Development of a Neural Network for Tree Species Classification in University Garden

A simple report on the developed model

|  |  |  |
| --- | --- | --- |
| Index number | Registration Number | Name |
| 5010 | ICT/19/20/072 | M.M.F. Manal |
| 4964 | ICT/19/20/023 | M.W.W.Dhilshani |
| 4952 | ICT/19/20/010 | M.Z.T. Arsalany |

Assignment -ICT3212 - Introduction to Intelligent Systems

Group Name: Byte Brilliance

Group Members:

Introduction

The primary objective of this assignment is to develop a classification model capable of accurately distinguishing between different tree species found in the university garden. With the increasing importance of tree species classification in various domains such as environmental studies, urban planning, and biodiversity conservation, the need for reliable classification models has increased.

By accurately identifying and categorizing tree species, researchers and practitioners can gain valuable insights into ecosystem dynamics, species distribution patterns, and habitat conservation priorities. In urban environments where green spaces are essential for enhancing quality of life, tree species classification supports initiatives aimed at maintaining and enhancing urban biodiversity.

As per these considerations, the development of a reliable classification model for identifying tree species found in the university garden is not only an academic exercise but also a practical exercise with real-world applications. As students through field survey data, dataset preparation, model training, and comprehensive reporting, this assignment aims to equip us with the necessary skills and knowledge to tackle complex classification tasks in environmental science and related disciplines.

Overview of selected Species.

The selected tree species found in the university garden include ***Sansevieria trifasciata Hahnii***, ***Dracaena Trifasciata***, and ***Sanseveria Liberica***. These species are challenging to distinguish through visual observation due to their similar morphological characteristics, such as leaf structure, leaf color, leaf spread, plant height, and leaf arrangement. Moreover, these species may exhibit variations within the same species, making manual classification prone to errors.

Pictures of the plants are shown below

1. ***Sanseveria Liberica***

Sanseveria Liberica, sometimes referred to as the Liberia Snake Plant, shares similarities with other snake plants but has its unique traits, particularly in leaf formation and patterning.

Key Characteristics:

* Leaf Structure: Broad, somewhat shorter than Dracaena Trifasciata.
* Leaf Color: Green with white to light green vertical streaks.
* Leaf Spread: Less compact, with leaves spreading out more.
* Plant Height: Up to 75 cm.
* Leaf Arrangement: Less dense rosette, more spreading.

**Visual Description:**

Sanseveria Liberica features a more open rosette with broader leaves compared to its relatives. Its leaves are usually lighter in color and have a distinctive vertical striping pattern.

1. ***Sansevieria trifasciata Hahnii***





Sansevieria trifasciata Hahnii, commonly known as the Bird's Nest Sansevieria, is a compact variety of the snake plant. It features short, wide leaves arranged in a rosette pattern, which is **distinct from other varieties due to its dwarf size**.

**Key Characteristics:**

* **Leaf Structure:** Broad and slightly concave.
* **Leaf Color:** Dark green with lighter horizontal stripes.
* **Leaf Spread:** Compact rosette.
* **Plant Height:** Typically up to 15-20 cm.
* **Leaf Arrangement:** Rosette formation.

**Visual Description:** Sansevieria trifasciata Hahnii looks like a small, leafy cup with a tight rosette pattern. Its leaves are thick and fleshy, exhibiting a mottled pattern of green shades.

1. ***Dracaena Trifasciata***.

Dracaena Trifasciata, also known as Snake Plant or Mother-in-Law's Tongue, is characterized by its upright, sword-like leaves. It is widely appreciated for its resilience and ability to thrive in low-light conditions.

Key Characteristics:

* Leaf Structure: Long, erect, and pointed.
* Leaf Color: Dark green with lighter green horizontal bands.
* Leaf Spread: Vertical and slightly spreading.
* Plant Height: Can reach up to 1 meter.
* Leaf Arrangement: Vertically arranged in a rosette pattern.

