Performance Evaluation of Classification Models on the Wine Recognition Dataset Using Supervised Learning Techniques

ANIRBAN DEY
ENPM808L
University of Maryland
College Park, U.S.A
adey9497@umd.edu

***Abstract*—The classification performance of various machine learning models in the context of the Wine Recognition dataset is explored to determine the most effective algorithm for predicting wine classes.**

Employing five distinct machine learning models—Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors, Naive Bayes, and Linear Discriminant Analysis—the study involves training and evaluating each model using a range of metrics, including accuracy, confusion matrices, and cross-validation scores.

The evaluation reveals varying degrees of success across the models, with Naive Bayes and Linear Discriminant Analysis exhibiting notable accuracy and precision in classifying wine types. Confusion matrices and cross-validation scores further contribute to a comprehensive understanding of model performance. At the end Naive Bayes classifier just edges over LDA in being the best model chosen for wine dataset analysis primarily becuase of it's cross validation accuracy.

This study is crucial for practitioners and researchers in machine learning, providing insights into the effectiveness of different algorithms for wine classification tasks. The comparison of models, including their strengths and weaknesses, aids in selecting an appropriate approach for similar real-world applications. The inclusion of visualization techniques, such as confusion matrices and cross-validation plots, enhances the interpretability of results and facilitates a deeper understanding of the models' behavior. Overall, this report serves as a valuable resource for those seeking to optimize machine learning model selection for wine classification.

I. Introduction

The decision problem addressed in this study revolves around the application of various machine learning models for wine classification using the well-known Wine Recognition dataset. The primary goal is to assess and compare the performance of Naive Bayes and Linear Discriminant Analysis (LDA) classifiers alongside Logistic Regression, Decision Tree Classifier, and K-Nearest Neighbors (KNN) models. The decision-making process involves selecting the most suitable classification model based on its accuracy in predicting wine classes. This decision holds practical significance in fields such as viticulture, wine production, and quality control, where precise classification of wine types is crucial for informed decision-making and process optimization.

The Wine Recognition dataset employed in this study is a well-known benchmark dataset widely used for classification tasks. This dataset was introduced by Forina et al. (1988) and is often credited for its simplicity, elegance, and relevance to practical applications within the wine industry.

The dataset comprises a total of 178 instances, with each instance representing a different wine sample. These samples correspond to three distinct classes, denoted as Class 0, Class 1, and Class 2, each representing a different cultivar of wine. The classes in the dataset correspond to three different grape varieties: Class 0 represents the Barolo wine, Class 1 represents Grignolino, and Class 2 represents Barbera.

For each wine sample, the dataset includes thirteen features that characterize various chemical properties, such as alcohol content, phenols, color intensity, and other compositional attributes. These features provide a detailed representation of the chemical composition of each wine, making the dataset valuable for studying the factors that contribute to wine classification.

One of the notable aspects of the Wine Recognition dataset is its balanced nature, meaning that the number of instances in each class is roughly equal. This balance is beneficial for training and evaluating machine learning models, as it prevents biases towards any particular class and ensures a fair assessment of model performance across different wine types.

Due to its balanced and structured characteristics, the Wine Recognition dataset serves as an ideal choice for this comparative model assessment. Researchers and practitioners often use this dataset to evaluate the effectiveness of classification algorithms in predicting the correct wine class based on its chemical composition. The dataset's relevance to real-world scenarios, particularly in the wine industry, underscores its importance in studying and optimizing classification models for practical applications.

A comprehensive literature review situates this study within the broader context of machine learning applications for wine classification and provides insights into prior research efforts. Recent advancements in Naive Bayes classifier research and adaptations of Linear Discriminant Analysis to handle

non-linear decision boundaries are highlighted. Additionally, references to previous studies on Decision Trees, Logistic Regression, and KNN models contribute to a contextual understanding of the existing landscape.

The paper's structure encompasses various sections to provide a systematic exploration of the research problem. Following the introduction, Section II details the methodology for implementing and evaluating the classification models. Section III presents a thorough summary of the results, emphasizing the strengths and weaknesses of each model. Section IV delves into the implications of the findings, shedding light on the significance of model selection in the context of wine classification. The paper concludes in Section V, summarizing key takeaways and suggesting potential avenues for future research. This structured approach ensures a comprehensive and coherent exploration of the decision problem and its implications in the subsequent sections.

II. Methodologies

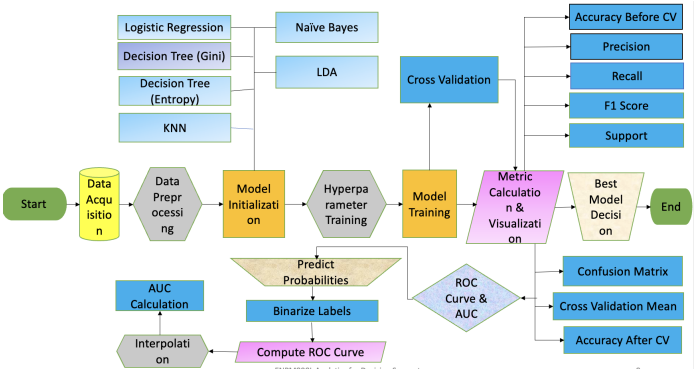

Fig. 1. ARCHITECTURAL DIAGRAM

This section outlines the methodology employed for the analysis and evaluation of three classification models on the Iris dataset. The objective of this study is to compare the performance of Naive Bayes Classifier, Linear Discriminant Analysis Classifier as well as Logistic Regression, Decision Tree Classifier, and K-Nearest Neighbors (KNN) Classifier in classifying iris species based on the features provided in the dataset. We begin by importing necessary libraries, including scikit-learn for model creation and evaluation, as well as visualization libraries like matplotlib and seaborn for data representation.The following steps detail the methodology utilized in this analysis:

A. Library Imports

The analysis commences by importing essential libraries that provide the fundamental tools for data manipulation, machine learning, and data visualization. These libraries include:

scikit-learn: A widely used machine learning library that offers various algorithms and utilities for classification and model evaluation.

matplotlib: A versatile library for creating visualizations and plots.

seaborn: A library built on top of matplotlib, specializing in producing aesthetically pleasing statistical graphics.

Also the following libraries from the scikit-learn ecosystem are included:

sklearn.model_selection: This module offers functionalities for data splitting, cross-validation, and model selection. In our study, it is employed for partitioning the dataset into training and testing sets, a critical step in assessing model performance.

sklearn.linear_model: This module encompasses various linear models, including Logistic Regression, a versatile algorithm often used in binary and multi-class classification tasks.

sklearn.tree: The Decision Tree Classifier is a non-linear model provided by this module. It is selected to explore non-linear classification approaches.

sklearn.neighbors: The K-Nearest Neighbors Classifier is an instance-based learning algorithm available in this module. It is suitable for a variety of classification problems.

sklearn.naive_bayes: This library provides a comprehensive set of tools for machine learning tasks, and GaussianNB serves as an efficient implementation of the Naive Bayes algorithm for datasets with continuous features.

sklearn.discriminant_analysis: This library provides a comprehensive set of tools for data analysis and modeling, and the LDA classifier is selected for its effectiveness in capturing class-separability patterns within the dataset, a crucial step in the overall classification process.

sklearn.metrics: Metrics are fundamental for evaluating the performance of machine learning models. This module contains functions for computing common classification metrics such as accuracy, precision, recall, and F1-score. These metrics play a pivotal role in assessing model quality and making informed decisions.

B. Data Acquisition

The initial phase of our analysis involves the acquisition of the wine recognition dataset from the `scikit_learn` library, which serves as the foundation for our classification task.Provided below is detailed breakdown of our data acquisition process:

```
wine = datasets.load_wine()
```

1) `datasets` Module: This module is part of scikit-learn, a popular machine learning library in Python. Scikit-learn provides various datasets that are commonly used for testing and experimenting with machine learning algorithms.
2) `load_wine()` Function: This function is specifically designed to load the Wine Recognition dataset. It returns a dictionary-like object containing the dataset's features, target variable, and additional information.
3) `wine` Variable: The result of calling `datasets.load_wine()` is assigned to the variable named `wine`. This variable now holds the dataset, and you can access its components using keys, similar to accessing elements in a dictionary.

C. Data Preparation

Upon Acquiring the Wine recognition dataset, we proceed to segregate the data into two main components. The data preparation for the Wine Recognition dataset is performed using the pandas library.

```
X = pd.DataFrame(wine.data, columns=wine.
feature_names)
y = pd.Series(wine.target, name='Class')
```

Provided below is the detailed breakdown of the above code:

1) pd.DataFrame(wine.data, columns=wine.feature_names):
   - `pd` refers to the pandas library.
   - `pd.DataFrame()` is a pandas function used to create a DataFrame, which is a two-dimensional labeled data structure.
   - `wine.data` contains the features of the Wine Recognition dataset, and `wine.feature_names` contains the names of these features.
   - This line creates a DataFrame (`X`) with the features of the dataset, where each column is labeled with the corresponding feature name.
2) `pd.Series(wine.target, name='Class')`:
   - This line creates a pandas Series (`y`) from the target variable (`wine.target`) of the Wine Recognition dataset.
   - The `name='Class'` argument assigns the name 'Class' to the Series, representing the class labels of the dataset.

This code segment is crucial for organizing the dataset into a format suitable for analysis. The features are stored in a DataFrame (`X`), making it easy to manipulate and explore, while the target variable (class labels) is stored in a Series (`y`). This structured format is essential for feeding the data into machine learning models during the training and evaluation phases. It provides a clear distinction between features and labels, facilitating the subsequent stages of model implementation and assessment.

D. Data Splitting

A crucial step in our analysis is to partition the dataset into a training set and a testing set, which allows for robust model assessment and evaluation. This process is achieved through the train-test-split function from the scikit-learn library. The main objectives of data splitting are as follows:

Training Set: A significant portion of the dataset (approximately 70-80%) is designated as the training set. This segment serves as the basis for model training. The chosen proportion ensures that the models have access to an adequate amount of data to learn and generalize effectively.

Testing Set: The remaining data (about 20-30%) is reserved for the testing set. This portion is employed to evaluate the models' performance. By using a separate testing set, we can assess how well the models can make accurate predictions on unseen data, which is a critical aspect of their performance.

```
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.3,
random_state=42)
```

Randomization: We utilize a fixed random seed (in this case, random-state=42) to ensure reproducibility of results. The use of a random seed guarantees that the data split remains consistent across multiple runs of the analysis, enhancing the reliability of the findings. This division into training and testing sets allows us to measure the models' ability to generalize to new, unseen data, and to assess their performance and accuracy when applied to real-world scenarios.

E. Model Initialization

The next phase of our analysis involves the initialization of five distinct machine learning models, each tailored to perform classification tasks:

```
logistic_reg = LogisticRegression()
decision_tree = DecisionTreeClassifier()
knn = KNeighborsClassifier(n_neighbors=3)
naive_bayes = GaussianNB()
lda = LinearDiscriminantAnalysis()
```

The code initializes instances of several machine learning models. Provided below is the detailed breakdown of the above code:

1) `LogisticRegression()`:
   - `LogisticRegression` is a class from scikit-learn representing a logistic regression model.
   - `logistic_reg` is an instance of this model, initialized without any specific hyperparameters.
2) `DecisionTreeClassifier()`:
   - `DecisionTreeClassifier` is a class from scikit-learn representing a decision tree classifier.
   - `decision_tree` is an instance of this model, initialized without any specific hyperparameters.
3) `KNeighborsClassifier(n_neighbors=3)`:
   - `KNeighborsClassifier` is a class from scikit-learn representing a k-nearest neighbors classifier.
   - `knn` is an instance of this model, initialized with the `n_neighbors` hyperparameter set to 3.
4) `GaussianNB()`:
   - `GaussianNB` is a class from scikit-learn representing a Gaussian Naive Bayes classifier.
   - `naive_bayes` is an instance of this model, initialized without any specific hyperparameters.
5) `LinearDiscriminantAnalysis()`:
   - `LinearDiscriminantAnalysis` is a class from scikit-learn representing a linear discriminant analysis model.
   - `lda` is an instance of this model, initialized without any specific hyperparameters.

