Plant Traits 2024 – Kaggle Competition

Henry Van Boskirk[1] and Zachary Debler[1] and Ryan Gilbert[1]

[1] University of Florida, Gainesville FL 32603, USA
https://github.com/henryvanboskirk/DA2-Project-Group-16

Abstract. The paper will analyze plant images taken by citizen science plant photographers as well as ancillary variables such as climate, soil, and multitemporal satellite variables using deep learning-based regression models, such as Neural Networks to predict key traits among plants. Stem Specific Density, Leaf Area Per Leaf Dry Mass, Plant Height, Seed Dry Mass, Leaf Nitrogen, and Leaf Area are the relevant traits to be predicted.

The Kaggle Competition we have chosen aims to use deep learning to make predictions on these six traits. As a baseline, we submitted totally random predictions. We increasingly used more complex models, next with a linear regression model on just the data table of ancillary variables. Our team also tried gradient boosted trees as well as random forest regression to make better predictions. These models do not consider the citizen science plant photographs at all, however.

For our final models, we transitioned to the most complex models of our project: Neural Networks. We used the PyTorch framework and its documentation to create models. We adjusted hyperparameters like number of epochs, batch size, learning rate, and depth of the neural network to increase the accuracy of our predictions.

1 Introduction

Citizen scientists have provided Kaggle with tens of thousands of plant images. They are labeled with a unique ID, as well as many ancillary variables. The ancillary variables are linked to the images by this ID. These "extra" variables are features such as temperature data, precipitation data, soil data (which interpolates various soil properties like sand or pH levels), MODIS (which measures the optical reflection of sunlight at that geocoordinate), and VOD, which is data from a radar constellation that is sensitive to water content and the biomass of plants. There are almost 200 of these ancillary features.

1.1 Project Goal

We want to use the data provided to us to predict these specified traits. Using the ancillary features, as well as the many images provided by citizen science photographers, or even using the two data sources in conjunction, we will create increasingly complex machine learning models that will predict those traits.

Listed below are the plant traits that we will be predicting using our model. These traits allow researchers to understand the ability of plants to adapt to environmental conditions and manage the responses of plant communities to environmental change to conserve biodiversity in natural ecosystems.

Table 1. Plant Traits

Plant Traits	Definition	Significance
Stem Specific Density(X4)	Density or mass per unit volume of the stem tissue of a plant	Assessing forest health.
Leaf Area Per Dry Leaf Mass(X 11)	Relationship between the surface area of a leaf and its dry weight	Higher drought tolerance, longer leaf lifespan, and lower photosynthetic capacity per unit area
Plant Height(X18)	Height of plant	Predicting and managing the responses of plant communities to environmental change
Seed Dry Mass(X26)	Refers to the weight or mass of a seed after it has undergone desiccation or drying	Influences plant fitness, population dynamics, community structure
Leaf Nitrogen(X50)	Concentration of nitrogen present in the leaves of a plant	Fundamental roles in plant growth, photosynthesis, nutrient cycling
Leaf Area(X3112)	Total surface area of a leaf exposed to light and the surrounding environment	Crucial for plant photosynthesis, resource capture, and plant growth.

2 Data Sources

The data provided for analysis includes thousands of photographs from thousands of citizens around the globe. With species identification apps, and AI algorithms, the species are identified and given in the data. The data of the plants cover all ecosystem types and continents and encompass an enormous array of species.

Fig. 2. Samples of the images included in the dataset can be visualized above. Their species is outlined in the print at the bottom of each image in the dataset. [2]

We were also provided csv files of the ancillary variables. 'train.csv' has around 55,000 rows and 176 columns, 'test.csv' has around 7,000 rows and 164 columns, and 'sample_submission' has around 6500 rows and 7 columns.
Our final model includes a section for loading in the csv files for training and testing. We created a path for understanding the images and the data we were working with. We wanted to determine what the context of a random image would entail such as angle of image, distance it was taken from, background considerations and more. To do this, we sampled nine random images from the testing and training image files and displayed them. During this part of the project, we also needed to ensure that all the images were formatted in a standardized way, so we verified that they were all the same size and image type. After running this verification process, we realized that while all the images were of the same size and type, there was a complete uniqueness to each photo. Furthermore, we understood that we needed to create a machine learning model to try to account for this randomness and uniqueness to the images provided.