Visual Description:

This species has tall, rigid leaves that grow upright, creating a striking, architectural appearance. The leaves are often variegated with lighter bands across a darker green background.

**Differentiating Features of Selected Species**

While the above species share many similar characteristics, there are specific features that help differentiate them:

* **Sansevieria trifasciata Hahnii**: This species is notably smaller than the other two, with shorter leaves. It displays a clear rosette structure that is compact and tightly arranged.
* **Dracaena Trifasciata**: The leaves of this species are almost vertical, with a very slight spread. This upright angle helps distinguish it from Sanseveria Liberica.
* **Sanseveria Liberica**: Compared to Dracaena Trifasciata, Sanseveria Liberica has a wider spread of leaves. The leaves are not as vertical and have a broader arrangement, giving the plant a more open appearance.

These differentiating features are crucial for accurate identification and classification, given the visual similarities in leaf patterns and plant structure among these species.

Dataset Preparation

The data collection process involved conducting manual field surveys within our university garden, where we identified and observed all three plant species. During these surveys, we measured various attributes of each plant to compile our dataset. This included determining the plant's overall height, as well as the length and width of its leaves. Additionally, we recorded the number of leaves clustered together and calculated the angle at which each leaf diverged from the vertical axis.

To ensure accuracy in our measurements, we utilized tools such as measuring tapes, rulers, and an online protractor for angle measurements.

In addition to traditional data collection methods, we collected multiple leaf samples from each plant found within the university premises to capture a diverse range of measurements and characteristics. This approach facilitated the creation of a comprehensive dataset that accurately represents the morphological variations present among the selected tree species in our garden.

**Sample Selection Criteria:**

1. Samples were specifically selected from the university garden to ensure consistency in environmental conditions and habitat characteristics.
2. Only specimens belonging to the target tree species Sansevieria trifasciata Hahnii, Dracaena Trifasciata, and Sanseveria Liberica were included in the dataset. This criterion ensured that the collected data accurately reflected the morphological characteristics of the selected species.
3. Tree age can influence morphological features, such as size and leaf structure, it was not explicitly used as a criterion for sample selection. Instead, efforts were made to include a diverse range of specimens representing different stages of growth within each species.
4. A limited number of samples from trees exhibiting minor variations were also included to account for natural variability within the population.
5. Emphasis was placed on capturing a wide range of morphological variations within each species, including differences in leaf size, plant size and arrangement of leaves. This criterion ensured that the dataset encompassed the full spectrum of morphological diversity present among the target tree species.

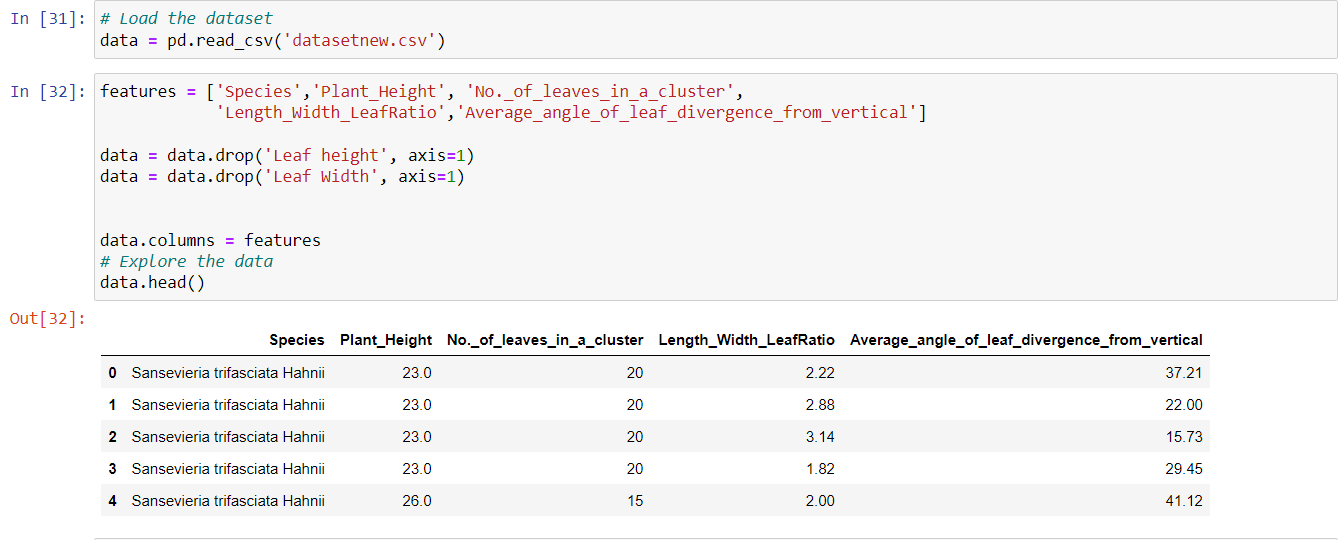
Feature engineering is essential for capturing the most relevant information from raw data. By selecting and transforming features, we can emphasize patterns that are crucial for distinguishing between different classes, such as tree species.

