**Predicting Plant Health Using Machine Learning Models**

**Week 2 Update**

**Introduction:**

* In this document, I aim to explain the predictive model that monitors and analyzes plant health based on various environmental parameters. This initiative is a part of the smart planter project which is designed to assist both home gardeners and farmers in optimizing their yield and minimizing waste. The predictive model will utilize machine learning algorithms to analyze data from various sensors and predict the health of the plant.

**Dataset Description:**

* The dataset contains the following attributes:
  + **Humidity (%):** The percentage of water vapor present in the air surrounding the plants.
  + **Nitrogen Levels (mg/kg):** The concentration of nitrogen in the soil, measured in milligrams per kilogram.
  + **Ambient Temperature (°C):** The surrounding temperature measured in degrees Celsius.
  + **Presence of Sunlight**: A binary attribute indicating whether the plant is exposed to sunlight.
  + **Plant Health:** The target variable indicating the health status of the plant.

**Objective**

* To develop a predictive model that accurately predicts plant health based on the given features.
* To assist in the development of a smart planter system that is user-friendly and provides real-time insights into plant health.

**Tools Uses For this project:**

1. **Google Colab:**
   * **Free Cloud Environment:** Google Colab provides free access to a cloud based Jupyter notebook environment with GPU support. This is extremely useful for individuals and small teams that may not have access to high-performance hardware for training large ML models.
   * **Pre-installed Libraries:** Colab comes with many pre-installed libraries commonly used in ML, such as TensorFlow, PyTorch, and scikit-learn. This saves time on environment setup.
   * **Easy Sharing and Collaboration:** You can easily share Colab notebooks with others, making it a great tool for collaborative work. Multiple people can work on the same notebook simultaneously.
   * **Integration with Google Drive:** Colab integrates seamlessly with Google Drive, making it simple to save and organize your notebooks and datasets.
   * **Access to Datasets:** Colab offers access to various datasets through Google Cloud, which can be helpful for experimenting with different data sources.
   * **Uses:** It is simple to use, just go to the website <https://colab.research.google.com/> and then click on the upload section and upload the ipynb file from the local computer or can create a new ipynb file.
2. **Visual Studio Code (VS Code) with Jupyter Notebook:**
   * **Customizable Development Environment:** VS Code is a highly customizable code editor. With the Jupyter extension, you can turn it into a full-fledged Jupyter Notebook environment. This allows you to use your preferred code editor with all the features and extensions you need.
   * **Local Development:** Unlike Colab, which is cloud-based, VS Code is a local development environment. This means you have more control over your environment and can work offline.
   * **Integrated Debugging:** VS Code offers powerful debugging capabilities, which can be extremely useful when developing complex ML models. You can set breakpoints, inspect variables, and debug step by step.
   * **Rich Ecosystem:** VS Code has a rich ecosystem of extensions, including those for Git integration, code linting, and many other development tasks. This makes it versatile for both ML and general software development.
   * **Code Version Control:** With VS Code, you can easily integrate with version control systems like Git, which is crucial for tracking changes in your ML projects.
   * **Uses:** To follow the local environment, there is need to setup the VS code and Ipynb in VS code and then open the ipynb file and run it.

The choice between Google Colab and VS Code with Jupyter Notebook integration often depends on the specific needs and preferences:

* Use Google Colab if you want a hassle-free, cloud-based environment with GPU support, easy collaboration, and access to various datasets.
* Use VS Code with Jupyter Notebook if you prefer a customizable local environment, require advanced debugging capabilities, or work on ML projects that involve a mix of code development and data analysis. For further installation and its configuration, you may follow this article link: <https://www.alphr.com/vs-code-open-jupyter-notebook/>
* Currently I am using VS Code with Jupyter Notebook, but if I run the same code with google colab there will be no issue with it.

**Data Simulation Function: generate\_data**

* This function simulates the generation of a dataset with 10000 data points, which are equally distributed across three categories: 'Healthy', 'Moderate', and 'Unhealthy'. These categories represent different states of plant health. The attributes for each data point include 'Humidity (%)', 'Nitrogen Levels (mg/kg)', 'Ambient Temperature (°C)', 'Presence of Sunlight', and 'Plant Health'. The values of these attributes are generated based on certain conditions defined for each category to simulate real-world scenarios.
* **Parameters**
  + **n**: The total number of data points to be generated. It is set to 10000 by default.
* **Returns**
  + **df**: A shuffled pandas DataFrame containing the generated data points with their respective attributes.