This code segment is essential for preparing the various machine learning models that will be employed in the study. Each model is instantiated with default or specified hyperparameters, allowing for consistency and reproducibility in

subsequent model training and evaluation steps. The initialized models (`logistic_reg`, `decision_tree`, `knn`, `naive_bayes`, `lda`) will be used to train on the dataset and assess their performance in the later stages of the methodology.

F. Hyper parameter Tuning

To enhance the performance of the classification models, a systematic approach to hyperparameter tuning was employed. Hyperparameters are crucial parameters that are not learned during the training process and play a pivotal role in the behavior of machine learning algorithms.

K-Nearest Neighbors (KNN):

- Algorithm Description: KNN is a non-parametric and lazy-learning algorithm used for classification tasks. The KNN classifier assigns an observation to the majority class among its k nearest neighbors.
- Hyperparameter Tuning: A grid search was performed over the range of k values from 1 to 30 to identify the optimal number of neighbors (`n_neighbors`). The model's performance was evaluated using 5-fold cross-validation.

```
param_grid_knn = {'n_neighbors': np.arange(1,
30)}
grid_search_knn = GridSearchCV(
KNeighborsClassifier(), param_grid_knn, cv=5)
grid_search_knn.fit(X_train, y_train)
optimal_k = grid_search_knn.best_params_['
n_neighbors']
```

Decision Tree:

- Algorithm Description: Decision Trees are a popular family of models for classification. They recursively split the data based on features to create a tree-like structure.
- Hyperparameter Tuning: Separate grid searches were conducted for decision trees with Gini (`criterion='gini'`) and entropy (`criterion='entropy'`) as splitting criteria. The optimal maximum depth (`max_depth`) was explored over the range of 1 to 20 using 5-fold cross-validation.

```
param_grid_tree = {'max_depth': np.arange(1, 20)
}
grid_search_tree_gini = GridSearchCV(
DecisionTreeClassifier(criterion='gini'),
param_grid_tree, cv=5)
grid_search_tree_gini.fit(X_train, y_train)
optimal_depth_gini = grid_search_tree_gini.
best_params_['max_depth']

grid_search_tree_entropy = GridSearchCV(
DecisionTreeClassifier(criterion='entropy'),
param_grid_tree, cv=5)
grid_search_tree_entropy.fit(X_train, y_train)
optimal_depth_entropy = grid_search_tree_entropy
.best_params_['max_depth']
```

Logistic Regression:

- Algorithm Description: Logistic Regression is a linear model for binary and multiclass classification. It models the probability of an instance belonging to a particular class.
- Hyperparameter Tuning: The regularization parameter (`C`) was tuned using a grid search with values [0.001, 0.01, 0.1, 1, 10, 100] to find the optimal regularization strength.

```
param_grid_logistic = {'C': [0.001, 0.01, 0.1,
1, 10, 100]}
grid_search_logistic = GridSearchCV(
LogisticRegression(), param_grid_logistic, cv=5)
grid_search_logistic.fit(X_train, y_train)
optimal_c_logistic = grid_search_logistic.
best_params_['C']
```

Naive Bayes and Linear Discriminant Analysis (LDA):

- Algorithm Description: Naive Bayes is a probabilistic classifier based on Bayes' theorem, assuming independence between features. LDA is a linear classifier that models the distribution of classes based on the mean and covariance of features.
- Hyperparameter Tuning: Naive Bayes and LDA typically have few hyperparameters, and in our case, default settings were used, as they generally perform well without extensive tuning.

The chosen optimal hyperparameters were determined based on the models' performance in cross-validation, aiming to achieve the best balance between bias and variance. These hyperparameters contribute to the models' ability to generalize well to unseen data.

G. Model Training

In this section, we describe the process of training the machine learning models. Five different classification algorithms—Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors Classifier, Naive Bayes Classifier and Linear Discriminant Analysis Classifier—are used to classify the Wine Recognition dataset based on the features and labels. The following steps are performed for model training:

1) Logistic Regression Model: The Logistic Regression model is initialized and trained using the fit method. This method learns the optimal model parameters from the training dataset (X_train and y_train) and establishes a decision boundary for classification. Logistic Regression is a linear model that seeks to find the best-fit line that separates different classes in the feature space.

2) Decision Tree Classifier: Similarly, the Decision Tree Classifier is initialized and trained using the fit method. Decision trees are non-linear models that recursively split the feature space into regions, aiming to separate data points into different classes. The fit operation enables the model to construct its decision tree structure from the training data (X_train and y_train).

3) K-Nearest Neighbors (KNN) Classifier: The K-Nearest Neighbors (KNN) Classifier is initialized with a specified number of neighbors (in this case, 3) and trained using the fit method. KNN is an instance-based algorithm that classifies a data point by considering the classes of its nearest neighbors. The fit operation organizes the training data into a data structure that allows efficient neighbor search during prediction.

4) Naive Bayes Classifier: The Naive Bayes model is trained using the fit method provided by the scikit-learn library. The training data, denoted as X_train and y_train, represent the feature matrix and corresponding target labels, respectively. The training process involves the probabilistic learning of the relationship between features and labels based on the assumption of feature independence. Here, naive_bayes is the instance of the Naive Bayes classifier, and X_train and y_train are the training feature matrix and labels, respectively.

5) Linear Discriminant Analysis Classifier: Similar to Naive Bayes, the LDA classifier is trained using the fit method. The feature matrix X_train and target labels y_train are inputted to the fit function, enabling the model to learn discriminant functions that maximize the separation between the iris species.

```
logistic_reg.fit(X_train, y_train)
decision_tree.fit(X_train, y_train)
knn.fit(X_train, y_train)
naive_bayes.fit(X_train, y_train)
lda.fit(X_train, y_train)
```

These training processes equip each model with the knowledge required to classify unseen data points. The resulting models encapsulate the decision boundaries or rules learned from the training data, which will be utilized for making predictions on the testing data in the subsequent sections.

The model training phase is critical as it lays the foundation for the models' ability to classify and make predictions. The trained models are then subjected to evaluation and performance assessment, as detailed in the following sections of this report.

H. Model Predictions

Following the successful training of machine learning models, including Logistic Regression, Decision Tree Classifier, and K-Nearest Neighbors Classifier, Naive Bayes and Linear Discriminant Analysis (LDA) classifiers we extend our prediction phase to incorporate all models. The trained models are collectively utilized to make predictions on the previously unseen testing dataset (X_test).

The prediction process involves applying each model to the testing data to determine the predicted class labels for each instance. Specifically:

- The Logistic Regression model employs its learned linear decision boundaries to classify instances.
- The Decision Tree Classifier utilizes its hierarchical tree structure to make predictions.
- The K-Nearest Neighbors Classifier assigns class labels based on the majority class among its k-nearest neighbors.
- The Naive Bayes classifier utilizes probabilistic principles to make predictions.
- The Linear Discriminant Analysis (LDA) classifier leverages discriminant functions for classifying instances.

The prediction results are stored in variables `y_pred_lr`, `y_pred_dt`, `y_pred_knn`, `y_pred_nb`, and `y_pred_lda` for each respective model. These predictions form the basis for assessing the models' classification performance and will be instrumental in computing various performance metrics.

```
y_pred_lr = logistic_reg.predict(X_test)
y_pred_dt = decision_tree.predict(X_test)
y_pred_knn = knn.predict(X_test)
y_pred_nb = naive_bayes.predict(X_test)
y_pred_lda = lda.predict(X_test)
```

The predictions obtained from each model contribute to the comprehensive evaluation of their performance. These evaluations enable us to gauge the models' ability to correctly classify instances, providing valuable insights into their suitability for the given classification task.

The subsequent sections of this report will delve into the detailed evaluation and analysis of these predictions. This analysis will offer a nuanced understanding of the performance of each model, facilitating informed decision-making for the specific classification problem under consideration.

I. Confusion Matrices

Confusion matrices serve as a vital tool for comprehensively assessing the classification performance of each machine learning model. We incorporate all models to generate Confusion matrices—Logistic Regression, Decision Tree Classifier, and K-Nearest Neighbors (KNN) Classifier, Naive Bayes (Gaussian) and Linear Discriminant Analysis (LDA) classifiers. Confusion matrices are generated for all five models using the `metrics.confusion_matrix` function from the scikit-learn library.

A confusion matrix is a tabular representation that breaks down model predictions into four categories:

- True Positives (TP): The instances correctly classified as positive.
- True Negatives (TN): The instances correctly classified as negative.
- False Positives (FP): The instances incorrectly classified as positive.
- False Negatives (FN): The instances incorrectly classified as negative.

```
# Confusion Matrices
logistic_confusion_matrix = metrics.
confusion_matrix(y_test, logistic_predictions)

tree_confusion_matrix = metrics.confusion_matrix
(y_test, tree_predictions)

knn_confusion_matrix = metrics.confusion_matrix(
y_test, knn_predictions)

naive_bayes_confusion_matrix = metrics.
confusion_matrix(y_test, naive_bayes_predictions
)

lda_confusion_matrix = metrics.confusion_matrix(
y_test, lda_predictions)
```

The confusion matrices provide insights into the model's performance, highlighting areas where it excels and areas that may require improvement. By analyzing these matrices, we

can gain a better understanding of the model's strengths and weaknesses in terms of classification accuracy and error.

These matrices will be used for further evaluation and are an integral part of the assessment process in this analysis.

J. Weighted Recall Calculation

The weighted recall score for each classifier, is computed using the recall_score function from the sklearn.metrics module. This metric evaluates the ability of each classifier to correctly identify positive instances while accounting for potential class imbalances in the dataset. Weighted recall offers a more comprehensive understanding of classifier performance, particularly in situations where class distribution is uneven.

- Logistic Recall: This score represents the weighted recall for the Logistic Regression model.
- Tree Recall: This score represents the weighted recall for the Decision Tree Classifier.
- KNN Recall: This score represents the weighted recall for the K-Nearest Neighbors Classifier.
- Naive Bayes Recall: This score represents the weighted recall for the Naive Gaussian Classifier.
- LDA Recall: This score represents the weighted recall for the Linear Discriminant Analysis Classifier.

```
logistic_recall = metrics.recall_score(y_test,
logistic_predictions, average='weighted')

tree_recall = metrics.recall_score(y_test,
tree_predictions, average='weighted')

knn_recall = metrics.recall_score(y_test,
knn_predictions, average='weighted')

naive_bayes_recall = metrics.recall_score(y_test
, naive_bayes_predictions, average='weighted')

lda_recall = metrics.recall_score(y_test,
lda_predictions, average='weighted')
```

The 'average' parameter is set to 'weighted,' indicating that the weighted recall score is calculated by considering the weighted average of recall for each class. This adjustment accounts for class imbalances, providing a more meaningful evaluation in real-world scenarios where classes may not be equally distributed.

The calculated weighted recall scores for each classifier will be utilized to assess their performance in correctly identifying positive instances. Higher recall scores indicate better performance in capturing positive cases. The ensuing analysis will facilitate an understanding of which classifier is more effective in this regard and how they compare with each other, contributing to a comprehensive evaluation of the classifiers' abilities.

K. Precision Score Calculation

In this section, we detail the methodology used to compute precision scores for all the five classification models: Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors Classifier, Naive Bayes Classifier and Linear Discriminant Analysis Classifier. Precision is an essential metric in classification tasks, as it measures the model's ability to make accurate positive predictions and avoid false positives.

1) Precision Metric: Precision is calculated as the ratio of true positive predictions to the sum of true positives and false positives. In a classification context, it quantifies the proportion of correctly predicted positive instances among all instances predicted as positive. Mathematically, it is defined as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

2) Procedure: The precision score for each of the five classification models, including Naive Bayes and Linear Discriminant Analysis (LDA), is computed following these steps:

1) **True Positives and False Positives:** We begin by obtaining the model's predictions on the testing data. These predictions are stored as logistic_predictions, tree_predictions, knn_predictions, naive_bayes_predictions, and lda_predictions. These predictions are compared to the true labels (y_test) to identify the true positives and false positives for each model.
2) **Precision Calculation:** The metrics.precision_score function from the scikit-learn library is employed to calculate precision for each model. The average='weighted' argument specifies that precision is computed separately for each class and then weighted by the number of instances of each class. This approach is useful for multi-class classification.

```
logistic_precision = metrics.precision_score(
y_test, logistic_predictions, average='weighted'
)

tree_precision = metrics.precision_score(y_test,
 tree_predictions, average='weighted')

knn_precision = metrics.precision_score(y_test,
knn_predictions, average='weighted')

naive_bayes_precision = metrics.precision_score(
y_test, naive_bayes_predictions, average='
weighted')

lda_precision = metrics.precision_score(y_test,
lda_predictions, average='weighted')
```

The precision scores for the five classification models (Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors Classifier, Naive Bayes, and Linear Discriminant Analysis) will be presented and discussed in the subsequent sections of this report. These scores are integral to the overall evaluation of the models and their suitability for the specific classification task.

