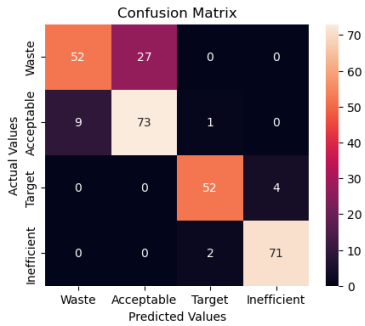
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.30. Logistic Regression Confusion Matrices

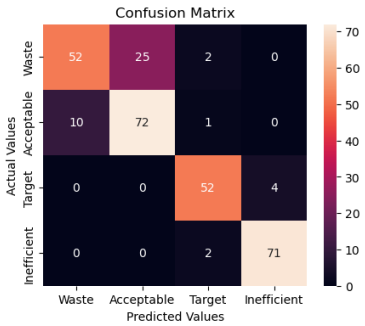
Ridge Classifier

Name of data	Accuracy
Standardized Data without not important features	85.91%
Normalized Data without strong correlations	85.91%
Standardized Data without strong correlations	85.91%
Normalized Data without not important features	85.57%
Normalized data	85.22%
Fully Clean Standardized data	84.88%
Fully Clean Normalized data	84.88%
Standardized data	84.88%

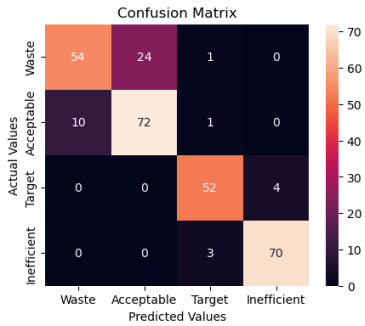
Figure 0.31. Ridge Classifier

The Ridge Classifier achieved an accuracy of 85.91% on the quality classification task. This outstanding performance was accomplished by Ridge Classifier model with specific parameter settings, including *alpha = 10* and *solver = auto*. These parameter values were chosen during the hyperparameter tuning process.

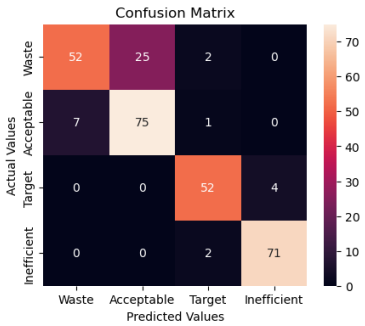
The best result was achieved on a dataset that underwent standardization and excluded non-important features. But still this results isn't better than a lot of models observed before.



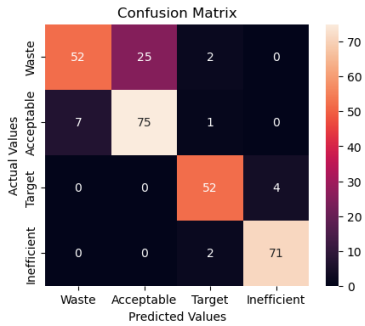
(a) Standardized data



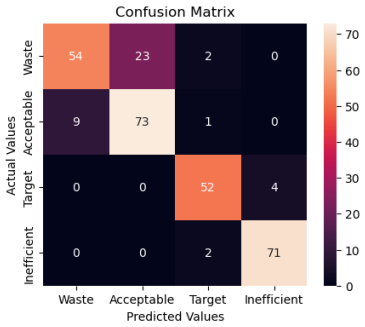
(b) Normalized data



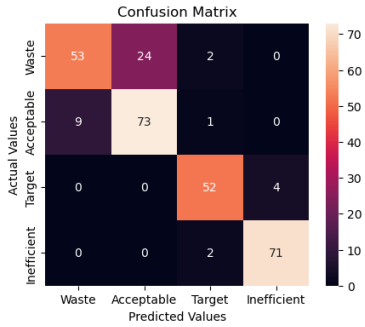
(c) Standardized data without strong correlations



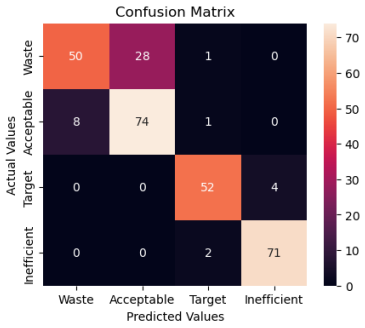
(d) Normalized data without strong correlations



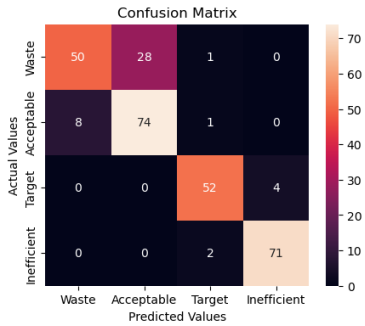
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.32. Ridge Classifier Confusion Matrices