To identify the most relevant distinguishing features, we collaborated with biology lecturers in our department who are domain experts. Additionally, we also conducted a research on these plants to determine the most distinguishing characteristics. In this way we were able to find some distinguishing features which could be measured externally with available resources.

Once the dataset was prepared we loaded it and performed some preprocessing techniques before feeding it to our model. The dataset was loaded by reading the CSV file named 'datasetnew.csv' into a pandas DataFrame named **data**. This initial step brought the raw data into the Python environment for further processing. We then defined the features to be used in our dataset, ensuring we included all relevant attributes for our analysis. To streamline our dataset, we removed the 'Leaf height' and 'Leaf Width' columns, as their ratio would be used for classification purposes. We then renamed the columns of the DataFrame to match the names specified in the **features** list, ensuring clarity and consistency. We also used data.head() to display the first few rows of the DataFrame.

Data Augmentation techniques

And then we applied some preprocessing techniques to our dataset which is shown in the below code.





Summary of Preprocessing Steps:

1. Data Cleaning:
   * Replaced missing values with an average value.
2. Feature Selection:
   * Dropped irrelevant columns ('Leaf height' and 'Leaf Width').
3. Renaming Columns:
   * Renamed columns for clarity and consistency.
4. Handling Categorical Variables:
   * Applied one-hot encoding to the 'Species' column using OneHotEncoder.
5. Feature Scaling:
   * Normalized the feature set using StandardScaler to ensure all features are on a similar scale.

These preprocessing steps were essential to prepare the raw data into a suitable format for our machine learning models, thereby enhancing the quality and performance of our classification model. Also we did not want to do any vectorization as our dataset is already in numerical format. Vectorization refers to the process of converting textual data into numerical vectors that machine learning models can understand.

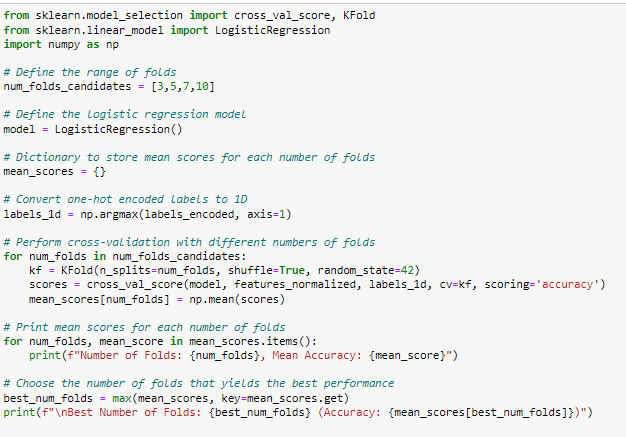
Data Split into training and testing datasets.