2.1 Methods/Models

Visualization

Our final model includes many correlation matrices that aim to understand the relationship between the feature columns and target columns. After determining that the above factors were the target columns, we needed to predict we assigned all other columns to be feature columns. To find the feature columns which had the greatest correlation with the target columns we aimed to create correlation matrices for each of the subsections of the feature columns. After reviewing the data understood that many of these columns follow a similar pattern so we wanted to sort of combine them into bands. Our first correlation matrix labeled any "relevant_features" as ones with a correlation of greater than or equal to 0.12. From this graph we discovered that only the X4 feature had correlations greater than this value. We thought this to be rather interesting because we figured a simple trait such as "Plant Height(X18)" would be more correlated with certain feature columns. After we created this correlation matrix, we then implemented a KBest selector to output the top 15 feature columns for each target column. To understand the data better, we printed the total list of the feature columns to see where each of these 15 feature columns originated and the complexity of their relationship.

Our final visualization step involved finding the correlations of each feature column. We accomplished this by creating multiple correlation matrices for each subsection of the feature column data. For example, we separated the MODIS features into bands and displayed a correlation matrix for each of the months in that specific band. We also did this with the

WORLDCLIM_ columns to see which of these factors correlated with each other. The entire goal with creating these correlation matrices was to get a better understanding of the feature column data so we could create a filtered list of the most important factors for our neural network. Due to the magnitude of the data involved in this project we recognized the need to filter down this information to only display the most vital information. These filtered lists ignored irrelevant feature columns when running the final regressor on the neural network. We would not have been able to make sense of this filtered list if we were not able to visualize the data as we had in the visualization section of our final code.

Linear Regression Models

We began by creating a linear regression model on just the table of ancillary data. Even though the tables have tens of thousands of rows, and almost 200 features, it was still much less data to process than if we worked on the images first. Since linear regression models are generally less complex, easier to understand, and easy for us to write in Python, we decided to start there. Since there are six traits to be predicted, we ran six separate linear regression models on the features for each predicted trait, and iteratively added the predictions on the test data into our submissions data frame.

Next, we employed feature selection in our linear regression model. Feature selection is a vital part of the process of machine learning. Since our data set has almost 200 features, many of these might be irrelevant to predicting the traits, so including them in a model risks overfitting the data to the training set, causing the model to underperform against unseen data on which it is tested. Reducing the number of features can also make the model faster to run.

We selected the top 50 features that had the highest correlation with the traits we were trying to predict, and we ran updated linear regression models using just these variables. Interestingly, the feature selection process gave us a much worse R-squared value for our Kaggle competition than using all the features.

Gradient Boosted Regression

Gradient Boosted Regression models are tree-based models that we can use to make predictions. The self-correcting nature of gradient boosting is effective in reducing variance, and since residual optimization is used, these models can have greatly reduced bias. Similarly to linear regression, we did six separate gradient boosted regression models on each of the six traits. However, we also found that the R-squared value was less than the original linear regression model's score.

Random Forest Models

We moved onto another complex tree model, the Random Forest Regression model. Random forest models are supervised learning models that combine trees to create a more accurate prediction. The downside of these kinds of models is the level of interpretability. While linear regression just outputs a formula for the data, random forest models might find it hard to understand how predictions are being made. We ran into the issue of applying the learned model to our test data. We were not able to make predictions using the random forest model. Additionally, the complexity of the model caused it to run for four hours on an above-average GPU, so we moved on to different models.

Gray scaling Images

Originally, one of our methods for cleaning the data was to grayscale all the images. Since the images provided were colored JPEG files, we thought that gray scaling them would result in easier models to run, since only ⅓ of the pixel data would be necessary. Using the image library from PIL, we iterated through all 60,000 images and converted them to grayscale. We did not use the grayscale images for several reasons. Most importantly, since we were reading the images off our PC's file system, it was intensive to have 60,000 images saved, and grayscaling them added another 60,000 images. We also found later that using a Kaggle notebook allowed us to read the image files directly from the website, without having to download all the images onto our personal computers. Also, since every person's computer has unique file pathing, reading directly from Kaggle's data allowed for more standardization when working on the project together.

Neural Networks

We began by loading in all the images from the test and training data sets. We also load in the list of trait names from the meta data file and the three csv files: train, test, and submission. We wrote code that iteratively went through each image to ensure that they were all the same size. Since every train and test image was 512x512 pixels, and all in jpeg form, we didn't need to resize any of the pictures. If the images were not similarly sized, as you might expect from many citizen science photographers, we would have to resize each image to a standard size. We adapted some code from one of the Kaggle starter notebooks to begin data exploration and work for the neural network. The link to it is included in the attribution page. We found features that were highly correlated with the traits we tried to predict and included those in the neural network. The neural network We played with the iterations to manipulate the number of epochs starting at 100 to see the impact on the score we receive. With 100 we scored 0.12125, 250 received a better score of 0.135, and then 1000 received 0.117. We included a graph below that showed the relationship between the train and validation R^2 values and the number of epochs for the range: [1,1000]. From this graph, we concluded that there was no correlation between the number of epochs and the R^2 values. Thus, we chose the final number of epochs to be equal to 10. This is so we could continue to adjust other factors while not having bias for the R^2 values. We tried adjusting the learning rate from 0.001 to 0.01. After fewer epochs (10), the predictions scored 0.118. From research online, as well as through trial and error in our models, we found that the optimal learning rate is around 0.001, as is the default for the Adam optimizer in the PyTorch Python library. The final main factor that we tested for changes was the batch size. The original code had a batch size of 64, and we questioned if this was the optimal value. After performing the experiment, all other factors constant, we discovered that the optimal batch size is 32.