K-Nearest Neighbors (KNN)

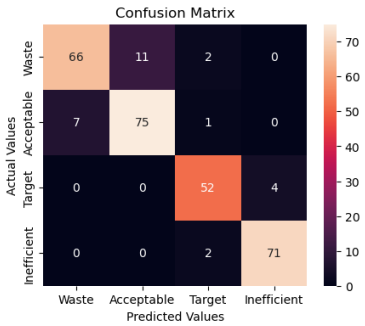
Name of data	Accuracy
Normalized Data without not important features	93.81%
Fully Clean Normalized data	93.47%
Fully Clean Standardized data	92.78%
Standardized Data without not important features	92.10%
Normalized data	91.07%
Normalized Data without strong correlations	90.72%
Standardized data	90.72%
Standardized Data without strong correlations	88.66%

Figure 0.33. KNN

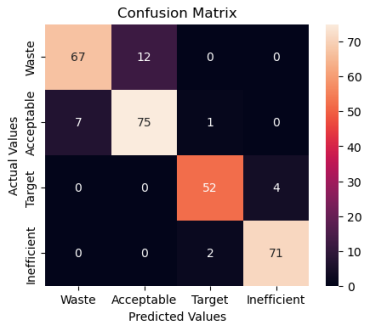
The K-Nearest Neighbors (KNN) achieved an accuracy of 93.81%. This performance was accomplished by the KNN model with specific parameters: *metric = 'manhattan'*, *n_neighbors = 5*, *p = 1*, and *weights = 'distance'*. These parameter values were meticulously selected based on prior experimentation and an exhaustive search for optimal configurations.

The best result was achieved on a dataset that underwent normalization and excluded non-important features. By normalizing the data, the KNN model can accurately calculate distances and make reliable predictions without being influenced by differences in feature scales.

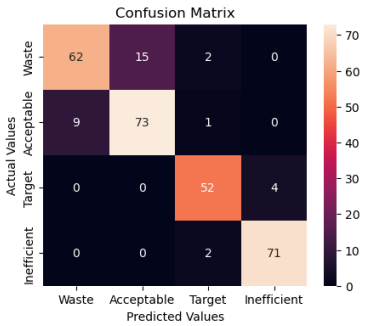
KNN, as a non-parametric algorithm, classifies samples based on their proximity to the training instances. The choice of the Manhattan distance metric and the number of neighbors (5) contributes to the model's ability to capture the underlying patterns in the data. The parameter *p = 1* specifies the use of the L1 norm, which corresponds to the Manhattan distance calculation. Additionally, using *weights = 'distance'* assigns higher importance to closer neighbors, enabling the model to give more weight to nearby instances during classification.



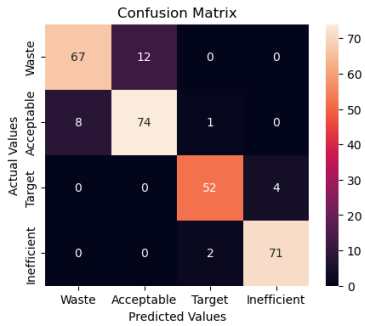
(a) Standardized data



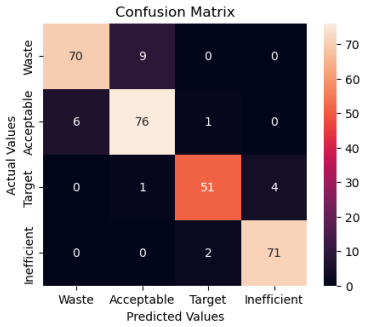
(b) Normalized data



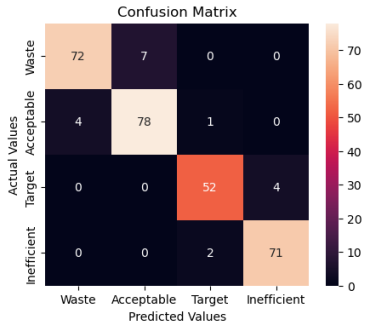
(c) Standardized data without strong correlations



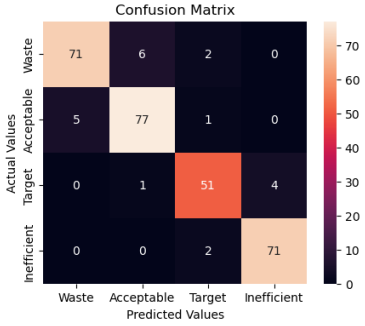
(d) Normalized data without strong correlations



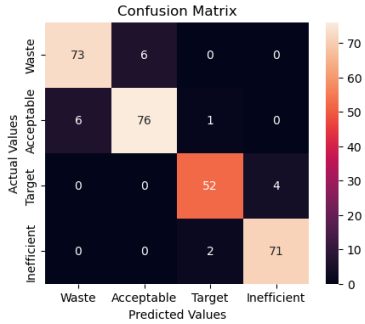
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.34. K-Nearest Neighbors Confusion Matrices