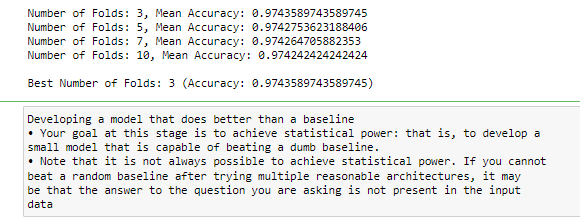
After applying the preprocessing techniques, we proceeded to split the data into training and testing datasets. Given the limited amount of data available, we explored two strategies for effective data splitting: Simple hold-out validation and K-fold cross-validation.

Initially, we used Simple hold-out validation to get a quick estimate of the model's performance. However, to ensure that the model's evaluation is reliable with a small dataset, we opted for K-fold cross-validation. This method involves dividing the dataset into multiple folds and training the model on different subsets of the data, ensuring that each data point gets a chance to be in the training and testing sets.

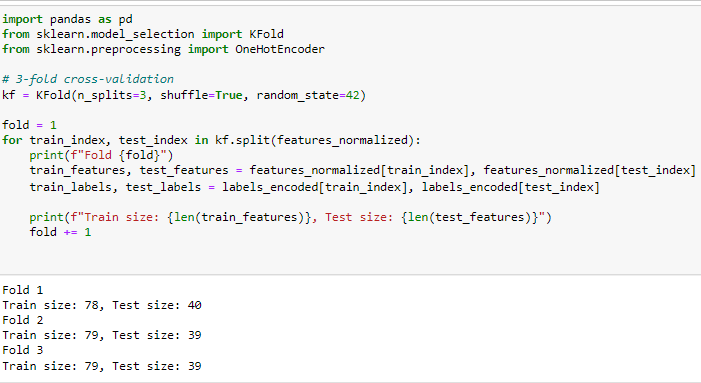
There was a small difference with the test accuracy but still, we decided to proceed with **K-fold cross-validation** as it provided a better accuracy compared to the simple hold-out method.

When proceeding with K-fold cross validation we had to decide the number of folds. For this we used a simple logistic regression model and tested with different number of folds and found the number of folds which has the best mean accuracy. This mean accuracy was also considered as the baseline accuracy. When developing a model, we always should try to develop a small model that is capable of beating a dumb baseline.





As per the above results our dataset was split into three folds. Each folds had the data as follows:



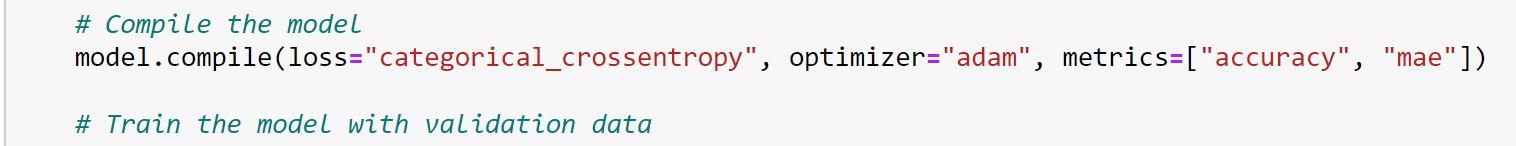
Architecture of the Proposed Model:

In the proposed model several key design decisions were made to create an effective model for the classification task. Grid search was used to make those decisions.

1. **Input Layer**:
   * The input layer accepts the normalized feature set (**features\_normalized**) of the data. Normalizing the features ensures that they are on a similar scale, which can improve the training process of the model.
2. **Hidden Layers**:
   * **First Hidden Layer**: This layer consists of 64 units and is activated by the Rectified Linear Unit (ReLU) activation function. ReLU is a commonly used activation function that introduces non-linearity into the model and helps alleviate the vanishing gradient problem.
     + Regularization: L2 regularization with a regularization strength of 0.001 is applied to the weights of this layer. L2 regularization penalizes large weights in the network, helping to prevent overfitting by adding a penalty term to the loss function.
   * **Second Hidden Layer**: The second hidden layer is deeper, with 128 units. It also uses the ReLU activation function and L2 regularization with the same regularization strength as the first hidden layer.
3. **Output Layer**:
   * The output layer consists of a dense layer with a number of units equal to the number of classes in the encoded labels (**labels\_encoded**).
   * Activation Function: The softmax activation function is used for multi-class classification. Softmax converts the raw output scores of the network into probabilities, with each output representing the probability of the input belonging to a particular class.