In addition to the Adam optimizer, we tried one other optimizer to see if that could improve our score. We ultimately settled on utilizing the ADAGrad optimization algorithm due to its relative relevance to our course work, the utilization of a gradient. After implementing this algorithm instead of Adam with similar hyperparameters we discovered that ADAGrad performed much worse. The reason for this poor performance, in comparison to Adam, is likely due to two main factors. To start ADAGrad takes adapts learning rates based on historical gradients while Adam adapts using the first and second moments of the gradient. This creates a much stronger learning rate adaptation. Secondly, Adam utilizes momentum while ADAGrad does not. This momentum factor helps the Adam algorithm to converge must faster and thus ADAGrad would need more epochs to reach a similar value. For the context of this problem, the time issue of ADAGrad when compared to Adam is significant as there is a lot of data to train and validate. Especially since we discovered that decreasing the batch size would be optimal, we need the program to run in the most time efficient and cost-efficient manner as we are already adding time for the decreased batch size. Overall, we realized that ADAGrad is essentially a suboptimal version of Adam and in this situation is impractical over the robustness and efficiency of the Adam algorithm.

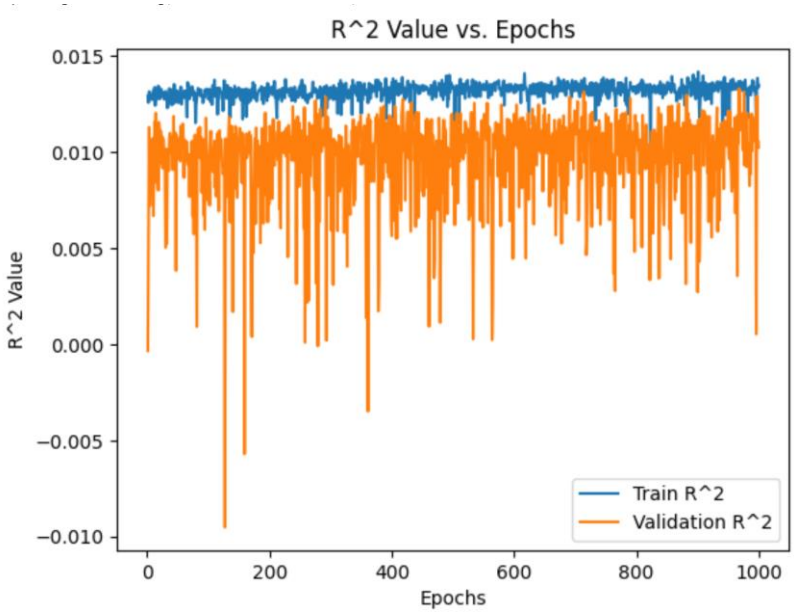

Fig. 3. R-Squared Scores vs Epochs

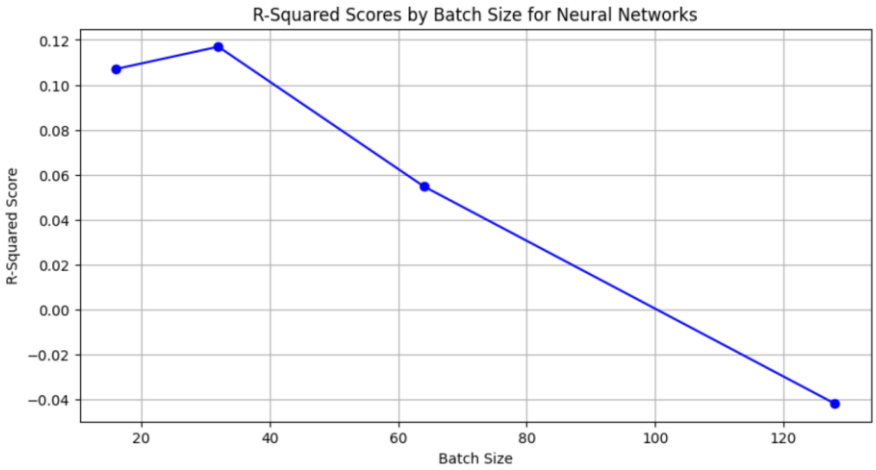

Fig. 4. R-Squared Scores vs Batch Size

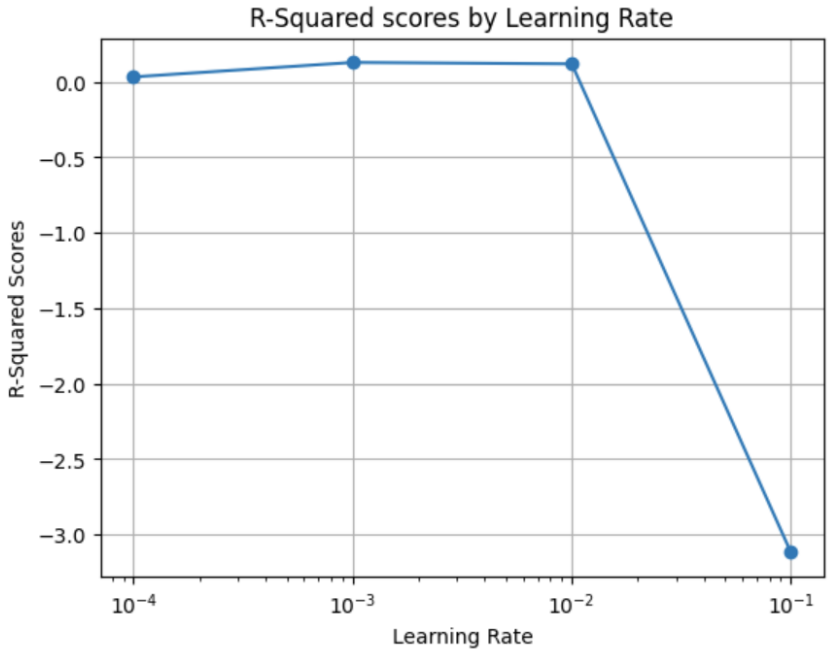

Fig. 5. R-Squared Scores vs Learning Rate

Using the maximum R^2 values based on our batch and learning rate scores, we were able to optimize our model one last time and receive a final score of 0.14086.

132	**Deb, Zach**		0.14086	10	37s

Your Best Entry!
Your most recent submission scored 0.14086, which is an improvement of your previous score of 0.13729. Great job!

Tweet this

3 Technical Assessment of Future Work

If we trained multiple neural networks to make predictions from the data set and then combine the results by taking their average would help to validate our model and reduce variance. Training multiple models will also allow us to experiment with parameters and the structure of neural networks. Changing the number of layers, filters, and tweaking the functions in addition to optimizing hyperparameters would