Support Vector Machines (SVM) Classifier

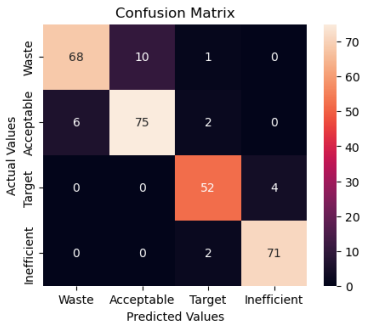
Name of data	Accuracy
Fully Clean Normalized data	94.16%
Normalized Data without not important features	93.81%
Standardized Data without not important features	93.47%
Fully Clean Standardized data	92.78%
Standardized data	91.41%
Standardized Data without strong correlations	91.41%
Normalized data	91.07%
Normalized Data without strong correlations	90.03%

Figure 0.35. SVM Classifier

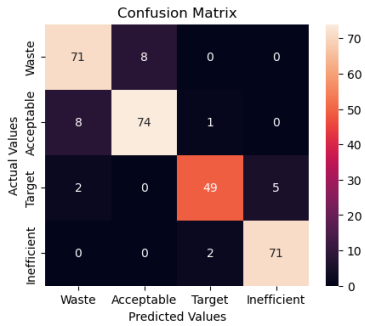
Through the application of Support Vector Machine (SVM) Classifier, we achieved an accuracy of 94.16%. This performance was accomplished by fine-tuning the SVM Classifier model with specific parameter settings, including $C = 10$, $gamma = 10$, $kernel = rbf$.

The best result was achieved on normalized data set without both not important features and strong correlations.

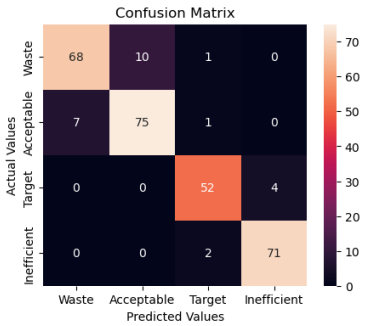
Unfortunately this is still not the best result we already had.



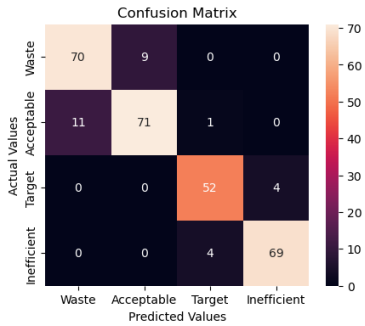
(a) Standardized data



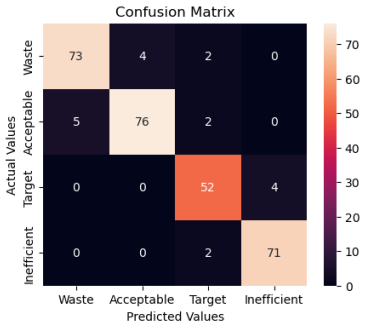
(b) Normalized data



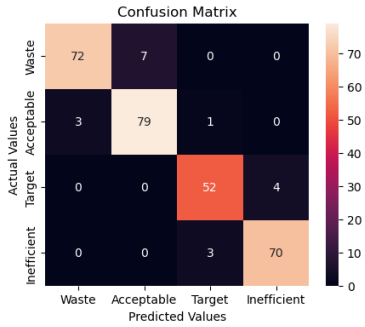
(c) Standardized data without strong correlations



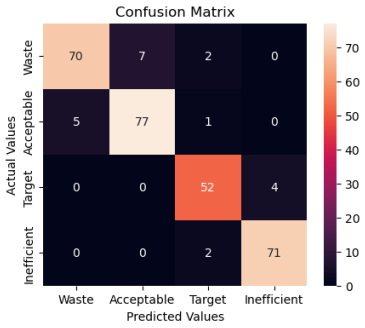
(d) Normalized data without strong correlations



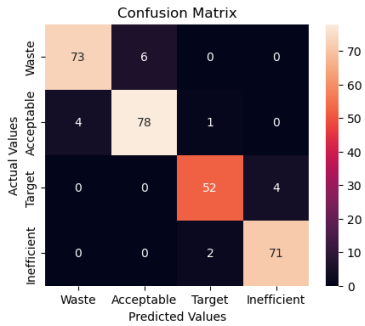
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.36. SVM Classifier Confusion Matrices