**Function Breakdown**

1. **Initialization**:
   * A dictionary named **data** is initialized with keys representing the different attributes and empty lists as their corresponding values.
2. **Data Generation**:
   * The function iterates over three categories: 'Healthy', 'Moderate', and 'Unhealthy'.
   * For each category, it generates **n/3** data points (to ensure an equal distribution) with attribute values generated based on predefined conditions that represent typical values for each category.
3. **Conditions for Data Generation**:
   * **Healthy**:
     + Humidity: Between 50% and 70%.
     + Nitrogen Levels: Between 100 and 150 mg/kg.
     + Ambient Temperature: Between 15°C and 25°C.
     + Presence of Sunlight: Always 1 (present).
     + Plant Health: Labeled as 'Healthy'.
   * **Moderate**:
     + Humidity: Between 40% and 80%.
     + Nitrogen Levels: Between 80 and 170 mg/kg.
     + Ambient Temperature: Between 10°C and 30°C.
     + Presence of Sunlight: Randomly chosen between 0 (absent) and 1 (present).
     + Plant Health: Labeled as 'Moderate'.
   * **Unhealthy**:
     + Humidity: Between 0% and 100%.
     + Nitrogen Levels: Between 0 and 200 mg/kg.
     + Ambient Temperature: Between -10°C and 40°C.
     + Presence of Sunlight: Randomly chosen between 0 (absent) and 1 (present).
     + Plant Health: Labeled as 'Unhealthy'.
4. **Dataframe Creation and Shuffling**:
   * After generating the data, it is converted into a panda DataFrame.
   * The DataFrame is then shuffled to randomize the order of data points and the index is reset.

* To create a balanced dataset with an equal number of 'Healthy', 'Moderate', and 'Unhealthy' instances, I create a function **generate\_data** to simulate the dataset with each label, and then shuffle the dataset to mix it.
* **Explanation**:
  + I divide the total number of data points by 3 to get an equal number of data points for each category ('Healthy', 'Moderate', and 'Unhealthy').
  + I then generate data for each category separately, using different ranges of values for the features to simulate the different conditions for each category.
  + After generating the data, I create a DataFrame and shuffle it using df.sample(frac=1).reset\_index(drop=True) to mix the data points from the different categories together.
* Here is the result, showing top 10 rows of dataset:

A screenshot of a computer screen

Description automatically generated

**Build Neural Networks to Predict Plant Health:**

**Step 1: Data Preprocessing**

Before feeding the data into a neural network, it's essential to preprocess it. This includes encoding categorical labels and normalizing the feature values.

**Step 2: Splitting Dataset**

Split the data into training and testing sets i.e 20 percent testing data and 80 percent training dataset.

**Step 3: Building neural networks:**

Initially I used 3 dense layers with one output layer having SoftMax activation function to classify the categories correctly.

Then compile which means build the model with Adam as optimizer, sparse\_categorical\_crossentropy as loss function and metrics is accuracy.

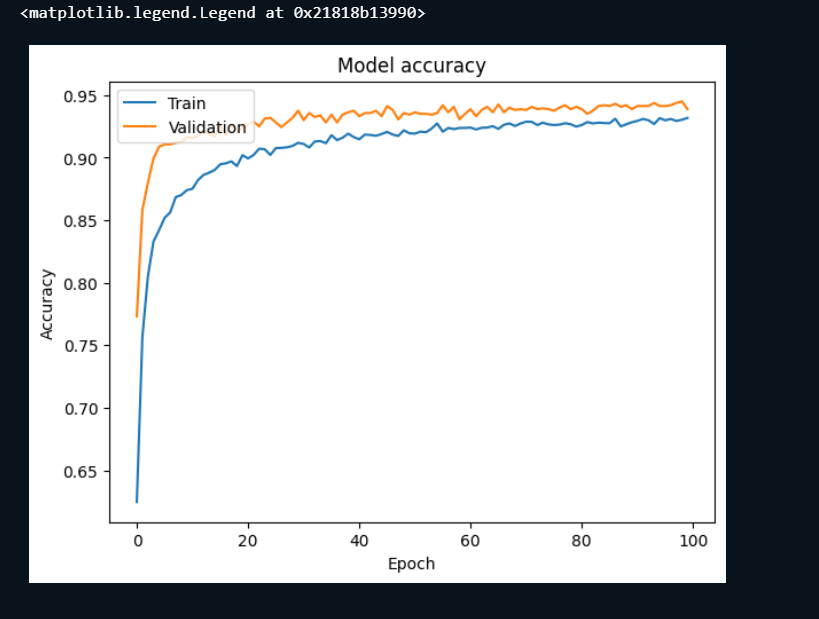
**Step 4: Training Model:**

Now its time to train the model, so I feed the training and testing data to the model with 100 epochs, 32 batch size and 20 percent validation split.

It takes a little time to train the model on the given dataset.

**Step 5: Visualize the model accuracy and loss:**

Here is the visualization.



The above graph shows the accuracy is not stable yet, which means it required further tunning to make the model further accurate. Later, I will do that.

**Step 6: Evaluate Model:**

Now its time to test the model on the testing data to make sure the model will predict the unseen data accurately or not.

After evaluating the model on test data, I got below result, which shows a good stats but still need to be improve



**Step 7: Making Predictions:**

I feed the model with some unseen data on which the model is not train and get the result which shows.

A screen shot of a computer code

Description automatically generated

Based on given data points the model predicts the plant is healthy.

**Conclusion:**

Until now the model is working fine. It requires further tunning.

I will also use other prediction models and will compare the results.