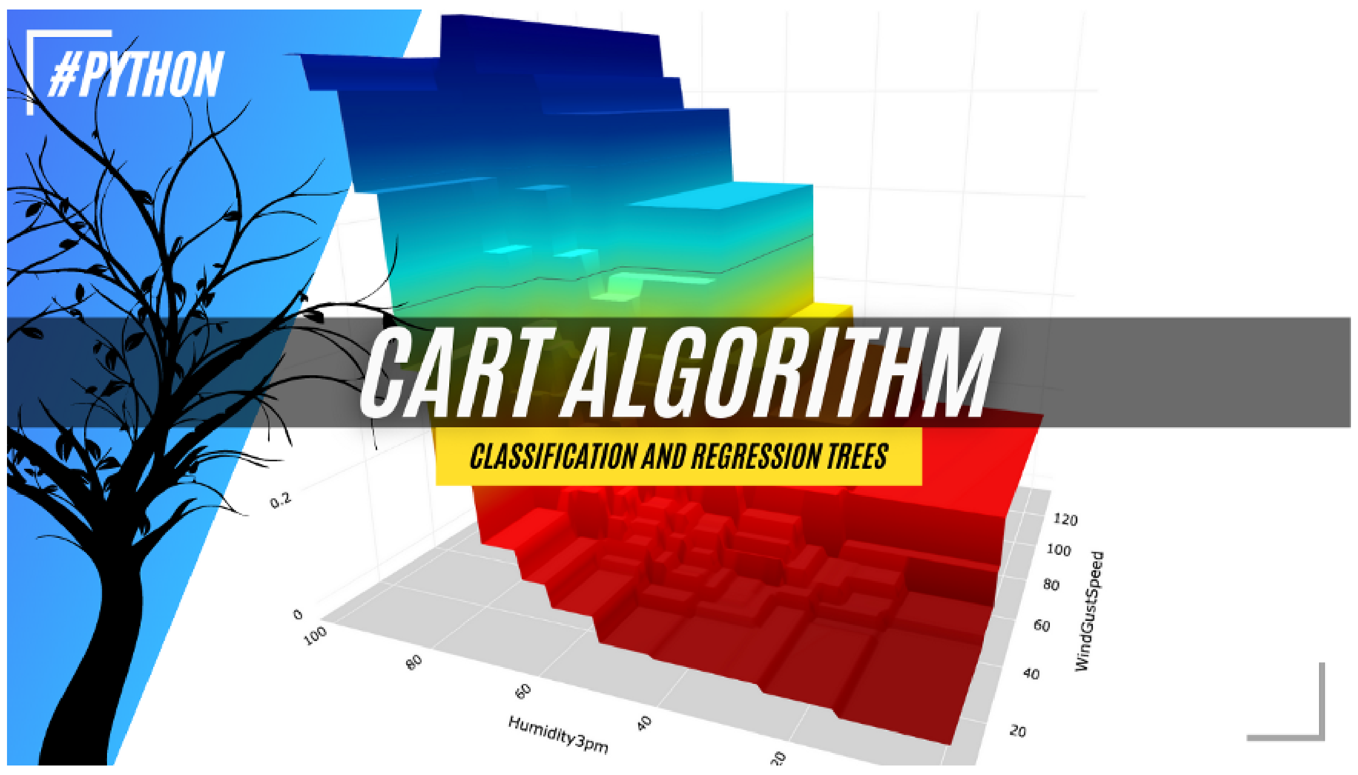
Published in Towards Data Science

Saul Dobilas
Jan 31, 2021 · 9 min read ★ · Listen

GETTING STARTED, MACHINE LEARNING

CART: Classification and Regression Trees for Clean but Powerful Models

How does the CART algorithm work, and how to successfully use it in Python?



CART model prediction surface. See how the chart was made in the Python section at the end of this story. Image by author.

Intro

If you want to be a successful Data Scientist, it is essential to understand how different Machine Learning algorithms work.

This story is part of the series that explains the nuances of each algorithm and provides a range of Python examples to help you build your own ML models. Not to mention some cool 3D visualizations!