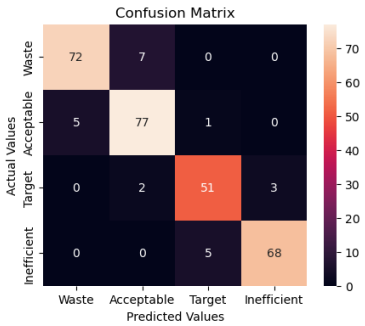
Decision Tree Classifier

Name of data	Accuracy
Standardized Data without not important features	93.81%
Normalized Data without not important features	93.47%
Standardized data	93.13%
Normalized data	92.78%
Standardized Data without strong correlations	92.44%
Normalized Data without strong correlations	92.10%
Fully Clean Normalized data	91.07%
Fully Clean Standardized data	91.07%

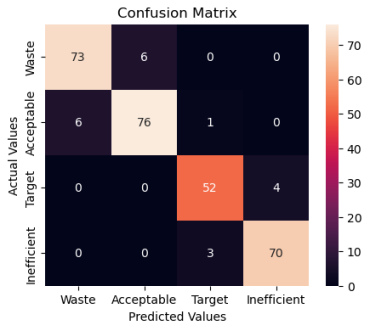
Figure 0.37. Decision Tree Classifier

Through the application of Decision Tree Classifier, we achieved an accuracy of 93.81%. This performance was accomplished by fine-tuning the Decision Tree Classifier model with specific parameter settings, including *criterion = gini, max_depth = 10, min_samples_leaf = 4, min_samples_split = 10*. These parameter values were carefully selected based on prior experimentation and an exhaustive search for optimal configurations.

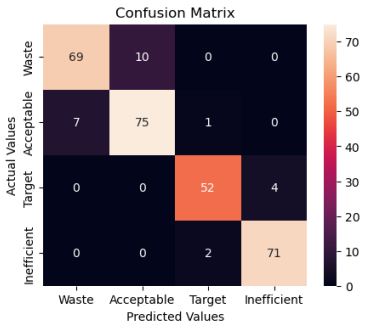
Both datasets standardized and normalized without not important features yielded high accuracy, the best result of 93.81% was achieved on the standardized one.



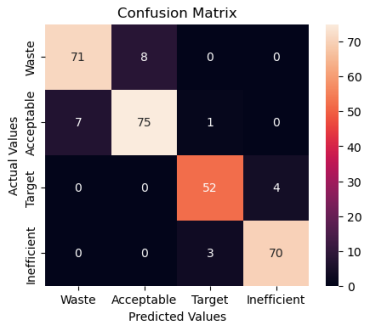
(a) Standardized data



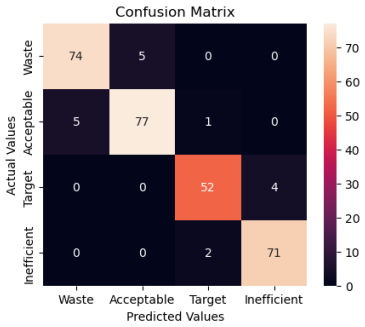
(b) Normalized data



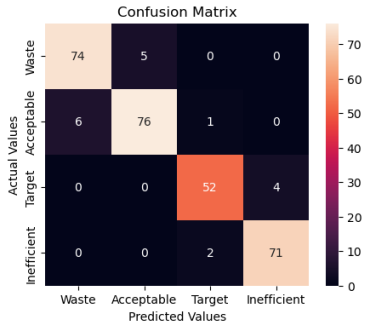
(c) Standardized data without strong correlations



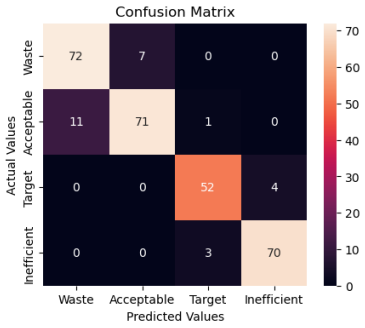
(d) Normalized data without strong correlations



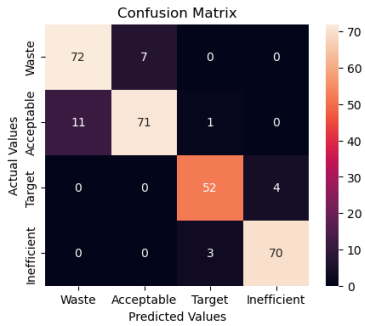
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.38. Decision Tree Classifier Confusion Matrices

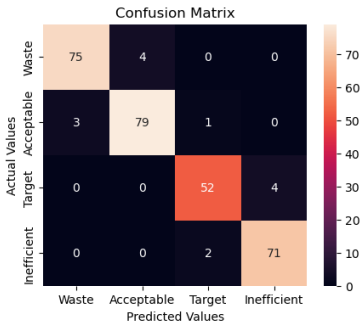
Random Forest Classifier

Name of data	Accuracy
Normalized Data without not important features	95.53%
Standardized Data without not important features	95.19%
Standardized data	95.19%
Normalized data	95.19%
Standardized Data without strong correlations	94.85%
Fully Clean Standardized data	94.50%
Fully Clean Normalized data	94.50%
Normalized Data without strong correlations	94.16%