**Selection of Architecture**:

* The architecture was selected through experimentation (Grid Search).
* Different configurations of hyperparameters, such as the number of hidden layers, the number of units in each layer, and regularization techniques, were tried.
* Regularization, specifically L2 regularization, was incorporated to prevent overfitting, especially given the limited size of the dataset. Overfitting occurs when the model learns to memorize the training data instead of generalizing well to unseen data.
* The final architecture was chosen based on validation performance, aiming to achieve a balance between model complexity and generalization ability.



To compile the model categorical\_crossentropy is the loss function used. It is commonly used for multi-class classification problems. This loss function measures the difference between the true distribution of the labels and the predicted distribution produced by the model.

The optimizer determines how the model is updated based on the gradient of the loss function. Here, **"adam"** is chosen as the optimizer. And here **"accuracy"** and **"mae"** (mean absolute error) are chosen as the metrics.

Training.

**Implementation Platform**:

The model was implemented using Keras, a high-level neural networks API, inside the Anaconda Navigator environment. Anaconda Navigator provides a user-friendly interface for managing packages and environments, making it easier to work with various data science tools. TensorFlow was the backend for Keras, providing efficient computation and optimization of neural network operations.

Libraries used:

1. **Pandas**: Used for data manipulation and analysis with DataFrames and Series..
2. **NumPy**: Supports numerical computing with multi-dimensional arrays and mathematical functions. Essential for numerical computations and handling the data in a format suitable for machine learning models.
3. **Matplotlib**: Used for creating static and interactive visualizations of data. It's utilized for plotting training and validation loss and accuracy curves to visualize model performance.
4. **TensorFlow**: Serves as a framework for implementing machine learning algorithms and deploying models. Provides comprehensive tools for building and optimizing neural networks.
5. **Keras**: Utilized to implement machine learning algorithms, it entails importing Keras modules like Sequential and Dense, offering a user-friendly interface for constructing and training artificial neural network models with TensorFlow as the backend.
6. **Scikit-Learn**: Used for implementing machine learning algorithms and splitting datasets into training and testing sets. Offers effective tools and a uniform interface for diverse tasks, including cross-validation, hyperparameter tuning, and evaluation metrics.
7. **EarlyStopping**: A callback from Keras used to monitor the validation loss and stop training early if the validation loss does not improve for a specified number of epochs, helping to prevent overfitting.
8. **StandardScaler**: Part of Scikit-Learn, used to standardize features by removing the mean and scaling to unit variance, which is crucial for many machine learning algorithms that are sensitive to the scale of input data.
9. **OneHotEncoder**: Also from Scikit-Learn, it encodes categorical features as a one-hot numeric array, converting categorical labels into a format suitable for machine learning algorithms.
10. **GridSearchCV**: A Scikit-Learn utility that performs hyperparameter tuning by exhaustively searching through a specified parameter grid to find the best combination of hyperparameters.
11. **KerasClassifier**: A wrapper that allows using Keras models within Scikit-Learn's API, enabling easy integration with Scikit-Learn utilities such as GridSearchCV for hyperparameter tuning.
12. **KFold Cross-Validation**: A method from Scikit-Learn that splits the data into K consecutive folds for cross-validation, helping to assess the model's performance more robustly by using each fold as a validation set once while the K-1 remaining folds form the training set.
13. **Keras Regularizers**: Used to apply penalties on layer parameters or layer activity during optimization to prevent overfitting by penalizing large weights, such as L2 regularization.
14. **Scikit-Learn Metrics**: Provides functions to evaluate the performance of the model using various metrics, including precision, recall, F1 score, ROC AUC, and accuracy, to assess the quality of the model's predictions.