The story covers the following topics:

- The category of algorithms that CART belongs to
- An explanation of how the CART algorithm works
- Python examples on how to build a CART Decision Tree model

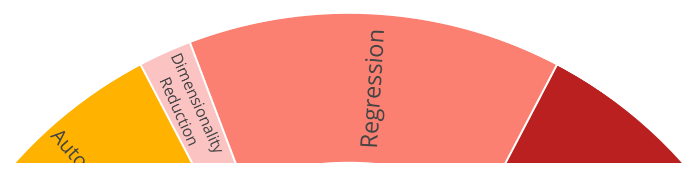
What category of algorithms does CART belong to?

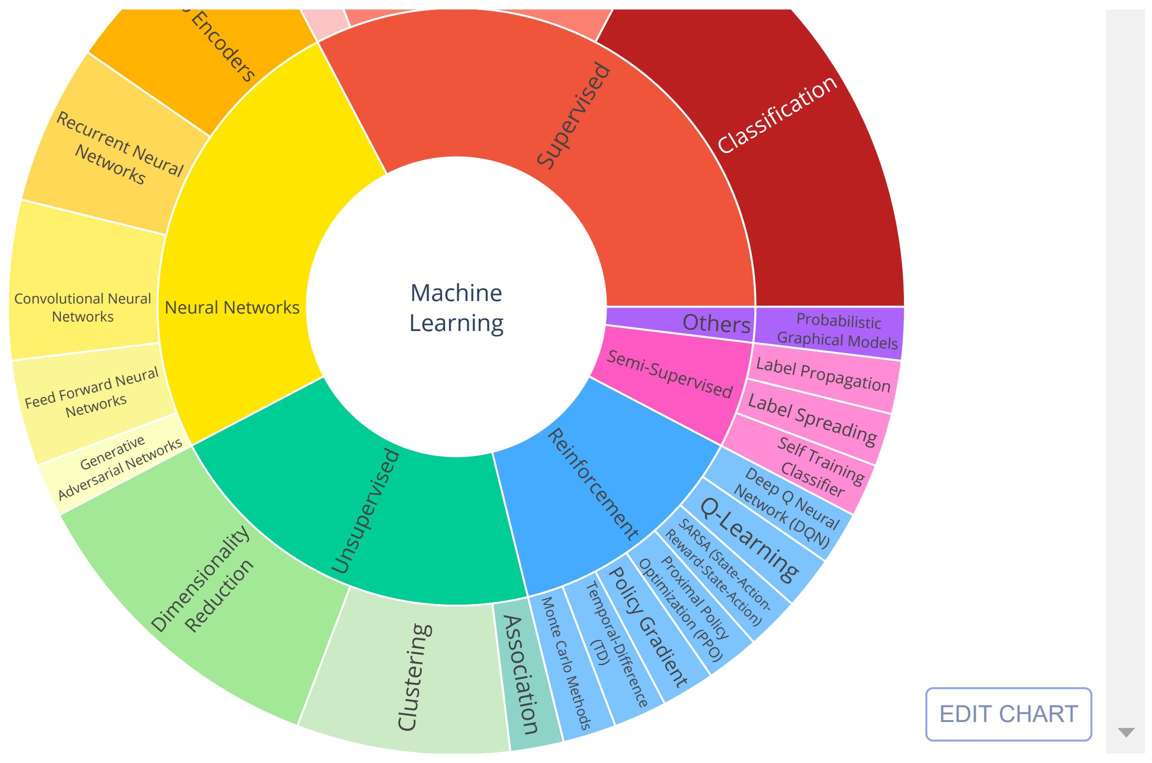
As the name suggests, CART (Classification and Regression Trees) can be used for both classification and regression problems. The difference lies in the target variable:

- With **classification**, we attempt to predict a class label. In other words, classification is used for problems where the output (target variable) takes a finite set of values, e.g., whether it will rain tomorrow or not.
- Meanwhile, **regression** is used to predict a numerical label. This means your output can take an infinite set of values, e.g., a house price.

Both cases fall under the supervised branch of machine learning algorithms.

> *Side note, I have put Neural Networks in a category of their own due to their unique approach to Machine Learning. However, they can be used to solve a wide range of problems, including but not limited to classification and regression. The below chart is* ***interactive*** *so make sure to click 👇 on different categories to* ***enlarge and reveal more****.*





Machine Learning algorithm classification. Interactive chart created by the author.