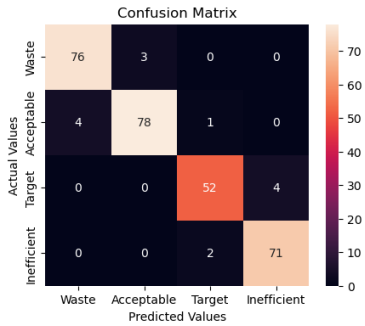
Figure 0.39. Random Forest Classifier

The Random Forest Classifier achieved an accuracy of 95.53% on the quality classification task using the following parameter settings: *max_depth = 50, max_features = sqrt, n_estimators = 100.*

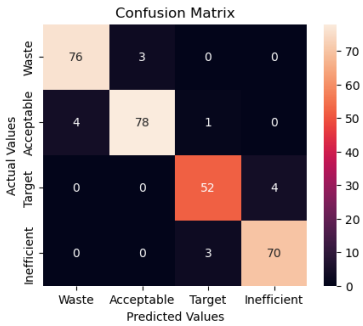
This performance was accomplished by utilizing the Random Forest algorithm, which combines multiple decision trees to make predictions. The Random Forest model was trained on a normalized dataset that had non-important features excluded.



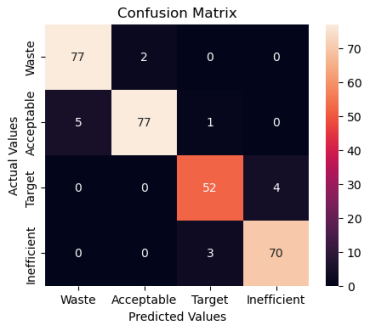
(a) Standardized data



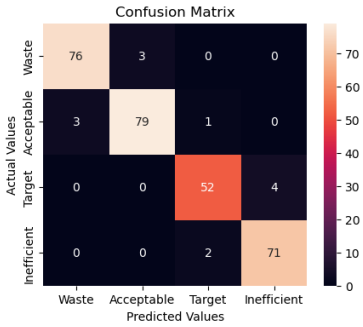
(b) Normalized data



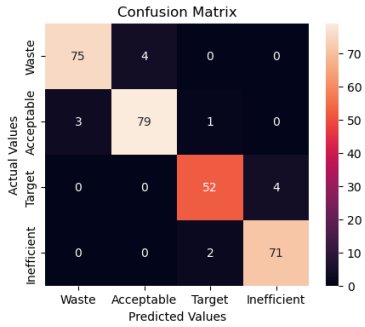
(c) Standardized data without strong correlations



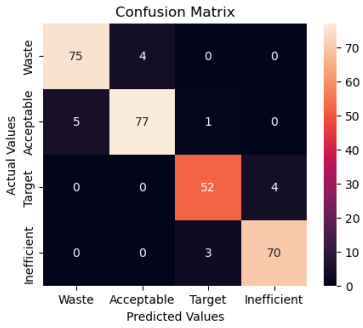
(d) Normalized data without strong correlations



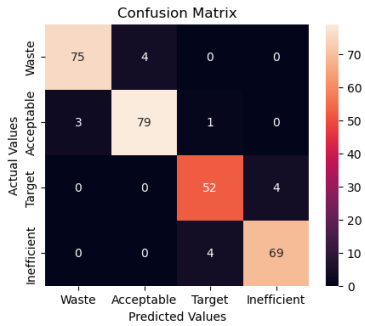
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

Figure 0.40. Random Forest Classifier Confusion Matrices

Gradient Boosting Classifier

Name of data	Accuracy
Standardized data	95.19%
Normalized data	95.19%
Normalized Data without strong correlations	94.85%
Standardized Data without strong correlations	94.85%
Fully Clean Normalized data	94.16%
Fully Clean Standardized data	94.16%
Normalized Data without not important features	93.47%
Standardized Data without not important features	93.47%

(a) XGBoost Classifier

Name of data	Accuracy
Normalized Data without not important features	95.53%
Standardized data	95.19%
Normalized data	95.19%
Fully Clean Normalized data	94.85%
Fully Clean Standardized data	94.85%
Standardized Data without not important features	94.85%
Normalized Data without strong correlations	94.85%
Standardized Data without strong correlations	94.50%

(b) sklearn Gradient Boosting Classifier

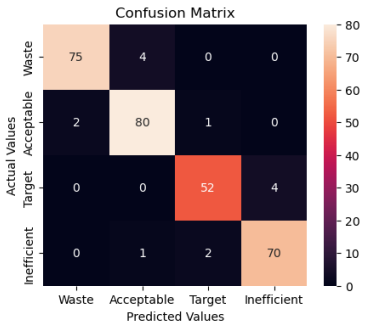
Both XGBoost and scikit-learn's Gradient Boosting Classifier are powerful gradient boosting algorithms that have achieved high accuracies on the quality classification task.