**Training Process**:

* **Early Stopping**: EarlyStopping is a callback mechanism used to monitor a specified metric, typically validation loss, during training. In this case, the validation loss was monitored, and if it did not improve for 3 consecutive epochs, training was halted. Early stopping helps prevent overfitting by stopping training when the model's performance on the validation set starts deteriorating, thus avoiding unnecessary computations and saving time.
* **Batch Size and Epochs**: The model was trained using a batch size of 1, meaning that each training iteration processed one sample at a time. The maximum number of epochs was set to 300, meaning that training would continue for up to 300 iterations. However, due to early stopping, training might terminate earlier if there is no improvement in the validation loss.
* **Validation Split**: During the final training phase, 20% of the data was used as a validation set. The validation set serves as a proxy for unseen data and is used to monitor the model's performance during training. By evaluating the model on a separate validation set, it's possible to detect overfitting and tune hyperparameters accordingly.
* **Callback Mechanism**: The use of callback mechanisms, such as EarlyStopping, allows for dynamic adjustments during the training process. Callbacks can be configured to perform specific actions at different stages of training, enhancing the model's adaptability and improving its performance. In this case, EarlyStopping helps ensure that the model does not overfit to the training data by terminating training when the validation loss stops improving.

Hyper parameter Optimization

Techniques used:

* 1. Grid Search: Grid Search is an exhaustive search over a specified parameter grid. For this project, Grid Search was used to explore the following hyperparameters:
* **Learning Rate:** The learning rate is a crucial hyperparameter that determines the step size during the weight update in training. The impact on model performance is as follows:

Low Learning Rate: When the learning rate is too low the model may converge very slowly or get stuck in a local minimum. This can result in longer training times and potentially suboptimal performance.

Optimal Learning Rate: An optimal learning rate allows the model to converge quickly and effectively to a good solution. It balances between making progress towards the global minimum and maintaining stability in training.

High Learning Rate: A high learning rate can cause the model to converge too quickly or even diverge, resulting in poor performance as the model may overshoot the optimal solution.

* **Number of Hidden Layers:** The number of hidden layers determines the depth of the neural network.

Too Few Layers: A shallow network may not capture the complexity of the data, leading to underfitting and poor performance.

Optimal Number of Layers: An optimal number of hidden layers provides the network with enough capacity to learn from the data without overfitting. This can improve model performance by capturing more intricate patterns.

Too Many Layers: A very deep network can lead to overfitting, especially if the training data is limited. This results in the model performing well on training data but poorly on unseen test data.

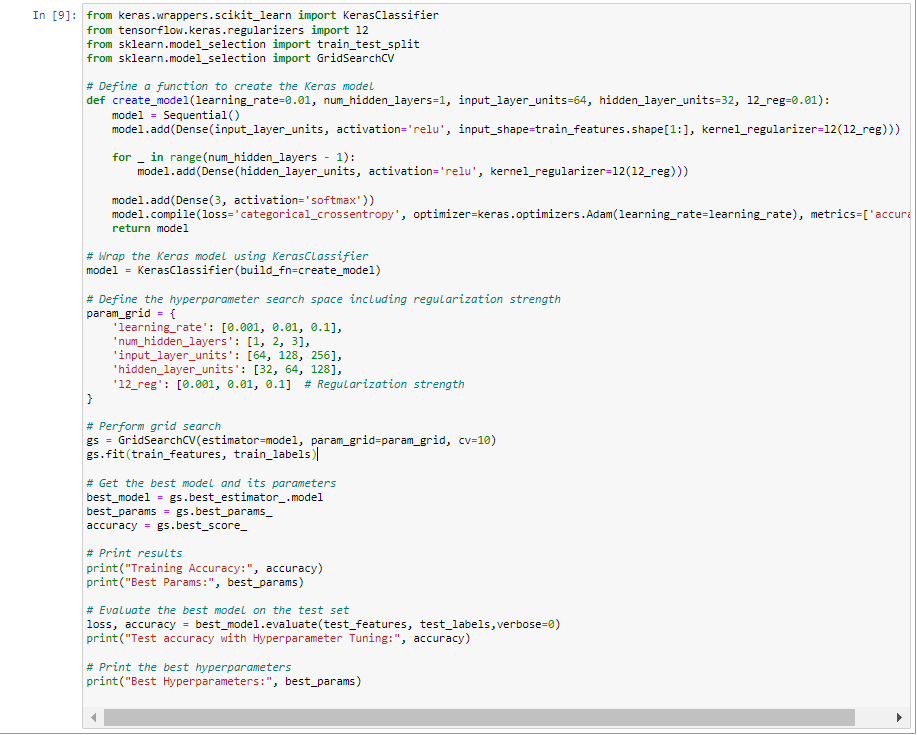
* **Number of Units in Each Layer:** The number of neurons in each hidden layer affects the capacity of the model to learn.