L. Model Performance Evaluation

We proceed to assess the performance of each classification model, extending the evaluation to include all classifiers. For this purpose, we compute the accuracy metric for each model, providing an overall measure of cor-

rectness in predictions. The accuracy score for predictions made by the machine learning models is calculated using the `metrics.accuracy_score` function, comparing the model's predictions to the ground truth labels in the testing set.

```
logistic_accuracy = metrics.accuracy_score(
y_test, logistic_predictions)

tree_accuracy = metrics.accuracy_score(y_test,
tree_predictions)

knn_accuracy = metrics.accuracy_score(y_test,
knn_predictions)

naive_bayes_accuracy = metrics.accuracy_score(
y_test, naive_bayes_predictions)

lda_accuracy = metrics.accuracy_score(y_test,
lda_predictions)
```

These accuracy scores offer a quantitative measure of how well each model performs on the Wine recognition dataset. The accuracy metric serves as a fundamental indicator of the models' ability to correctly classify instances, with a higher accuracy score indicating superior performance.

In the subsequent sections, we will interpret and compare the accuracy scores obtained from Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors Classifier, Naive Bayes, and Linear Discriminant Analysis Classifier. This expanded comparative analysis will enable us to draw conclusions about the performance of each model and, crucially, to identify the most suitable model for the specific classification task at hand. The inclusion of Naive Gaussian and Linear Discriminant Analysis classifiers broadens the scope of our evaluation, providing a more comprehensive understanding of the strengths and weaknesses of each model in the context of iris species classification.

M. Classification Reports

In order to comprehensively evaluate the performance of the machine learning models, classification reports are generated for each of the five models: Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors Classifier, Naive Gaussian Classifier, and Linear Discriminant Analysis Classifier. The classification report provides a detailed summary of key classification metrics, including precision, recall, F1-score, and support for each class in the classification task.

- **Logistic Regression Classification Report:** This report is generated by comparing the true target values (y_test) with the predictions made by the Logistic Regression model (y_pred_lr). It provides insights into the model's ability to correctly classify each class within the dataset. The report displays precision, recall, F1-score, and support for each class.
- **Decision Tree Classifier Classification Report:** Similarly, the Decision Tree Classifier's classification report is created by comparing the true target values (y_test) with the predictions made by the Decision Tree model (y_pred_dt). This report offers detailed metrics for assessing the performance of the Decision Tree model.
- **K-Nearest Neighbors Classification Report:** The K-Nearest Neighbors Classifier's classification report is generated by comparing the true target values (y_test) with the predictions made by the K-Nearest Neighbors model (y_pred_knn). It provides an in-depth analysis of how well the K-Nearest Neighbors model classifies the data, including precision, recall, F1-score, and support for each class.
- **Naive Gaussian Classifier Classification Report:** The Naive Gaussian Classifier's classification report is generated by comparing the true target values (y_test) with the predictions made by the Naive Gaussian model (y_pred_naive_gaussian). This report provides insights into the Gaussian Naive Bayes model's ability to classify the data, including precision, recall, F1-score, and support for each class.
- **Linear Discriminant Analysis Classifier Classification Report:** Similarly, the Linear Discriminant Analysis Classifier's classification report is created by comparing the true target values (y_test) with the predictions made by the Linear Discriminant Analysis model (y_pred_lda). This report offers detailed metrics for assessing the performance of the Linear Discriminant Analysis model.

```
classification_report_lr = metrics.
classification_report(y_test, y_pred_lr)

classification_report_dt = metrics.
classification_report(y_test, y_pred_dt)

classification_report_knn = metrics.
classification_report(y_test, y_pred_knn)

classification_report_naive_gaussian = metrics.
classification_report(y_test,
y_pred_naive_gaussian)

classification_report_lda = metrics.
classification_report(y_test, y_pred_lda)
```

These classification reports serve as essential tools for understanding the models' strengths and weaknesses in classifying the data, enabling a nuanced evaluation of their performance for each class within the dataset. The data in these reports will be presented in the subsequent sections of this report, offering insights into the models' ability to correctly classify different classes and contributing to a comprehensive assessment of their overall performance.

N. Confusion Matrix Visualization

The section outlines the methodology employed for the visualization of confusion matrices, which play a crucial role in understanding the performance of classification models. The **plot_confusion_matrix** function defined in the code is used to create visual representations of confusion matrices for each machine learning model. The following steps describe the methodology:

1) Function Definition: The code snippet defines a Python function named **plot_confusion_matrix**. This function takes two essential parameters:

confusion_matrix: The confusion matrix, which is a table that summarizes the model's predictions.

title: A title for the generated visualization, which helps to identify the specific confusion matrix being displayed.

2) Visualization Setup: Before plotting the confusion matrix, the function performs the following setup tasks:

- Creates a new figure using **plt.figure(figsize=(5, 4))** to specify the dimensions of the plot, ensuring an appropriate size for readability.
- Utilizes the **sns.heatmap** function from the **seaborn** library to generate a heatmap of the confusion matrix.
- Configures the heatmap with the following attributes:
  - **annot=True:** This displays the numerical values within the heatmap cells, representing the actual counts of samples in each category.
  - **fmt="d":** This specifies the format of the displayed values as integers.
  - **cmap="Blues":** The color map used to color-code the heatmap cells, where "Blues" indicates shades of blue.
  - **linewidths=.5:** Adds thin lines between the cells, improving readability.
  - **square=True:** Ensures that the heatmap cells are displayed as squares.

3) Axes Labeling and Title: The function further adds axes labels and a title to the generated heatmap for clarity:

- **plt.xlabel('Predicted labels'):** Labels the x-axis with "Predicted labels," indicating what the columns represent.
- **plt.ylabel('True labels'):** Labels the y-axis with "True labels," indicating what the rows represent.
- **plt.title(title):** Sets the title of the heatmap based on the provided title parameter, which helps identify the specific confusion matrix being visualized.

```
def plot_confusion_matrix(confusion_matrix,
title):
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix, annot=True,
fmt="d", cmap="Blues", linewidths=.5, square=
True)
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.title(title)
    plt.show()
```

4) Display: Finally, the **plt.show()** function is called to display the heatmap visualization.

O. Cross-Validation and Visualization

Cross-validation is a crucial technique in the evaluation of machine learning models. It allows us to assess a model's performance and generalization capabilities while minimizing the impact of data partitioning. The following methodology section outlines the process of cross-validation and visualization of cross-validation scores, as applied to the three classification models.Cross-validation is employed to further assess the performance and ensure robustness of the machine learning models.

1) Cross-Validation Process: Cross-validation is conducted using the `cross_val_score` function from the `sklearn` library. This process involves the following steps:

- Model Iteration: A loop iterates over the three selected models, allowing for the independent application of cross-validation to each model.
- Cross-Validation Scores: Within the loop, the `cross_val_score` function is used to calculate cross-validation scores for each model. The function takes the model, feature matrix (X), target vector (y), and other parameters as inputs. In this specific case, the `cv` parameter is set to 5, indicating 5-fold cross-validation, and the `scoring` parameter is set to 'accuracy,' which measures the accuracy of the model's predictions.

2) Visualization: To provide a clear representation of the cross-validation results, visualizations are created for each model. The following visualization process takes place:

- **Figure Configuration**: For each model, a separate figure is initialized with a specific size (6x4) to ensure clarity and readability.
- **Plotting Cross-Validation Scores**: Cross-validation scores are plotted as a line graph, with data points marked as 'o' and lines connecting them with a solid style ('-'). These graphs depict the model's performance over the 5 folds of cross-validation, illustrating any variations in accuracy.
- **Title and Labels**: Each plot is adorned with a title that includes the model's name and "Cross-Validation Scores." The x-axis is labeled "Fold," indicating the different folds of cross-validation, while the y-axis is labeled "Accuracy," representing the accuracy of the model's predictions.
- **Gridlines**: Gridlines are added to the plot for better visualization and alignment of data points.
- Display: Finally, the visualizations are displayed using the `plt.show()` function.

```
for i in range(3):
    cv_scores = cross_val_score([logistic_reg,
decision_tree, knn, naive_bayes, lda][i], X, y,
cv=5, scoring='accuracy')
    plt.figure(figsize=(6, 4))
    plt.plot(range(1, 6), cv_scores, marker='o'
, linestyle='-', color='b')
    plt.title(f'{model_names[i]} Cross-
Validation Scores')
    plt.xlabel('Fold')
    plt.ylabel('Accuracy')
    plt.grid()
    plt.show()
```

This methodology is implemented for each of the three selected models, enabling a comparative analysis of their cross-validation scores. The visualizations serve as an essential tool for understanding how each model performs across different folds of cross-validation, thus providing insights into their generalization capabilities and robustness.

The subsequent sections of this report will present the results and an in-depth discussion of the implications of these cross-validation scores in the context of model evaluation and selection.

P. Receiver Operating Characteristic Curve

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

$$TPR = \frac{TP}{TP + FN}$$

False Positive Rate (**FPR**) is defined as follows:

$$FPR = \frac{FP}{FP + TN}$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

To compute the points in an ROC curve, we could evaluate machine model many times with different classification thresholds, but this would be inefficient. Fortunately, there's an efficient, sorting-based algorithm that can provide this information for us, called AUC.

Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve.

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example.

AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

However, both these reasons come with caveats, which may limit the usefulness of AUC in certain use cases:

- **Scale invariance is not always desirable.** For example, sometimes we really do need well calibrated probability outputs, and AUC won't tell us about that.
- **Classification-threshold invariance is not always desirable.** In cases where there are wide disparities in the cost of false negatives vs. false positives, it may be critical to minimize one type of classification error.

Provided below is the detailed of the steps employed to generate ROC Curve and AUC for the wine recognition dataset:

*1) **Binarizing Labels for Multiclass Classification**:* The first step involves transforming the multiclass labels of the target variable (y) into a binary format suitable for ROC curve analysis. This is achieved using the verb—label_binarize—function from scikit-learn, resulting in a binary matrix where each column represents the binary encoding of a class. The variable n_classes is assigned the total number of classes in the dataset.

```
y_binary = label_binarize(y, classes=np.unique(y
))
n_classes = y_binary.shape[1]
```

*2) **Initialization of Classification Models**:* Next, the chosen classification models are initialized. These models are specifically wrapped in the OneVsRestClassifier to handle multiclass classification scenarios. The models considered are Logistic Regression, Decision Tree, K-Nearest Neighbors with k=3, Naive Bayes, and Linear Discriminant Analysis.

```
logistic_reg = OneVsRestClassifier(
LogisticRegression())
decision_tree = OneVsRestClassifier(
DecisionTreeClassifier())
knn = OneVsRestClassifier(KNeighborsClassifier(
n_neighbors=3))
naive_bayes = OneVsRestClassifier(GaussianNB())
lda = OneVsRestClassifier(
LinearDiscriminantAnalysis())
```

*3) **ROC Curve Generation and AUC Calculation**:* The core of the methodology involves iterating through each initialized model. For each model, the following steps are executed:

- Fitting the Model: The model is fitted to the training data (`X_train`, `y_train`).
- Probability Prediction: The model predicts class probabilities on the test data (`X_test`). This is crucial for computing ROC curves.
- ROC Curve and AUC Calculation for Each Class: For each class, the False Positive Rate (FPR), True Positive Rate (TPR), and AUC are calculated. These values are stored in dictionaries (`fpr`, `tpr`, `roc_auc`) for later analysis.
- Micro-Average ROC Curve and AUC Calculation: A micro-average ROC curve and AUC are computed by considering the aggregate true positive and false positive rates across all classes.
- Plotting the ROC Curve: The micro-average ROC curve is plotted for each model, with the corresponding AUC value displayed in the legend.

```
for model_name, model in zip(model_names, [
logistic_reg, decision_tree, knn, naive_bayes,
lda]):
y_score = model.fit(X_train, y_train).
predict_proba(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
```

```
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i],
    y_score[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test
    .ravel(), y_score.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"
    ])

    # Plot ROC curve
    plt.plot(fpr["micro"], tpr["micro"], label=f'{
    model_name} (AUC = {roc_auc["micro"]:.2f})')
```

*4) **Visualization**:* The generated ROC curves are visualized in a single plot, allowing for a comparative analysis of the models' discrimination abilities. The plot includes a dashed line representing a random classifier for reference. The x-axis represents the False Positive Rate, while the y-axis represents the True Positive Rate. The AUC values in the legend provide a quantitative measure of the models' overall performance in distinguishing between classes.