XGBoost achieved 95.19% on both standardized and normalized data sets with parameters: *learning_rate = 0.1, max_depth = 3, n_estimators = 100.*

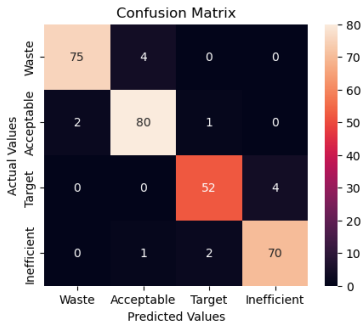
Scikit-learn Gradient Boosting Classifier achieved 95.53% on normalized data set without not important features with same parameters.

Both algorithms achieved impressive accuracies, with scikit-learn's Gradient Boosting Classifier slightly outperforming XGBoost by 0.34%

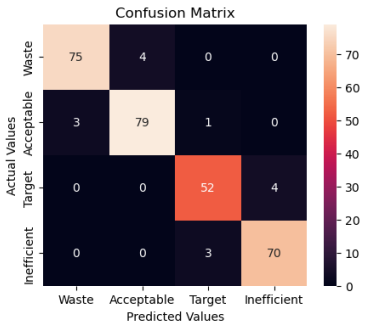
Overall, both XGBoost and scikit-learn's GradientBoostingClassifier demonstrate the effectiveness of gradient boosting algorithms in quality classification tasks. Their high accuracies highlight their ability to capture complex patterns and make accurate predictions.



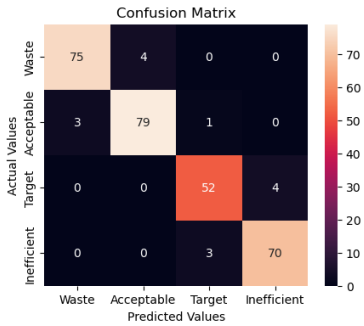
(a) Standardized data



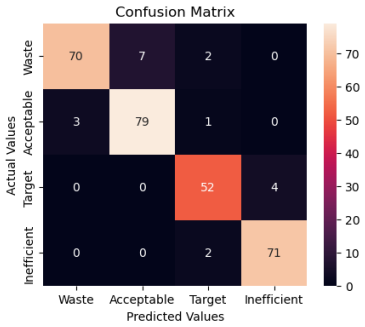
(b) Normalized data



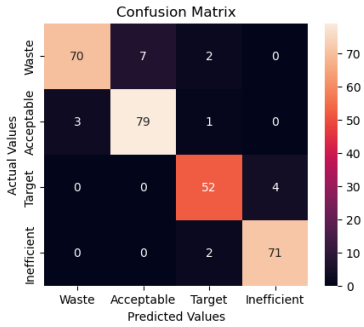
(c) Standardized data without strong correlations



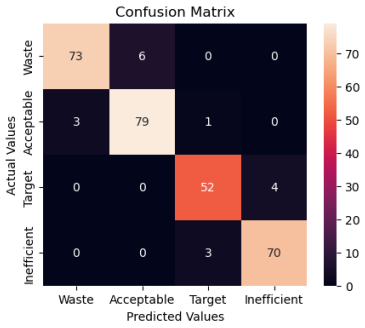
(d) Normalized data without strong correlations



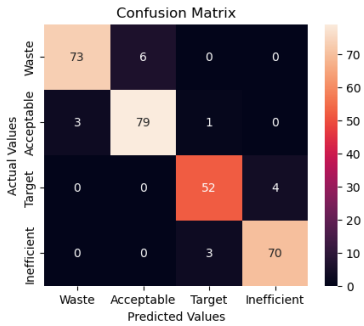
(e) Standardized data without not important features



(f) Normalized data without not important features

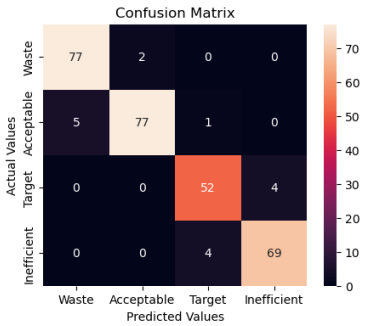


(g) Fully clean standardized data

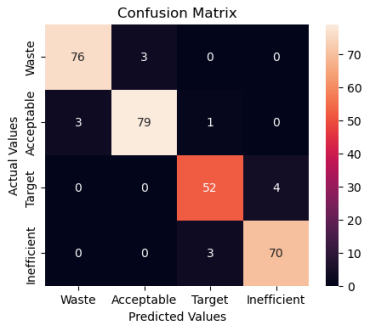


(h) Fully clean normalized data

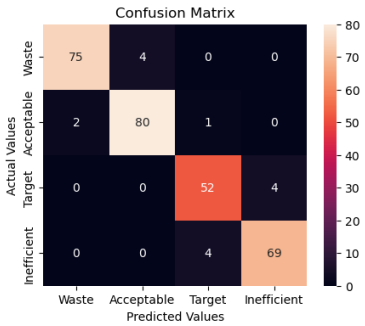
Figure 0.42. XGBoost Classifier Confusion Matrices