become easier with multiple models. Their impacts could be interpreted and compared with more of a baseline of knowledge to allow for perpetual improvement. We also could have utilized a convolutional neural network, which self-engineers features by sweeping through images, detecting patterns. This, as well as extra "pooling" layers allow the CNN to better identify edges and important parts of images.

Improved data cleaning from the start would be a way to directly improve results right from the source. Because this dataset is so large, it is computationally intensive and therefore challenging to standardize the data to ensure quality. In doing so, we could remove the outliers and rogue images that may be missing or compromised that may, as a result, be skewing the data.

Adding regularization techniques into the code including L1/L2 regularization and batch normalization to manage the data and prevent overfitting would be a way to further improve model performance. With less overall noise being interpreted by machine learning, the model's results will improve.

Additionally, iterating through different batch sizes and different learning rates systematically to optimize these hyperparameters to increase R-squared would be a good approach for improving predictions. In our project, we chose these hyperparameters around the defaults, to see if we could improve the scores. Optimizing it through code instead of picking values by hand will allow us to zero in on the optimal values.

4 Attributions

1. https://www.kaggle.com/competitions/planttraits2024/overview
2. https://www.kaggle.com/code/astitwaagarwal/planttraits
   https://www.w3schools.com/python/python_try_except.asp
3. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
4. https://pytorch.org/docs/stable/generated/torch.optim.SGD.html
5. https://docs.python.org/3/library/os.html
6. https://pillow.readthedocs.io/en/stable/
7. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
8. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html
9. https://pytorch.org/vision/
10. https://pytorch.org/docs/stable/optim.html
11. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
12. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
13. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html
14. https://pillow.readthedocs.io/en/stable/reference/Image.html
15. https://www.geeksforgeeks.org/python-os-path-join-method/
16. https://chat.openai.com/
17. Github Copilot