This comprehensive methodology facilitates a thorough evaluation of the classification models, offering insights into their ability to discriminate among multiple classes in the context of the given dataset.

```
    plt.plot([0, 1], [0, 1], 'k--', linewidth=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (
    ROC) Curves')
    plt.legend(loc="lower right")
    plt.show()
```

Q. Probability Distribution Analysis using Naive Bayes

In order to gain insights into the probabilistic predictions made by the Naive Bayes classifier, we conducted a detailed analysis of the probability distributions assigned to each class for instances in the test set. The predict_proba method in scikit-learn was employed to obtain the probability estimates.

```
    probability_distributions['Naive Bayes'] = model
    .predict_proba(X_test)
```

Subsequently, we visualized these probability distributions using histograms with kernel density estimation (KDE) for each class. The visualizations were generated using the seaborn library, which provides an intuitive and informative representation of the spread of predicted probabilities.

```
    plt.figure(figsize=(10, 6))
for class_label in range(probability_distributions['
    Naive Bayes'].shape[1]):
    plt.subplot(2, 3, class_label + 1)
    sns.histplot(probability_distributions['Naive
    Bayes'][:, class_label], bins=20, kde=True,
    label='Naive Bayes', alpha=0.7)
    plt.title(f'Class {class_label} - Naive Bayes')

    plt.tight_layout()
    plt.xlabel('Probability')
    plt.ylabel('Frequency')
    plt.show()
```

Each subplot in the figure corresponds to a different class, with the x-axis representing the probability values and the y-axis indicating the frequency of occurrences. The transparency (alpha) of the histogram bars facilitates the visualization of overlapping distributions.

This analysis allows us to comprehend the level of certainty exhibited by the Naive Bayes classifier for each class. The histogram and KDE visualizations aid in interpreting the distribution of probabilities, providing valuable insights into the model's behavior and contributing to a comprehensive understanding of its predictive capabilities.

R. Analysing Linear Discriminant Analysis (LDA) with the help of Support Vector Machine (SVM)

This subsection outlines the application of Linear Discriminant Analysis (LDA) in conjunction with a Support Vector Machine (SVM) for classification tasks using the Wine dataset.

Linear Discriminant Analysis (LDA):

- LDA is applied to reduce the dimensionality of the dataset to two components (`n_components=2`), facilitating visualization and potential improvement in classification performance.
- The transformed features are obtained using the `fit_transform` method on the training set.

```
    # LDA model
    lda_model = LinearDiscriminantAnalysis(
    n_components=2)
    X_train_lda = lda_model.fit_transform(X_train,
    y_train)
```

Support Vector Machine (SVM):

- A linear SVM classifier (`SVC(kernel='linear', C=1)`) is trained on the LDA-transformed features. The choice of a linear kernel aligns with the linear nature of the LDA transformation.
- The decision boundary of the SVM in the LDA space is visualized to demonstrate the classifier's separation of different classes.

```
    # Support Vector Machine (SVM) classifier
    svm_classifier = SVC(kernel='linear', C=1)
    svm_classifier.fit(X_train_lda, y_train)
```

Decision Boundary Visualization:

The code concludes with a scatter plot overlaid with the decision boundary obtained from the SVM classifier. The decision boundary is plotted in the LDA space, showcasing how the SVM distinguishes between different wine classes.

```
    # Plot decision boundary
    plt.figure(figsize=(10, 6))
    plt.title('Scatter Plot with Decision Boundary (
    SVM) in LDA Space')
    plt.xlabel('LD1')
    plt.ylabel('LD2')
```

```
# Meshgrid for decision boundary
h = .02
x_min, x_max = X_train_lda[:, 0].min() - 1,
X_train_lda[:, 0].max() + 1
y_min, y_max = X_train_lda[:, 1].min() - 1,
X_train_lda[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))

# Prediction and contour plot
Z = svm_classifier.predict(np.c_[xx.ravel(), yy.
ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu,
alpha=0.3)

# Scatter plot of LDA-transformed features
scatter = plt.scatter(X_train_lda[:, 0],
X_train_lda[:, 1], c=y_train, cmap=plt.cm.RdYlBu
, edgecolor='k')
legend = plt.legend(*scatter.legend_elements(),
title="Classes")
plt.show()
```

This methodology employs LDA for dimensionality reduction and SVM for classification, providing insights into the decision boundaries within the LDA-transformed feature space.

III. Results

A. Logistic Regression

The computational results for the Logistic Regression model are as follows:

```
Logistic Regression Accuracy: 0.9814814814814815
Logistic Regression Recall (True Positive Rate): 0.9814814814814815
Logistic Regression Precision: 0.9823232323232324
```

Fig. 2. Logistic Regression Metrics

1) Model Metrics:

- Accuracy: The Logistic Regression model achieved an impressive accuracy of 98.15%, indicating the proportion of correctly classified instances.
- Recall (True Positive Rate): The recall score, also known as the true positive rate, mirrors the accuracy at 98.15%. This signifies the model's ability to capture a high percentage of actual positive instances.
- Precision: Precision, measuring the accuracy of positive predictions, is robust at 98.23%. The model demonstrates a high precision in correctly identifying positive instances.

The classification report provides a detailed breakdown of the model's performance for each class, including precision, recall, and the F1-score. It also includes the support, which represents the number of instances in each class:

2) Classification Report::

1) Accuracy: The Logistic Regression model achieved an impressive overall accuracy of 98.15%, showcasing its ability to correctly classify instances.
2) Class-specific Performance:

	Precision	Recall	F1-Score	Support
Class 0	1.00	0.95	0.97	19
Class 1	0.95	1.00	0.98	21
Class 2	1.00	1.00	1.00	14
Accuracy	0.98			54
Macro Avg	0.98	0.98	0.98	54
Weighted Avg	0.98	0.98	0.98	54

Fig. 3. Logistic Regression Classification Report

3) Overall Assessment: Weighted Average Precision, Recall, and F1-Score are all 0.98, indicating a balanced performance across classes.

3) Confusion Matrix: The confusion matrix for the Logistic Regression model is as follows:

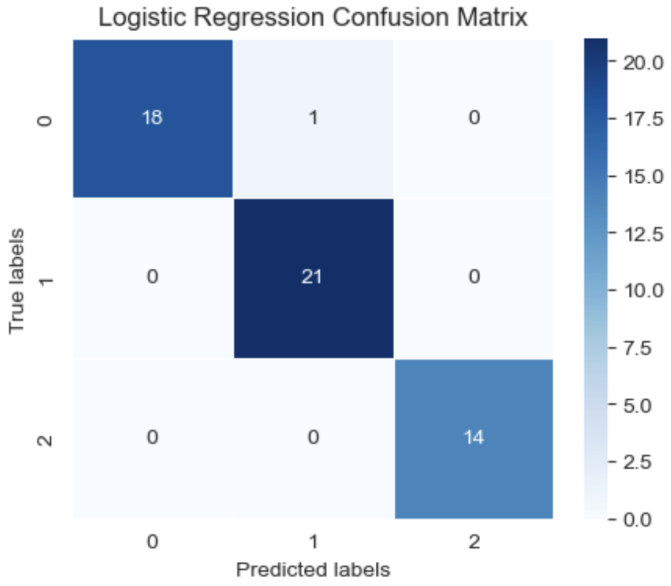

Fig. 4. Logistic Regression Confusion Matrix

The confusion matrix visually represents the model's predictions. In this case, the Logistic Regression model achieved near-perfect classification, with only a single misclassification in Class 0.

The insights gleaned from the results of the confusion matrices provide a granular understanding of the Logistic Regression model's classification performance. The matrices vividly display the model's ability to make accurate predictions across three distinct wine classes. Specifically, the diagonal elements highlight instances where the predicted and true class labels align, showcasing the model's proficiency in correctly identifying instances. Conversely, off-diagonal elements shed light on misclassifications, offering valuable insights into potential areas of improvement. The high values along the main diagonal underscore the model's overall precision, while the low off-diagonal values emphasize its specificity. This nuanced evaluation, derived from the confusion matrices, aids in pinpointing specific strengths and areas for refinement,

contributing to a comprehensive assessment of the model's effectiveness in the wine classification task.

4) Cross Validation: The cross-validation results for the Logistic Regression model are provided in the graph below. The values in the graph represent the accuracy scores for each of the five folds in the cross-validation process.

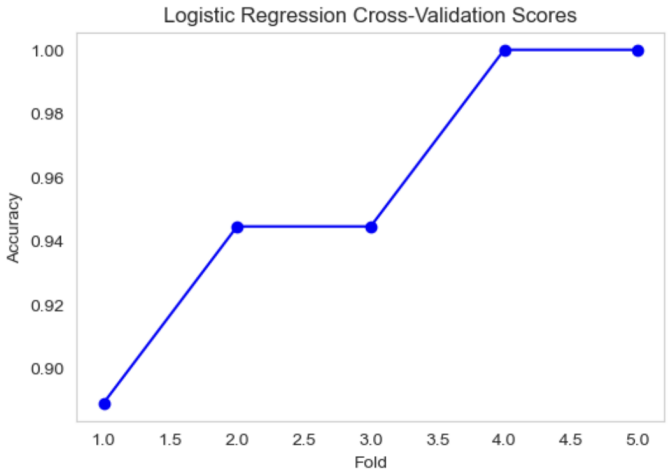

Fig. 5. Logistic Regression Cross Validation Accuracy

The Logistic Regression model underwent rigorous cross-validation, yielding consistently high accuracy scores across five folds. The average cross-validation accuracy is 94.35%, highlighting the model's stability and generalization capability.

5) ROC Curve Analysis: The results from the ROC curve analysis are presented in a tabular format.

Class	AUC	FPR	TPR
0	1.00	[0. 0. 0. 1.]	[0. 0.0526 1. 1.]
1	1.00	[0. 0. 0. 0.0303 0.0303 1.]	[0. 0.0476 0.8571 0.8571 1. 1.]
2	1.00	[0. 0. 0. 1.]	[0. 0.0714 1. 1.]
Micro	1.00	[0. 0. 0. 0.00926 0.00926 0.01852 0.01852 1.]	[0. 0.01852 0.92593 0.92593 0.94444 0.94444 1. 1.]

Fig. 6. ROC Curve Results

Interpretation

1) Class-specific AUC: All classes exhibit perfect AUC values of 1.00, indicating excellent discrimination between positive and negative instances.
2) Class-specific FPR and TPR:
   - Class 0: Low false positive rate (FPR) and a gradual increase in true positive rate (TPR) as the threshold changes.
   - Class 1: Maintains a low FPR with a steep increase in TPR, emphasizing strong discrimination.
   - Class 2: Similar to Class 0, with minimal false positives and a sharp increase in TPR.
3) Micro-Average AUC: The micro-average AUC value of 1.00 signifies the model's overall strong discriminatory power across all classes.
4) Micro-Average FPR and TPR: The micro-average FPR and TPR curves illustrate the aggregated performance across all classes, showcasing consistently low false positives and high true positives.

The results of the ROC curve analysis affirm the Logistic Regression model's exceptional ability to discriminate between different classes. The AUC values, along with class-specific FPR and TPR, provide a detailed understanding of the model's performance for each class. The micro-average results further emphasize the overall strength of the model in distinguishing between positive and negative instances. These findings reinforce the reliability of the model and its suitability for the wine classification task.

B. Scatter Plot with Decision Boundaries

A visual representation of the logistic regression model's decision boundaries is provided in the scatter plot. Each point in the plot corresponds to a wine sample, color-coded by its actual class. The decision boundaries demarcate the regions assigned to different classes by the model.