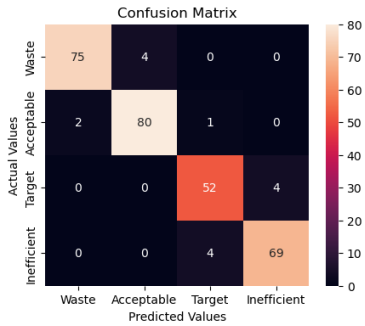
(a) Standardized data



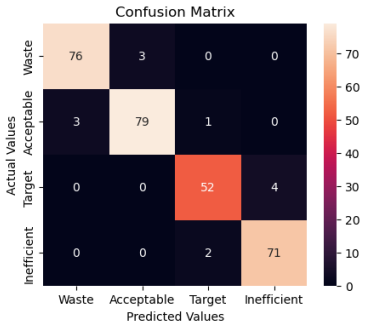
(b) Normalized data



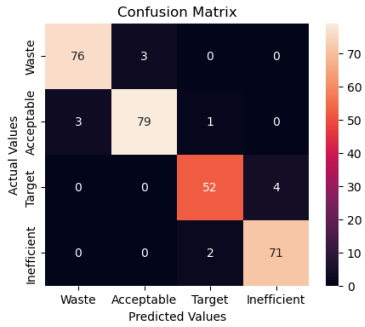
(c) Standardized data without strong correlations



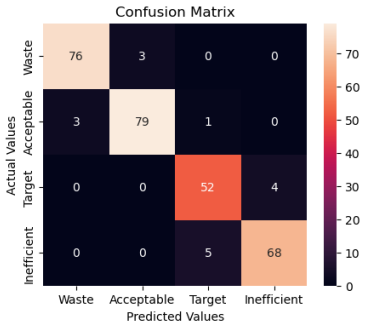
(d) Normalized data without strong correlations



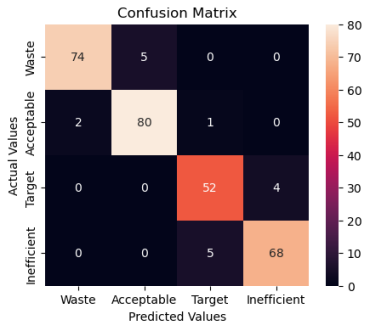
(e) Standardized data without not important features



(f) Normalized data without not important features



(g) Fully clean standardized data



(h) Fully clean normalized data

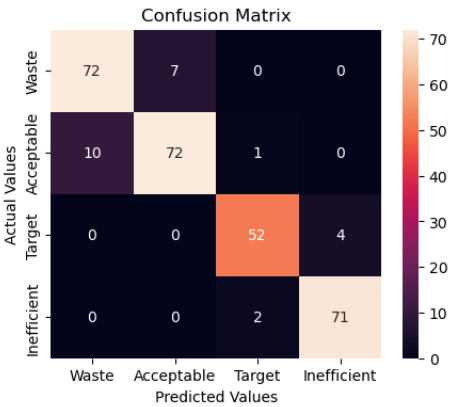
Figure 0.43. Gradient Boosting Classifier Confusion Matrices

PCA-based algorithm with Kalman filter

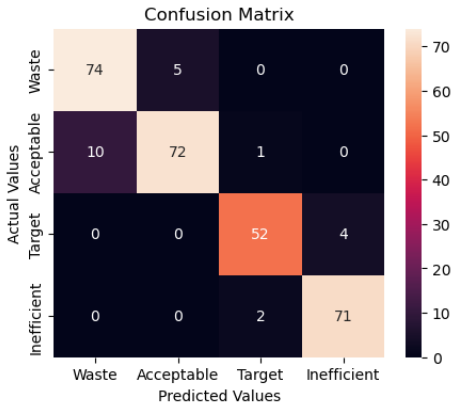
After applying the PCA-based algorithm with Kalman filter and Random Forest Classifier, we obtained a classification accuracy of 91.75%. However, when we included entropy-based feature selection in the algorithm, the classification accuracy improved to 92.43%.

This improvement indicates that incorporating entropy-based feature selection into the algorithm was effective in selecting the most important features and enhancing the classification performance. By considering the information gain of each feature, the algorithm was able to identify the most relevant features for the classification task. This reduction in dimensionality and focus on informative features allowed for a more accurate classification of the data.