***If you share a passion for Data Science and Machine Learning**, please subscribe to receive an email whenever I publish a new story.*

While in this story, I focus on CART for classification, the regression case is very similar except for using a different method to calculate the best splits in the tree.

How do classification and regression trees work?

Example

Let’s start with a simple example. Assume you have a bunch of oranges and mandrins with labels on them, and you want to identify a set of simple rules that you can use in the future to distinguish between these two types of fruit.

Photo by Philippe Gauthier on Unsplash

Typically, oranges (diameter 6–10cm) are bigger than mandarins (diameter 4–8cm), so the first rule found by your algorithm might be based on size:

- Diameter ≤ 7cm.

Next, you may notice that mandarins tend to be slightly darker in color than oranges. So, you use a color scale (1=dark to 10=light) to split your tree further:

- Color ≤5 for the left side of the sub-tree
- Color ≤6 for the right side of the sub-tree

Your final result is a tree that consists of 3 simple rules that help you to correctly distinguish between oranges and mandarins in the majority of the cases:

199 | 3 | •••

Follow

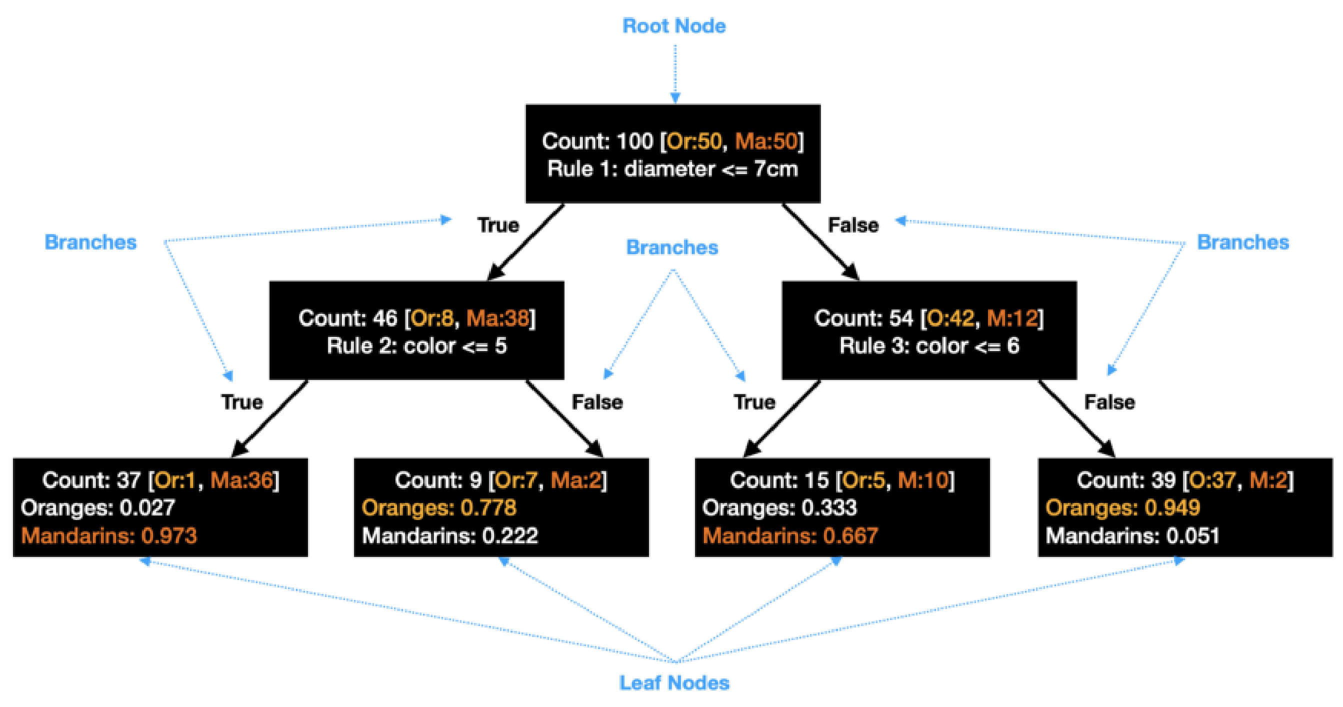
More from Medium

Alexand... in Tow...
An In-depth Guide to Local Outlier Factor (LOF) for...