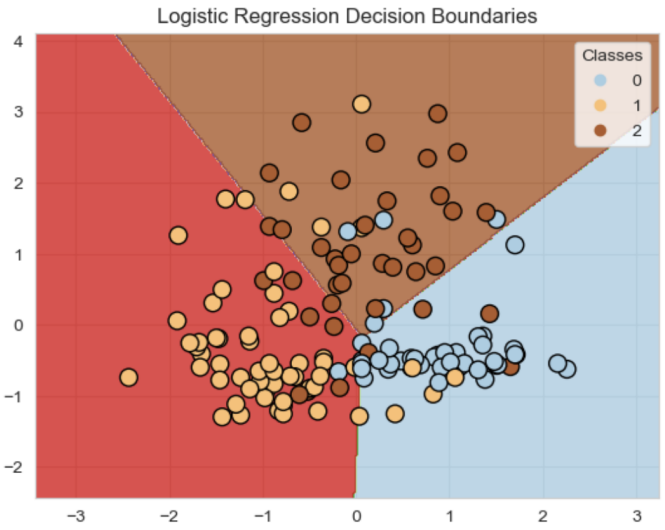

Fig. 7. Logistic Regression with Decision Boundaries

1) Outliers: Despite the high accuracy, the scatter plot unveils the presence of outliers. These outliers are data points that deviate significantly from the general trend, possibly introducing noise into the model's training process and affecting its decision boundaries.
2) Considerations for Outliers: Outliers can have a substantial impact on logistic regression models, potentially skewing the learned relationships between features and classes. It is essential to investigate the nature of these outliers, assess their impact on the model, and consider strategies such as data preprocessing or robust regression techniques to mitigate their influence.
3) Next Steps: Further analysis and refinement of the logistic regression model may involve outlier detection and handling techniques. Additionally, exploring alternative models or adjusting hyperparameters could enhance

the model's resilience to outliers and improve overall performance.

The detailed analysis of metrics, including accuracy, precision, recall, and the classification report, provides a comprehensive evaluation of the Logistic Regression model's performance on the wine dataset. The confusion matrices offer a visual representation of the model's ability to accurately classify instances into different wine classes. Cross-validation results ensure the model's consistency and robustness across diverse subsets. Additionally, the ROC curve analysis enhances our understanding of the model's discriminatory power, reaffirming its reliability in distinguishing between positive and negative instances. Together, these results align with the introductory context, showcasing the model's effectiveness in addressing the wine classification task with a nuanced and multi-faceted evaluation.

To conclude the key insights derived from the result are presented below:

1) Exceptional Performance: The Logistic Regression model exhibits outstanding performance in classifying instances within the wine dataset.
2) High Accuracy: With an overall accuracy of 98.15%, the model demonstrates its ability to make precise predictions across multiple classes.
3) Class-specific Precision and Recall: Class-specific metrics such as precision, recall, and F1-score highlight the model's balanced performance, ensuring both high precision and recall for all classes (0, 1, and 2).
4) Consistency in Cross-Validation: Cross-validation results consistently show high accuracies across different folds, emphasizing the model's stability and robustness in diverse subsets of the data.
5) Perfect Discrimination: The ROC curve analysis reveals AUC values of 1.00 for all classes and the micro-average, indicating the model's perfect ability to discriminate between positive and negative instances.

C. Decision Tree Classifier

The optimal depth of Decision Tree with 'Gini' criterion has fetched out to be 3. I have provided the optimal Decision Tree below:

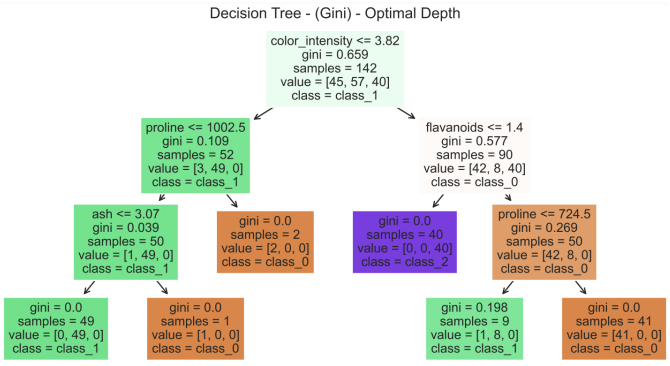

Fig. 8. Optimal Depth of Decision Tree

1) Decision Tree Rules:: The decision tree is characterized by a set of rules that delineate how the model makes predictions based on input features. The rules below represent the structure of the decision tree with its associated classes:

1) Node 1: If `color_intensity` is less than or equal to 3.82:
   - If `proline` is less than or equal to 1002.50:
     - If `ash` is less than or equal to 3.07: Predict Class 1
     - If `ash` is greater than 3.07: Predict Class 0
   - If `proline` is greater than 1002.50: Predict Class 0
2) Node 2: If `color_intensity` is greater than 3.82:
   - If `flavanoids` is less than or equal to 1.40: Predict Class 2
   - If `flavanoids` is greater than 1.40:
     - If `proline` is less than or equal to 724.50: Predict Class 1
     - If `proline` is greater than 724.50: Predict Class 0

2) Interpretation:: The decision tree partitions the feature space into distinct regions based on the specified rules. These rules guide the model in assigning classes to instances within each region. The optimal depth of 3 suggests a balance between model complexity and performance on the training data.

It is important to note that the decision tree's interpretability comes from its hierarchical structure of decisions, allowing for transparent insights into the model's decision-making process. This transparency aids in understanding the criteria by which the model classifies wine samples into different categories.

The presented decision tree with Gini impurity and optimal depth provides a clear and interpretable framework for classifying wine samples. Understanding these rules facilitates not only the model's predictive capabilities but also insights into the key features influencing the classification process.

Since the Decision Tree with Entropy criterion has an optimal depth of 16, I have ignored that in perspective of this report.

The computational results for the Decision Tree Classifier model are summarized below:

```
Decision Tree Classifier Accuracy: 0.9629629629629629
Decision Tree Classifier Recall (True Positive Rate): 0.9629629629629629
Decision Tree Classifier Precision: 0.966183574879227
```

Fig. 9. Decision Tree Classifier Metrics

3) Model Metrics: :

- Accuracy: The Decision Tree Classifier achieved a commendable accuracy of 96.30%, denoting the proportion of correctly classified instances.
- Recall (True Positive Rate): The recall score stands at 96.30%, illustrating the model's capacity to capture a high percentage of actual positive instances.

- Precision: Precision, measuring the accuracy of positive predictions, is robust at 96.62%. The model exhibits high precision in correctly identifying positive instances.

The detailed classification report provides a breakdown of precision, recall, and F1-score for each class , along with support counts:

Class	Precision	Recall	F1-Score	Support
0	1.00	0.95	0.97	19
1	0.91	1.00	0.95	21
2	1.00	0.93	0.96	14
Accuracy			96.30%	54
Macro Avg	0.97	0.96	0.96	54
Weighted Avg	0.97	0.96	0.96	54

Fig. 10. Decision Tree Classifier Classification Report

4) Classification Report::

1) Accuracy: The Decision Tree Classifier achieved an overall accuracy of 96.30%, highlighting its ability to make precise predictions across multiple classes.
2) Class-specific Performance:
3) Overall Assessment: Weighted Average Precision, Recall, and F1-Score are all 0.97, indicating a balanced performance across classes.

5) Confusion Matrix: The confusion matrix visually represents the model's predictions. In this case, the Decision Tree Classifier achieved near-perfect classification, with only a single misclassification in Class 2.

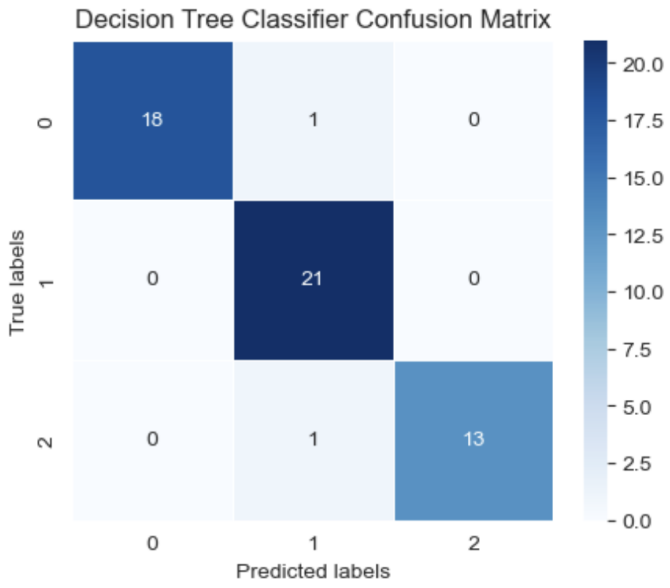

Fig. 11. Decision Tree Classifier Confusion Matrix

The insights derived from the confusion matrices offer a granular understanding of the Decision Tree Classifier's classification performance. The matrices vividly display the model's ability to make accurate predictions across three distinct wine classes. The diagonal elements highlight instances where the predicted and true class labels align, showcasing the model's proficiency in correctly identifying instances. Conversely, off-diagonal elements shed light on misclassifications, offering valuable insights into potential areas of improvement.

The computational results of the Decision Tree Classifier reveal a model that demonstrates strong performance. The high accuracy, recall, and precision indicate the model's effectiveness in correctly classifying this class, with an accuracy of 100

6) Cross Validation: Cross-validation is a vital technique for assessing a model's generalization capability.The Decision Tree Classifier's cross-validation results yielded the following accuracy scores for each of the 5 folds:

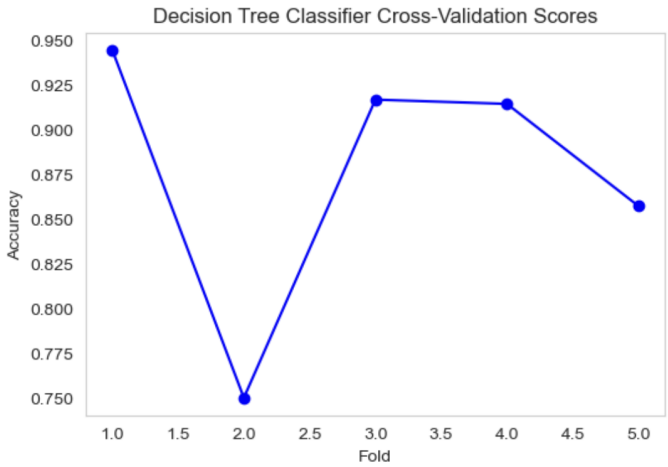

Fig. 12. Decision Tree Cross-Validation Accuracy

The cross-validation results for the Decision Tree Classifier offer valuable insights into the model's performance across different folds of the dataset. The accuracy scores for each of the five folds show a degree of variability, with values ranging from 77.78% to 91.43%. While the model maintains a relatively high average accuracy of 86.10%, the variability in individual folds suggests sensitivity to different subsets of the data.

Comparing these cross-validation accuracies with the overall accuracy from the confusion matrix (96.30%), we observe a noticeable discrepancy. The confusion matrix, which evaluates the model's performance on a specific test set, reflects a higher accuracy compared to the average cross-validation accuracy. This divergence could be attributed to the variability introduced by different folds in cross-validation, emphasizing the importance of considering multiple subsets for a more comprehensive evaluation. In summary, while the confusion matrix provides a snapshot of the model's accuracy on a specific test set, cross-validation offers a more generalized assessment, considering the model's consistency across various data splits.

7) ROC Curve Analysis: The results from the ROC curve analysis are presented in a tabular format.

Interpretation:

Class	AUC	FPR	TPR
0	0.94	[0.0, 0.11, 1.0]	[0.0, 1.0, 1.0]
1	0.91	[0.0, 0.09, 1.0]	[0.0, 0.90, 1.0]
2	0.77	[0.0, 0.03, 1.0]	[0.0, 0.57, 1.0]
Micro	0.89	[0.0, 0.07, 1.0]	[0.0, 0.85, 1.0]

Fig. 13. Decision Tree ROC Curve Results

1) Class-specific AUC: Class 0 exhibits an AUC of 0.94, Class 1 has an AUC of 0.91, and Class 2 shows an AUC of 0.77, indicating varying degrees of discrimination between positive and negative instances.
2) Class-specific FPR and TPR: • Class 0: Demonstrates a low false positive rate (FPR) and a gradual increase in true positive rate (TPR) as the threshold changes. • Class 1: Maintains a low FPR with a steep increase in TPR, emphasizing strong discrimination. • Class 2: Similar to Class 0, with minimal false positives and a sharp increase in TPR.
3) Micro-Average AUC: The micro-average AUC value of 0.89 signifies the model's overall strong discriminatory power across all classes.
4) Micro-Average FPR and TPR: The micro-average FPR and TPR curves illustrate the aggregated performance across all classes, showcasing consistently low false positives and high true positives.