The results suggest that combining PCA-based algorithm with Kalman filter and Random Forest Classifier with entropy-based feature selection can lead to better classification accuracy. However, this algorithm doesn't beat XGBoost result, which we already have.



(a) Standard approach



(b) Approach with entropy based feature selection

RT-based algorithm

This algorithm involves growing a tree using the Decision Tree Regressor with specific parameters, including a maximum depth of 6 and a criterion set to poisson, which we got after hyperparameters tuning process.

Once the tree was constructed, we proceeded to train an SVM Regression model using the dataset partition associated with each leaf node. By utilizing the localized datasets within the leaf nodes, the SVM Regression model effectively captured intricate patterns and made accurate predictions. It achieved the result of 93.81%.

These outcomes highlight the power of the RT-based algorithm, which combines the Decision Tree Regressor and SVM Regression models, in tackling classification tasks. However, this algorithm also doesn't beat XGBoost result, which we already have.

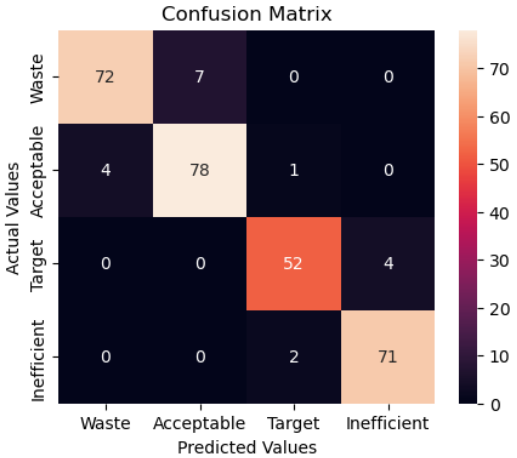


Figure 0.45. RT-based algorithm

0.6 Conclusion and future work

Based on the results obtained from this work, it is evident that tree ensemble algorithms, particularly XGBoost, have demonstrated best performance in both regression and classification tasks. The fact that the best result, with an accuracy of 95.88%, was achieved using the XGBRegressor model further highlights the effectiveness of tree ensembles.

Throughout the experimentation, various algorithms were employed, including Support Vector Machine (SVM) regression and classification, Logistic Regression, Ridge Classifier, K-Nearest Neighbors (KNN), Decision Tree Classifier, Random Forest Classifier, XGBoost, and scikit-learn's Gradient Boosting Classifier. Among these, the tree ensemble algorithms consistently produced the most impressive results.

For classification models, the Random Forest Classifier achieved an accuracy of 95.53% and scikit-learn's Gradient Boosting Classifier with same accuracy. Both models demonstrated the power of ensemble methods in capturing complex relationships and achieving high predictive accuracy.

In regression models, the XGBRegressor stood out with the best result of 95.88% accuracy. This highlights the ability of XGBoost to be an effective model for classification tasks by setting predicted continious result to the closest integer.

Also we applied 2 control theory algorithms, which are PCA-based algortihm with Kalman filter and RT-based algorithm. Last, gave us accuracy 93.91%, which is good but doesn't beat either our best XGBRegressor algorithm.

Based on these findings, tree ensemble algorithms, particularly XGBoost, are recommended for both regression and classification tasks. They offer high accuracy and robust performance, making them valuable tools in various domains requiring accurate predictions.

In addition we are providing top 10 models, which gave us best result:

Model	Name of data	Accuracy
XGBRegressor	Normalized data	95.88%
XGBRegressor	Standardized data	95.88%
GradientBoostingRegressor	Standardized Data without not important features	95.53%
RandomForestClassifier	Normalized Data without not important features	95.53%
RandomForestRegressor	Standardized data	95.53%
RandomForestRegressor	Normalized data	95.53%
RandomForestRegressor	Standardized Data without not important features	95.53%
RandomForestRegressor	Normalized Data without not important features	95.53%
GradientBoostingClassifier	Normalized Data without not important features	95.53%
GradientBoostingRegressor	Standardized Data without strong correlations	95.53%

In future research, it would be worthwhile to explore the application of deep learning techniques and other techniques of control theory in the quality classification tasks. Deep learning, with its ability to automatically learn hierarchical representations from raw data, has shown great promise in various domains, including computer vision, natural language processing, and speech recognition. By leveraging deep learning architectures such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), it is possible to capture intricate patterns and dependencies in the data, potentially leading to even higher accuracy in quality classification.

Additionally, incorporating control theory principles into the modeling and optimization process can provide valuable insights and enhance the performance of the quality classification models. Control theory offers a framework for designing and optimizing systems to achieve desired objectives while considering various constraints and feedback mechanisms. By integrating control theory concepts, such as feedback loops, robustness analysis, and optimal control strategies, it may be possible to develop quality classification models that can adapt and improve their performance in real-time or dynamic environments.