Classification and Clustering Model for leaves

Fundamentals of Machine Learning’s second project

Ali Akbar Ahrari

Isfahan University

second project

Classification and Clustering Model for leaves

Course: Fundamentals of Machine Learning

Student Name: Ali Akbar Ahrari - 4003613001

Teacher Name: Mr. Mohammad Kiani

Table of Contents

[Note 3](#_Toc170410481)

[Part 1: Classification 3](#_Toc170410482)

[Implementation 3](#_Toc170410483)

[Libraries 3](#_Toc170410484)

[Loading the Data 3](#_Toc170410485)

[Data Pre-process 4](#_Toc170410486)

[Data Preparation 4](#_Toc170410487)

[Train Model 4](#_Toc170410488)

[Making Predictions 5](#_Toc170410489)

[Evaluate the model 5](#_Toc170410490)

[Part 2: Clustering 5](#_Toc170410491)

Note

All codes are available in the directory.

For reviewing code, open main.ipynb.

# Part 1: Classification

## Implementation

### Libraries

In this section, we import the necessary libraries:

* pandas: For data manipulation and analysis.
* numpy: For numerical operations.
* train\_test\_split from sklearn.model\_selection: To split the dataset into training and testing sets.
* RandomForestClassifier from sklearn.ensemble: To build and train the Random Forest classification model.
* classification\_report, accuracy\_score, confusion\_matrix from sklearn.metrics: For evaluating the performance of the classification model.
* matplotlib.pyplot and seaborn: For data visualization.

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix  
import matplotlib.pyplot as plt  
import seaborn as sns

### Loading the Data

In this section, we prepare the data for the classification task:

1. Loading the Dataset: We loaded the data into a DataFrame df. The dataset is assumed to have no headers, hence header=None. After loading, column names are assigned programmatically.

1. Splitting Features and Target: We separate the features (X) and the target variable (y). X contains all the columns except target, while y contains only the target column(“feature\_0” which is the class labels).
2. Train-Test Split: We split the data into training and testing sets using train\_test\_split. The test\_size=0.3 parameter specifies that 30% of the data will be used for testing. The random\_state=42 parameter ensures reproducibility, and stratify=y ensures that the target distribution is similar in both the training and testing sets.

### Data Pre-process

For data preprocessing, first we check the statistical information using the df.describe(). Since the column count for each feature is set to 340, we can ensure that there is no missing value. Another important thing we gained throw this table is that some columns contain outliers. We are using random forest, so it is not necessary to handle them.

for any missing values in the dataset:

### Data Preparation

We separated the features and the target variable. Here, feature\_0 is assumed to be the target variable.

X = df.drop(columns='feature\_0')  
y = df['feature\_0']

After defining X and Y, let’s split the dataset into training and testing sets:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)

* test\_size=0.3: Specifies that 30% of the data should be used for testing.
* random\_state=42: Ensures reproducibility of the results.
* stratify=y: Ensures that the class distribution is similar in both training and testing sets.

### Train Model

We chose the Random Forest Classifier for classifying this model. Here are several reasons:

* High Accuracy and Robustness: Random Forest is known for its ability to produce high accuracy. This is due to its ensemble nature, where it builds multiple decision trees and merges them together to get a more accurate and stable prediction. This approach reduces the variance of the model and improves accuracy.
* Robust To Outliers: Random Forest is capable of handling missing values and is relatively robust to outliers. In this dataset outliers are common, and having an algorithm that can manage these without extensive preprocessing is advantageous.
* Interpretability: Feature importance scores provided by Random Forest help in understanding the contribution of each feature to the model. This can be crucial for domains where interpretability is important.
* Handling High-Dimensional Data: dataset has many features (attributes of leaves). Random Forest provides insights into which features are most important, helping with feature selection and also, it can handle datasets with high dimensionality efficiently.
* Non-Linearity: Random Forest can capture non-linear relationships in the data since each tree in the forest can independently model complex interactions between features.

Initialize the Random Forest classifier and train it on the training data.

clf = RandomForestClassifier(n\_estimators=100, random\_state=42)  
clf.fit(X\_train, y\_train)

n\_estimators=100: Specifies the number of trees in the forest.

random\_state=42: Ensures reproducibility of the results.

clf.fit(): Fits the model to the training data.

### Making Predictions

Now we can make predictions on the test set.

# Make predictions on the test set  
y\_pred = clf.predict(X\_test)

### Evaluate the model

Using the accuracy score, we found a model with an accuracy of 81%.

accuracy = accuracy\_score(y\_test, y\_pred)  
print(f'Accuracy: {accuracy:.2f}')

# Part 2: Clustering