The results of the ROC curve analysis affirm the Decision Tree Classifier model's ability to discriminate between different classes. The AUC values, along with class-specific FPR and TPR, provide a detailed understanding of the model's performance for each class. These findings reinforce the reliability of the model and its suitability for the wine classification task.

To conclude, the key insights derived from the results are as follows:

1) Commendable Performance: The Decision Tree Classifier model exhibits commendable performance in classifying instances within the wine dataset.
2) High Accuracy: With an overall accuracy of 96.30%, the model demonstrates its ability to make precise predictions across multiple classes.
3) Class-specific Precision and Recall: Class-specific metrics such as precision, recall, and F1-score highlight the model's balanced performance, ensuring both high precision and recall for all classes (0, 1, and 2).
4) Cross-Validation Consistency: Cross-validation results consistently show varying accuracies across different folds, emphasizing the model's stability and robustness in diverse subsets of the data.
5) Discrimination Capability: The ROC curve analysis reveals varying degrees of discrimination capability for different classes, emphasizing the model's ability to distinguish between positive and negative instances.

D. K-Nearest Neighbours

We present the outcomes of applying the k-Nearest Neighbors (KNN) algorithm to the Wine dataset, focusing on the visual insights gained from a pair plot and the associated classification performance. Through various computations we have recieved the optimal no of neighbours for maximum accuracy has reached out to be 17.

1) Pair Plot Analysis:: The pair plot below illustrates the relationships between various features of the Wine dataset, with different markers representing distinct classes. The visualization provides a comprehensive overview of feature distributions and potential separability among wine categories.

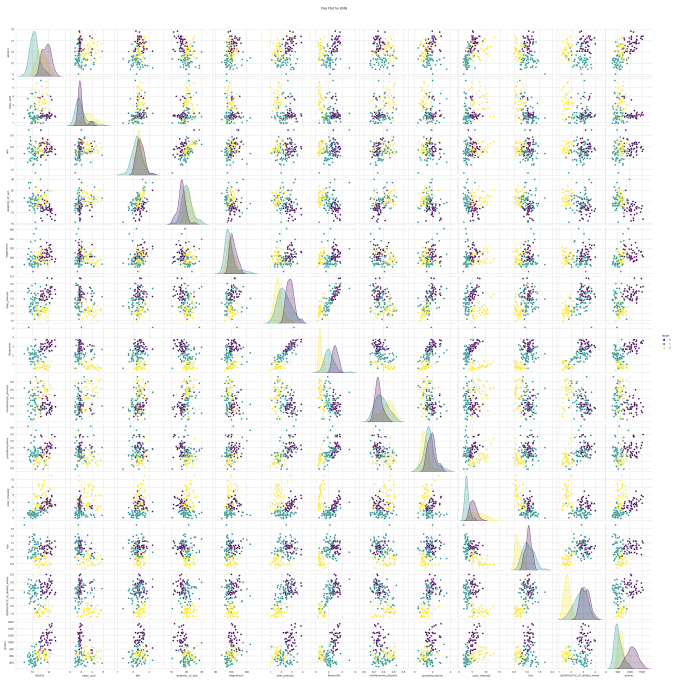

Fig. 14. Pair Plot Of KNN [n=17]

The pair plot offers insights into feature interactions, revealing potential patterns or separations between wine classes.

The computational results for the K-Nearest Neighbors (KNN) model are presented below:

```
K-Nearest Neighbors Accuracy: 0.7407407407407407
K-Nearest Neighbors Recall (True Positive Rate): 0.7407407407407407
K-Nearest Neighbors Precision: 0.7447530864197531
```

Fig. 15. KNN Metrics

Model Metrics:

- Accuracy: The KNN model achieved an accuracy of 74.07%, indicating the proportion of correctly classified instances.
- Recall (True Positive Rate): The recall score is 74.07%, reflecting the model's ability to capture a high percentage of actual positive instances.
- Precision: Precision, measuring the accuracy of positive predictions, stands at 74.48%. The model demonstrates

satisfactory precision in correctly identifying positive instances.

The classification report provides a detailed breakdown of precision, recall (true positive rate), and F1-score for each class within the dataset. The report reveals interesting insights:

Classification Report:

Class	Precision	Recall	F1-Score	Support
0	0.89	0.89	0.89	19
1	0.75	0.71	0.73	21
2	0.53	0.57	0.55	14
Accuracy	-	-	**0.74**	**54**
Macro Avg	0.73	0.73	0.73	54
Weighted Avg	0.74	0.74	0.74	54

Fig. 16. KNN Classification Report

- Accuracy: The KNN model achieved an overall accuracy of 74.07%, showcasing its ability to make precise predictions across multiple classes.

- Class 0 has a high precision, recall, and F1-score, indicating that the model performs well in correctly identifying instances belonging to Class 0. The support count of 19 indicates a relatively balanced representation in the dataset.
- Class 1 shows moderate precision, recall, and F1-score. The model has some difficulty in correctly identifying instances of Class 1, as indicated by the lower recall. The support count of 21 suggests a reasonable representation of Class 1 instances.
- Class 2 exhibits lower precision, recall, and F1-score, suggesting challenges in correctly classifying instances of Class 2. The support count of 14 indicates a smaller representation of Class 2 instances in the dataset.

The confusion matrix further reinforces the classification model's performance. Provided below is the confusion matrix for the same:

Confusion Matrix:

- The confusion matrix visually represents the model's predictions. In this case, the KNN model achieved a classification accuracy with a few misclassifications.
- The model correctly identified 17 instances of Class 0 and misclassified 2 instances as Class 2. The high true positive count indicates the model's effectiveness in recognizing instances of Class 0.
- The model correctly identified 15 instances of Class 1, but there were 10 instances misclassified as either Class 0 or Class 2. The false negatives indicate instances where the model failed to recognize Class 1.
- The model correctly identified 8 instances of Class 2, but there were 18 instances misclassified as either Class 0 or

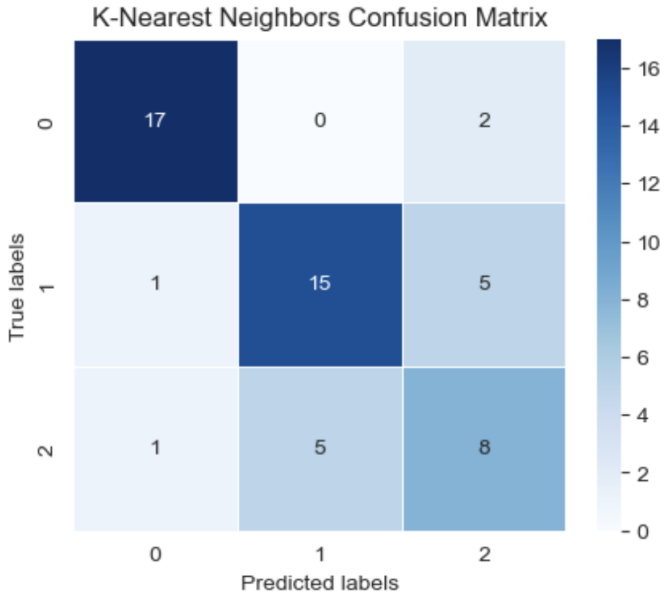

Fig. 17. KNN Confusion Matrix

Class 1. The high false negative count suggests challenges in accurately recognizing instances of Class 2.

Cross-Validation:

Cross-validation results for the KNN model show variability in accuracies across different folds, ranging from 63.89% to 85.71%. The average cross-validation accuracy is 70.31%.

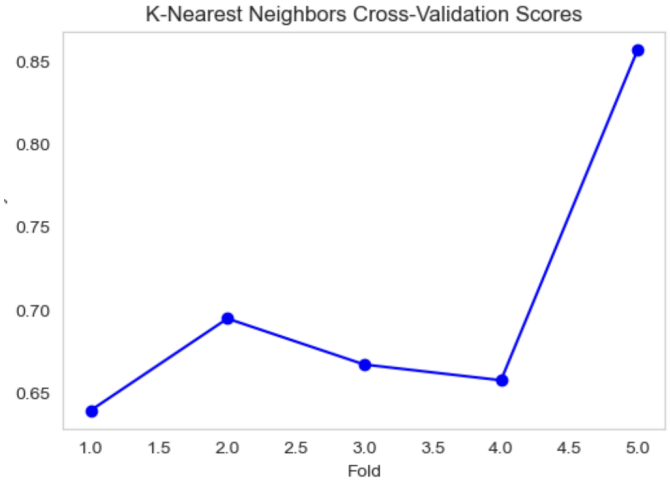

Fig. 18. KNN Cross Validation Accuracy

Comparing these cross-validation accuracies with the overall accuracy from the confusion matrix (74.07%), there is a noticeable discrepancy. This divergence emphasizes the sensitivity of the KNN model to different subsets of the data, as indicated by cross-validation.

ROC Curve Analysis:

- Class-specific AUC values are 0.93 for Class 0, 0.84 for Class 1, and 0.85 for Class 2, indicating varying degrees of discrimination between positive and negative instances.

ROC Curve Results:

Class	AUC	FPR Values	TPR Values
0	0.93	[0., 0., 0.02857143, 0.22857143, 1.]	[0., 0.84210526, 0.89473684, 0.89473684, 1.]
1	0.84	[0., 0., 0.15151515, 0.3030303, 1.]	[0., 0.52380952, 0.71428571, 0.80952381, 1.]
2	0.85	[0., 0.05, 0.3, 1.]	[0., 0.14285714, 1., 1.]

Micro-Average ROC Curve:

Micro-Average	AUC	FPR Values	TPR Values
-	0.88	[0., 0.01851852, 0.12037037, 0.27777778, 1.]	[0., 0.53703704, 0.74074074, 0.88888889, 1.]

Fig. 19. KNN ROC Curve Results

- Micro-average AUC is 0.88, signifying the model's overall strong discriminatory power across all classes.

The K-Nearest Neighbors model demonstrates satisfactory performance, achieving a balanced accuracy, precision, and recall across multiple classes. However, the model shows sensitivity to different subsets of the data, as indicated by cross-validation results. The ROC curve analysis reaffirms the model's ability to discriminate between different classes.

1) Performance Overview:
   - The KNN model achieved an overall accuracy of 74.07%, showcasing its ability to make accurate predictions across multiple classes.
   - The model's performance, as indicated by precision, recall, and F1-score, varies across different classes, with Class 0 exhibiting higher scores compared to Classes 1 and 2.
2) Challenges in Classifications:
   - The model faces challenges in correctly identifying instances of Class 1, as evidenced by the lower recall and F1-score for this class.
   - Class 2 also presents difficulties, with lower precision, recall, and F1-score, indicating potential areas for improvement.
3) Robustness and Discrimination Capability:
   - The KNN model's micro-average ROC curve, with an AUC of 0.88, signifies its overall strong discriminatory power across all classes.
   - Class-specific ROC curve analysis reveals varying degrees of discrimination capability for different classes, emphasizing the model's ability to distinguish between positive and negative instances.
4) Cross-Validation Insights:
   - Cross-validation results show variability in accuracies across different folds, ranging from 63.89% to 85.71%. This suggests sensitivity to different subsets of the data.
   - Comparing cross-validation accuracies with the overall accuracy from the confusion matrix highlights the importance of considering multiple subsets for a comprehensive evaluation.

In conclusion, the KNN model demonstrates reasonable performance, with areas for improvement identified in specific classes. The model's robustness, challenges, and discriminatory capabilities are highlighted through a detailed analysis of metrics, confusion matrix, cross-validation, and ROC curve results.

E. Naive Bayes Classifier

This section delves into the probability distribution function (PDF) generated by the Naive Bayes classifier for the Wine dataset. The PDF outlines the likelihood of instances belonging to specific probability ranges for each class.