Too Few Units: A small number of units in each layer may not provide sufficient capacity to capture the underlying patterns in the data, leading to underfitting and poor performance.

Optimal Number of Units: An optimal number of units allows the network to effectively learn and generalize from the data. This balance ensures that the model is powerful enough to learn complex patterns without overfitting.

Too Many Units: A large number of units can increase the model's capacity but also its tendency to overfit. This is especially problematic if combined with insufficient regularization or small datasets.

* L2 Regularization Strength**:** L2 regularization helps prevent overfitting by penalizing large weights. This regularization helps in ensuring that the model does not memorize the training data but rather learns to generalize from it.



* 1. K-Fold Cross-Validation: K-Fold Cross-Validation is used to ensure that the model generalizes well across different subsets of the data. In this case, different fold sizes (3, 5, 7, 10) were tested to find the optimal number of folds that provided the best cross-validation accuracy.

Quantitative and Qualitative Results.

**Quantitative Results:**

Quantitative results were calculated across different folds and then the average results were finally calculated. The average results are shown below.

* Accuracy: Accuracy measures the proportion of correct predictions out of the total predictions made by the model. It is calculated as the number of correct predictions divided by the total number of predictions.
* Precision: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It is calculated as the number of true positive predictions divided by the sum of true positive and false positive predictions.
* Recall: Recall, also known as sensitivity, measures the proportion of true positive predictions out of all actual positive instances in the dataset.
* F1 Score: The F1 score is the harmonic mean of precision and recall. It is calculated as 2 \* ((precision \* recall) / (precision + recall)).
* ROC AUC (Area Under the Receiver Operating Characteristic Curve): ROC AUC measures the area under the ROC curve, which plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) for different threshold values.

****

**Qualitative Results:**

For the qualitative assessment of the model's performance, several methods were used.

Confusion Matrix Analysis: The confusion matrix provides a breakdown of the number of true positive, true negative, false positive, and false negative predictions for each class.

Correctly Classified Examples: The first five samples that were correctly classified by the model were analyzed.

Incorrectly Classified Examples: Similarly, the first five samples that were incorrectly classified were scrutinized.