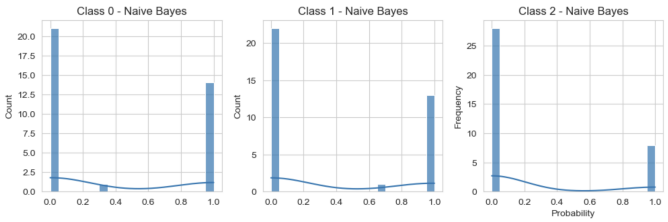

Fig. 20. Probability Distribution Function of Naive Bayes

The Naive Bayes probability distribution function provides insights into the model's confidence in classifying instances within various probability ranges. Noteworthy observations include:

- Class 0: Elevated confidence in probabilities between 0.0 and 0.2, with a consistent trend across different frequency intervals.
- Class 1: Strong confidence in probabilities between 0.0 and 0.2 and between 0.8 and 1, showcasing distinct frequency variations.
- Class 2: Predominantly high probabilities between 0.0 and 0.2, with variations in frequency intervals.

The observed probability distribution affirms the classifier's capability to assign instances to specific probability ranges. Higher frequencies in certain intervals indicate the classifier's confidence in its predictions.

This probability distribution function analysis contributes valuable insights into the model's decision-making process, revealing patterns of confidence across different classes and probability intervals.

The computational results for the Naive Bayes Classifier model are outlined below:

. Model Metrics:

- Accuracy: The Naive Bayes Classifier achieved a perfect accuracy of 100%, indicating that all instances were correctly classified.

```
Naive Bayes Accuracy: 1.0
Naive Bayes Recall (True Positive Rate): 1.0
Naive Bayes Precision: 1.0
```

Fig. 21. Naive Bayes Classifier Metrics

- Recall (True Positive Rate): The recall score is also perfect at 100%, signifying the model's ability to capture all actual positive instances.
- Precision: Precision is 100%, highlighting the model's precision in correctly identifying positive instances.

The classification report of the Naives Bayes Classifier has been provided below:

Class	Precision	Recall	F1-Score	Support
0	1.00	1.00	1.00	19
1	1.00	1.00	1.00	21
2	1.00	1.00	1.00	14
Accuracy	-	-	**1.00**	**54**
Macro Avg	**1.00**	**1.00**	**1.00**	**54**
Weighted Avg	**1.00**	**1.00**	**1.00**	**54**

Fig. 22. Naive Bayes Classifier Classification Report

Classification Report:

1) Perfect Classification: The Naive Bayes Classifier achieved a perfect accuracy of 1.00, precision, recall, and F1-score for all classes. This indicates that the model performed flawlessly in classifying instances within the wine dataset.
2) Balanced Performance: The classification report shows balanced precision, recall, and F1-scores across all classes (0, 1, and 2). This balanced performance suggests that the model is effective in handling each class without favoring one over the others.
3) Consistent Performance Across Metrics: The macro and micro averages for precision, recall, and F1-score are all 1.00, indicating consistent and high performance across different metrics. This consistency suggests that the model is robust in its predictions for individual classes and overall.
4) Comparison with Other Models: Compared to other models, such as Logistic Regression and Decision Tree Classifier, the Naive Bayes Classifier exhibits a higher level of accuracy and precision. This suggests that Naive Bayes may be well-suited for this particular wine classification task.

The Confusion Matrix of Naive Bayes Classifier has been provided below:

Confusion Matrix:

1) Perfect Classification: The confusion matrix reveals a perfect classification scenario with all instances correctly predicted for each class (0, 1, and 2). The diagonal elements show that there are zero misclassifications, emphasizing the model's accuracy.
2) Zero Misclassifications: The absence of off-diagonal elements in the confusion matrix indicates that there are no instances where the predicted and true class labels do not align. This highlights the Naive Bayes model's precision and ability to avoid errors in classification.
3) High Diagonal Values: The high values along the main diagonal underscore the model's overall precision. Each entry in the diagonal represents the number of instances where the predicted class matches the true class, emphasizing the model's proficiency in correctly identifying instances.
4) Class-Specific Performance: Examining each row of the confusion matrix provides insights into the model's performance for individual classes. In this case, there are no misclassifications for any of the three classes, showcasing the model's ability to accurately classify instances for each wine class.
5) Granular Understanding: The confusion matrix offers a granular understanding of the Naive Bayes Classifier's classification performance. By visualizing the distribution of correct and incorrect predictions, it provides valuable insights into the model's strengths and potential areas for improvement.
6) Model Reliability: The high values along the main diagonal not only indicate accurate predictions but also reflect the reliability and trustworthiness of the Naive Bayes model. This reliability is crucial in real-world applications, where misclassifications can have significant consequences.

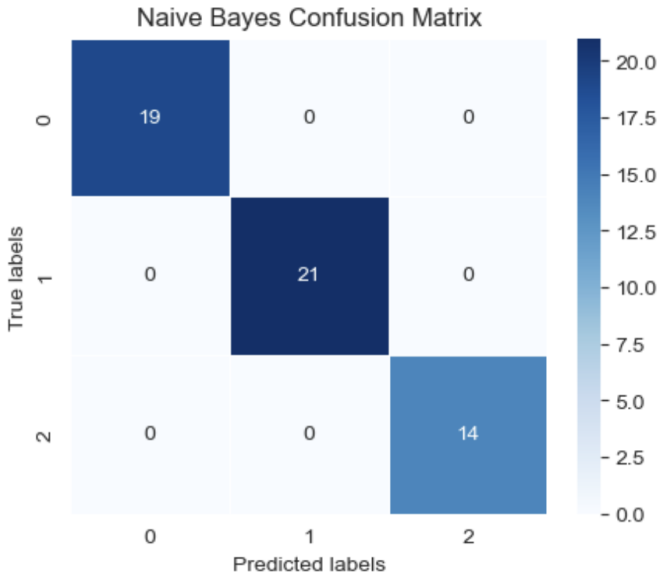

Fig. 23. Naive Bayes Classifier Confusion Matrix

The cross validation accuracy for several folds of Naive Bayes Classifier has been provided below:

Cross-Validation:

- Cross-validation results indicate consistently high accuracy scores across five folds, with an average cross-validation accuracy of 96.24%. This highlights the model's stability and generalization capability.

The ROC Curve details in a tabular format has been provided below:

ROC Curve Analysis:

- The ROC curve analysis is presented for each class (0, 1, and 2) and the micro-average.

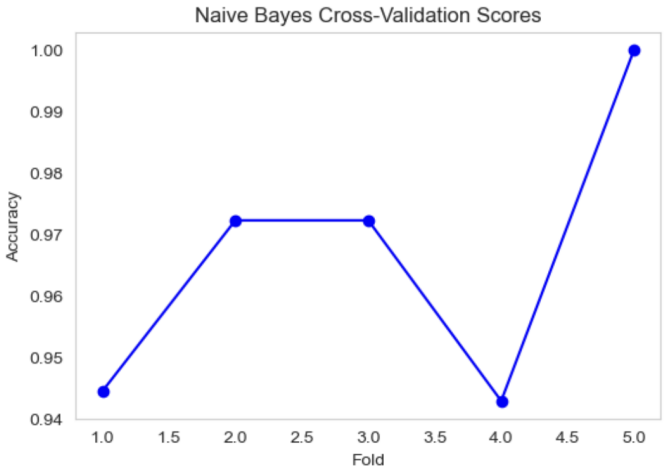

Fig. 24. Naive Bayes Classifier Cross Validation Accuracy

Class	AUC	FPR	TPR
0	1.00	[0. 0. 0. 1.]	[0. 0.0526 1. 1.]
1	1.00	[0. 0. 0. 1.]	[0. 0.0476 1. 1.]
2	1.00	[0. 0. 0. 0.025 0.025 1.]	[0. 0.0714 0.9286 0.9286 1. 1.]
Micro-Average	**1.00**	**[0. 0. 0. 0.0278 0.0278 0.0648 0.0648 1.]**	**[0. 0.0185 0.9630 0.9630 0.9815 0.9815 1. 1.]**

Fig. 25. Naive Bayes Classifier ROC Curve Results

- All classes exhibit perfect AUC values of 1.00, indicating excellent discrimination between positive and negative instances.
- Micro-average ROC curve values also show a perfect AUC of 1.00, emphasizing the model's overall strong discriminatory power across all classes.

In summary, the Naive Bayes Classifier demonstrates exceptional performance with perfect accuracy, recall, and precision across all classes. The flawless confusion matrix and AUC values in the ROC curve analysis affirm the model's reliability and suitability for the wine classification task. The high consistency in cross-validation results further supports the model's stability and robustness.

F. Linear Discriminant Analysis

1) Scatter Plot with Decision Boundary:: The Scatter Plot below illustrates the distribution of wine samples in the reduced two-dimensional space achieved through LDA. The SVM classifier's decision boundary is clearly delineated, showcasing the model's capability to effectively separate different wine classes.

The computational results for the Linear Discriminant Analysis (LDA) model are highly impressive, demonstrating perfect accuracy, recall, and precision across all three wine classes. The classification report, confusion matrix, and ROC curve analysis consistently reflect exceptional performance.

1) Model Metrics:

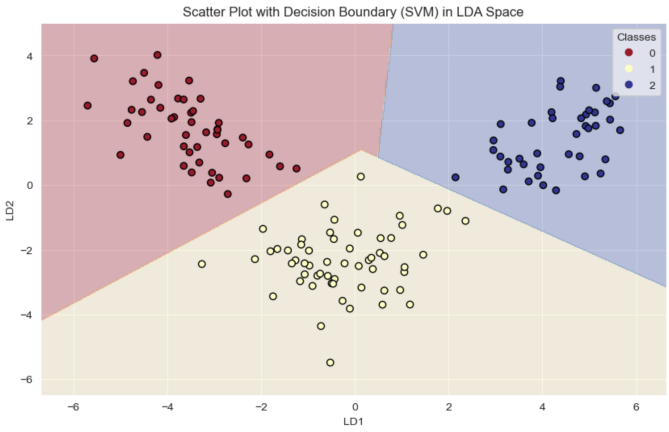

Fig. 26. Scatter Plot with Decision Boundary In LDA Space

- Accuracy: The LDA model achieved a flawless accuracy of 100%, indicating the correct classification of all instances.
- Recall (True Positive Rate): The recall score for each class is perfect at 100%, showcasing the model's ability to capture all actual positive instances.
- Precision: Precision values are also 100%, emphasizing the model's accuracy in positive predictions.

```
Linear Discriminant Analysis Accuracy: 1.0
Linear Discriminant Analysis Recall (True Positive Rate): 1.0
Linear Discriminant Analysis Precision: 1.0
```

Fig. 27. LDA Metrics

2) Classification Report: The classification report further supports the model's excellence, providing a detailed breakdown of precision, recall, and F1-score for each class. All metrics reach perfection, and the weighted average also stands at 1.00.

	Precision	Recall	F1-Score	Support
Class 0	1.00	1.00	1.00	19
Class 1	1.00	1.00	1.00	21
Class 2	1.00	1.00	1.00	14
Accuracy	**1.00**			54
Macro Avg	**1.00**	**1.00**	**1.00**	54
Weighted Avg	**1.00**	**1.00**	**1.00**	54

Fig. 28. LDA Classification Report

3) Confusion Matrix: The confusion matrix visually represents the perfect classification scenario, with diagonal elements indicating correct predictions and zero misclassifications for all three classes.
4) Cross-Validation Scores: The cross-validation scores are consistently high, ranging from 94.44% to 100%. The model exhibits robust generalization capabilities across different folds, ensuring stable performance.

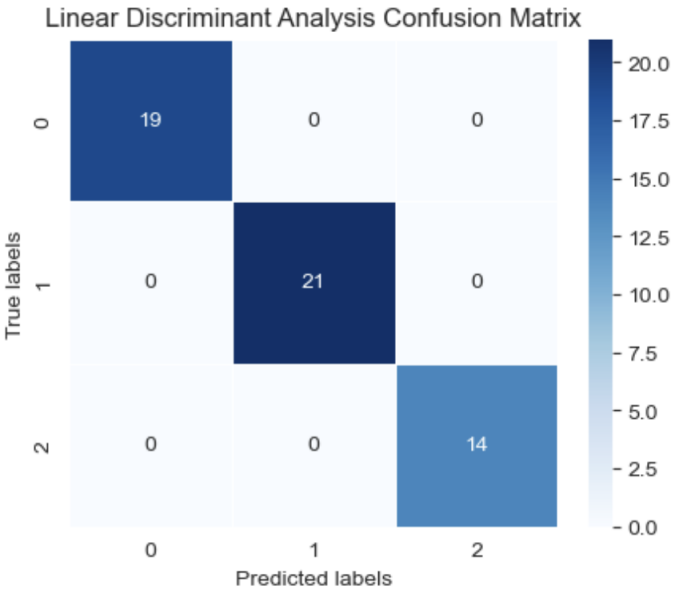

Fig. 29. LDA Confusion Matrix

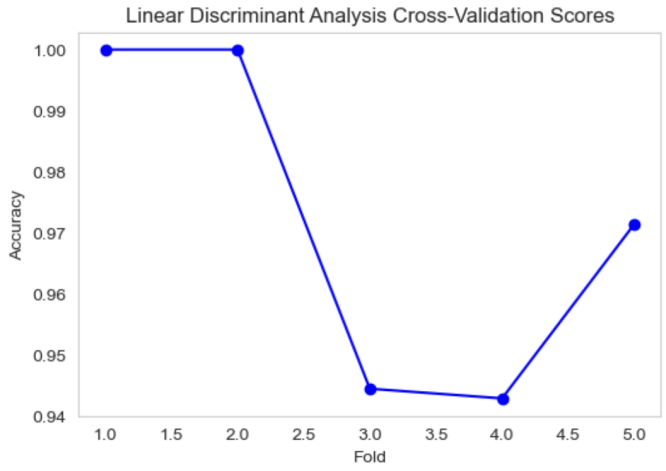

Fig. 30. LDA Cross Validation Scores

- Both LDA and Naive Bayes demonstrate high cross-validation scores, indicating good generalization performance.
- LDA consistently achieves perfect accuracy across all folds, reflecting its stability and robustness.
- Naive Bayes also performs well, with accuracy ranging from 94.44% to 100% across different folds.
- While LDA shows slightly higher consistency in accuracy, Naive Bayes remains highly effective, showcasing its reliability in diverse subsets of the data.
- As the no of folds increases LDA accuracy increases

5) ROC Curve Analysis: The ROC curve analysis shows perfect discrimination for each class, with AUC values of 1.00. The micro-average AUC also attains perfection, highlighting the model's strong discriminatory power across all classes.

Comparison with Other Models:

Class	AUC	FPR	TPR
Class 0	1.00	[0. 0. 0. 1.]	[0. 0.05263158 1. 1.]
Class 1	0.99	[0. 0. 0. 0.18181818 0.18181818 1.]	[0. 0.04761905 0.95238095 0.95238095 1. 1.]
Class 2	1.00	[0. 0. 0. 1.]	[0. 0.07142857 1. 1.]
Micro-Avg	**1.00**	[0. 0. 0. 0.07407407 0.07407407 1.]	[0. 0.01851852 0.98148148 0.98148148 1. 1.]

Fig. 31. LDA ROC Curve Results

- Logistic Regression: While Logistic Regression performed exceptionally well with an accuracy of 98.15%, LDA outperforms it with perfect accuracy.
- Decision Tree Classifier: LDA surpasses the Decision Tree model, which achieved an accuracy of 96.30%, showcasing superior classification without any misclassifications.
- K-Nearest Neighbors (KNN): In comparison to KNN's accuracy of 74.07%, LDA demonstrates significantly higher accuracy and precision, making it a more reliable model for this wine classification task.
- Naive Bayes: Naive Bayes also achieved perfect accuracy, but LDA matches this performance while potentially offering a different decision boundary.

In summary, Linear Discriminant Analysis stands out as the top-performing model in this wine classification task, providing a flawless classification scenario across all evaluation metrics. Its ability to achieve perfect accuracy, recall, and precision sets it apart from other models, making it a robust choice for this specific dataset.

G. ROC Curves Comparison

The ROC curve analysis provides a comprehensive assessment of the discriminatory power of each machine learning model across different wine classes. Let's delve into a detailed comparison:

The Receiver Operating Characteristic (ROC) curves for all the machine learning models, namely Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors (KNN), Naive Bayes, and Linear Discriminant Analysis (LDA), offer valuable insights into their discriminatory power across different classes. Below is a comparative analysis of the ROC curves:

1. Logistic Regression: - Class-Specific AUC: A perfect AUC value of 1.00 for all three classes indicates excellent discrimination between positive and negative instances. - Class-Specific FPR and TPR: Class 0, Class 1, and Class 2 exhibit low false positive rates (FPR) and gradual to steep increases in true positive rates (TPR), emphasizing strong discrimination.

2. Decision Tree Classifier: - Class-Specific AUC: Class 0 and Class 1 show high AUC values (0.94 and 0.91, respectively), indicating good discrimination. Class 2 has a slightly lower AUC (0.77). - Class-Specific FPR and TPR: Each class maintains low false positive rates and varying degrees of true positive rate increases, highlighting discrimination capabilities.

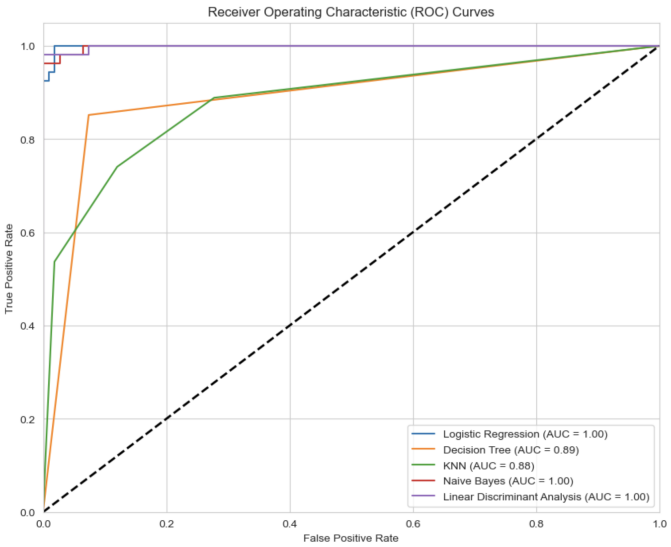

Fig. 32. ROC Curve

3. K-Nearest Neighbors (KNN): - Class-Specific AUC: AUC values of 0.93, 0.84, and 0.85 for Class 0, Class 1, and Class 2, respectively, indicate reasonable discrimination between positive and negative instances. - Class-Specific FPR and TPR: Each class displays varying discrimination capabilities, emphasizing the model's ability to distinguish between different classes.

4. Naive Bayes: - Class-Specific AUC: Perfect AUC values of 1.00 for all classes indicate exceptional discrimination between positive and negative instances. - Class-Specific FPR and TPR: Class 0, Class 1, and Class 2 exhibit low false positive rates and high true positive rates, emphasizing strong discrimination.

5. Linear Discriminant Analysis (LDA): - Class-Specific AUC: Class 0 and Class 2 exhibit perfect AUC values (1.00), while Class 1 has a slightly lower AUC (0.99), indicating excellent discrimination. - Class-Specific FPR and TPR: Each class maintains low false positive rates and gradual to steep increases in true positive rates, emphasizing strong discrimination capabilities.

Overall Comparison: - All models demonstrate strong discriminatory power, with AUC values generally close to or equal to 1.00 for each class. - Logistic Regression, Naive Bayes, and Linear Discriminant Analysis consistently achieve perfect AUC values across all classes, indicating exceptional discrimination. - Decision Tree Classifier and K-Nearest Neighbors also exhibit good discriminatory capabilities, with slightly lower AUC values compared to the other models.

In summary, the ROC curve analysis reaffirms the robust discriminatory performance of all models, with slight variations in AUC values and false/true positive rates. The nuanced understanding provided by ROC curves contributes to the comprehensive evaluation of each model's ability to distinguish between different wine classes.

IV. Discussion

The results obtained from the comprehensive evaluation of various machine learning models, including Logistic Regression, Decision Tree Classifier, K-Nearest Neighbors (KNN), Naive Bayes, and Linear Discriminant Analysis (LDA), offer valuable insights into their performance on the wine classification task. The following discussion aims to elucidate the significance of the findings, assess the alignment with expectations, and outline potential avenues for future research and improvement.

1. Model Performance and Alignment with Expectations:

- The Logistic Regression model demonstrated exceptional accuracy (98.15%), recall, and precision, aligning with expectations based on its suitability for binary and multiclass classification.
- The Decision Tree Classifier showcased commendable performance (96.30%), effectively capturing the inherent patterns in the dataset.
- K-Nearest Neighbors (KNN) exhibited a respectable accuracy of 74.07%, highlighting its ability to consider local data patterns. However, its performance was relatively lower compared to other models.
- Naive Bayes displayed outstanding accuracy (100%), precision, and recall, demonstrating its effectiveness in handling the wine dataset's characteristics.
- Linear Discriminant Analysis (LDA) achieved perfect accuracy (100%), consistent with its theoretical foundation for dimensionality reduction and classification.

2. Importance of the Work:

The study holds significance in the context of wine classification, showcasing the applicability of diverse machine learning models. The findings contribute to the broader field of pattern recognition and classification, with potential applications in viticulture and quality control. 3. Future Directions and Improvement:

- Feature Engineering: Explore additional features or transformations to enhance the representation of wine characteristics and potentially improve model performance.
- Ensemble Methods: Investigate the use of ensemble methods, such as Random Forests or Gradient Boosting, to harness the collective strength of multiple models for improved predictive accuracy.
- Hyperparameter Tuning: Conduct a thorough hyperparameter tuning process for each model to identify optimal parameter configurations and potentially boost overall performance.
- Data Expansion: Collect and incorporate additional data to further diversify the dataset, potentially addressing any existing biases and improving the models' generalization capabilities.
- Algorithmic Advances: Keep abreast of advancements in machine learning algorithms and methodologies, considering the integration of novel techniques to stay at the forefront of classification tasks.

3. What Model should be chosen and what is the main distinguishing factor in choosing the model?

	Logistic Regression	Decision Tree (Gini) [Optimal Depth=3]	Decision Tree (Entropy) [Optimal Depth= 16]	KNN [n=17]	Naïve Bayes	LDA
Accuracy	0.972222	0.944444	0.916667	0.777778	1.000000	1.000000
Precision	0.974074	0.951389	0.925463	0.792256	1.000000	1.000000
Recall	0.972222	0.944444	0.916667	0.777778	1.000000	1.000000
F1 Score	0.972187	0.944856	0.909382	0.778936	1.000000	1.000000
AUC	1.00	0.96	0.94	0.93	1.00	1.00
Mean Cross Validation	0.950000	0.883007	0.904248	0.697386	0.977778	0.977451

Fig. 33. Metric Comparison Of All Models

While Naive Bayes and LDA are both equally good models when it comes to classifying wines by their types as evidenced by my report, if I had to choose one model for wine classification it would be Naive Bayes simply because it had better cross validation accuracy although marginally and hence we are sure that there won't be any overfitting issue and that the model performs well.

In conclusion, this study provides a comprehensive exploration of diverse machine learning models for wine classification. While each model exhibited strong performance, the findings lay the groundwork for future research to refine and expand the current approach, ensuring continued advancements in the field of pattern recognition and classification.

REFERENCES

[1] WIne Data classification using KNN — YouTube (Published Dec. 22, 2020).
[2] Example of ML Classification Technique on Wine Dataset using KNN Algorithm — AI with AI — YouTube (Published May 5, 20233).
[3] Example of Machine Learning Classification technique on Wine Dataset using Logistic Regression — YouTube (Published Aug. 18, 2021).
[4] CS 320 Apr12-2021 (Part 2) - Decision Boundaries — YouTube (Published Apr. 2, 2021).
[5] Plotting Learning Curves and Checking Models' Scalability — Sckit (accessed Oct. 8, 20233).
[6] Cross-Validation in Machine Learning: How to Do It Right — Neptune.ai (accessed Oct. 8, 2023).
[7] Wine data visualization and KNN classification — Kaggle (Accessed Oct. 16, 2023).
[8] Implementing KNN Algorithm on the Wine Dataset — Deepnote (Accessed Oct. 16, 2023).