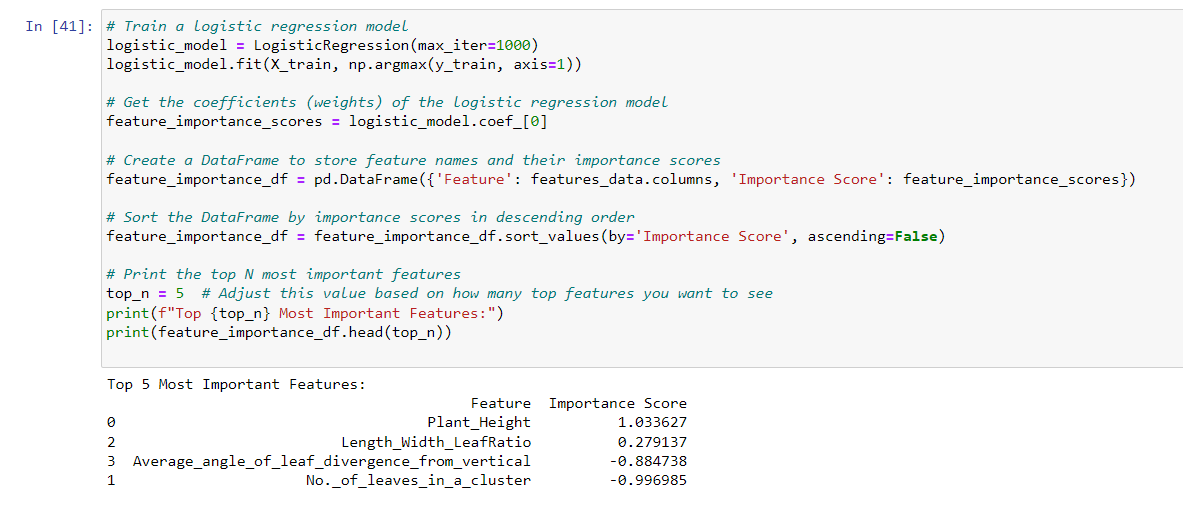
Predictions for the First 10 Samples: Lastly, the predictions for the first 10 samples in the dataset were examined in detail. This step allows for an analysis of the model's output, including the predicted class labels, associated probabilities, and corresponding true class labels.

Discussion on Results.

The model shows good performance on accuracy, precision, recall, F1 Score and ROC AUC. While high performance metrics are desirable, they may sometimes indicate overfitting, especially if the dataset is small or not diverse enough.

Also more attention and effort should be given to feature quality. The most appropriate features should be selected and they should play a major role in the classification. The most important features were considered after doing a manual feature engineering. But as per the below results it shows a negative impact on the features average angle of leaf divergence from vertical and number of leaves in a cluster. This is a problem we faced at the end and we should find a solution for this.

Moreover we should try expanding or dataset for better diversity and accurate better results and better training.



Summary of Key Findings and Implications

**Key Findings:**

* 1. High Model Performance

The model achieved high accuracy (97.63%), precision (98.06%), recall (97.39%), F1 score (97.57%), and ROC AUC (99.72%), indicating good predictive capabilities.

* 1. Effective Feature Utilization:

The features used, such as plant height, leaf height, number of leaves in a cluster, leaf width, leaf length-width ratio, and average angle of leaf divergence from vertical, were used in distinguishing between species. This shows that morphological characteristics can be used as indicators of species in this context.

* 1. Successful Handling of Missing Data:

Missing values were replaced with column averages, allowing the model to maintain high performance without being negatively affected by incomplete data.

* 1. Potential Overfitting and Class Imbalance Concerns:

The high performance metrics may indicate potential overfitting, especially if the dataset is not sufficiently diverse or large. To solve this regularization, normalization techniques were used along with k fold cross validation to split the data.

**Implications:**

1. Application in Agricultural and Botanical Fields:

The model's high accuracy and precision make it suitable for applications in agriculture, horticulture, and botanical research where accurate species identification is crucial.

This can be useful in tasks such as monitoring plant health, ensuring correct species usage, and preserving biodiversity.

1. Enhanced Plant Species Identification:

The model can be used to develop automated systems for plant species identification, reducing the need for expert human intervention and speeding up the identification process.

1. Improving Dataset Quality:

Ensuring the dataset is representative of real-world variability will be important for maintaining the model’s performance on unseen data. This may involve collecting more data points and have a broader range of species and morphological characteristics.

1. Addressing Overfitting and Class Imbalance:

Implementing regularization techniques, cross-validation, and balancing class weights or using oversampling methods can help address potential overfitting and class imbalance issues, leading to a better model.

1. Future Research and Model Enhancement:

Further research could explore more advanced methods for handling missing data and enhancing model interpretability.

1. Deploying in Real-World Scenarios:

Before deploying the model in real-world scenarios, it’s essential to validate its performance on diverse and extensive datasets to ensure generalizability.Collaborations with botanical experts and agricultural practitioners can provide valuable feedback for refining and improving the model.