Rya... in Towards...
One-dimensional CNN for human behavior...

A... in Becomin...
XGBoost Hyperparameters Tuning using...

Must... in Towar...
Decision Tree Models using Python — Build,...



Decision Tree for identifying oranges vs. mandarins. Image by author.

How does CART find the best split?

Several methods can be used in CART to identify the best splits. Here are two of the most common ones for classification trees:

Gini Impurity

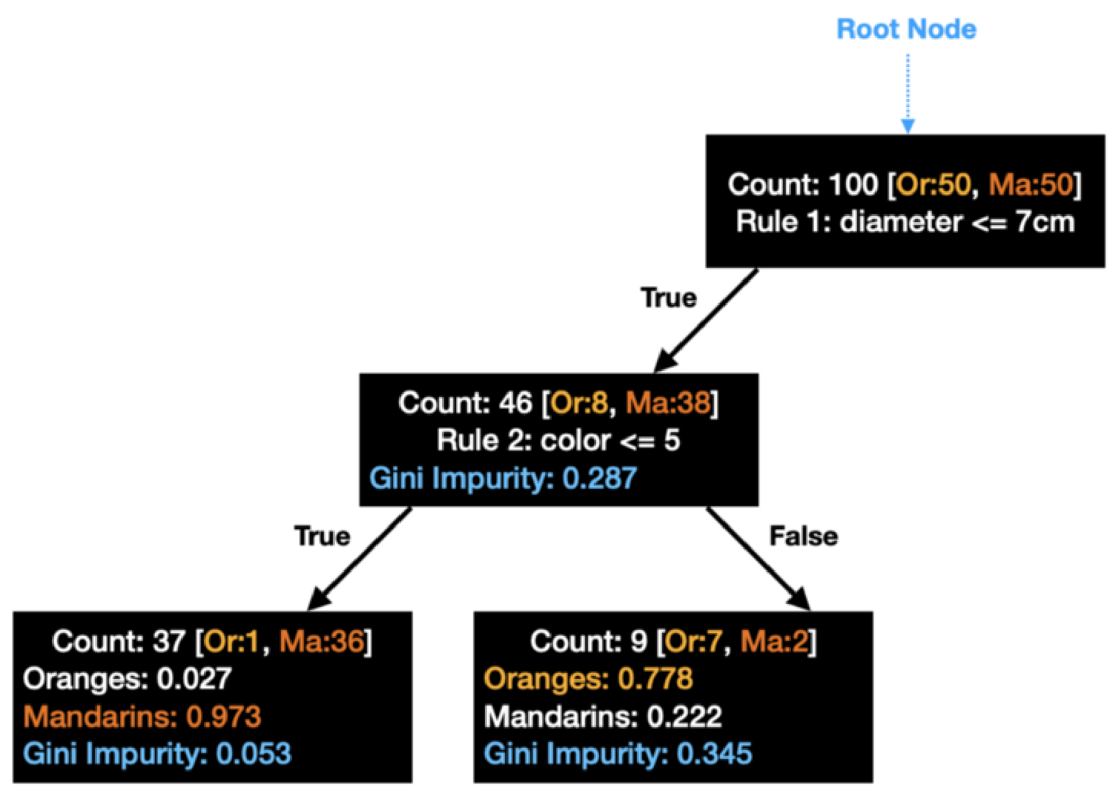
$$Gini\ Impurity = 1 - Gini = 1 - \sum_{i=1}^{n} p_i^2$$

```
where p_i is the fraction of items in the class i.
```

Using the above tree as an example, Gini Impurity for the leftmost leaf node would be:

```
1 - (0.027^2 + 0.973^2) = 0.053
```

To find the best split, we need to calculate the weighted sum of Gini Impurity for both child nodes. We do this for all possible splits and then take the one with the **lowest Gini Impurity** as the best split.



The weighted sum of the Gini Impurity for the two child nodes is:

$$Gini\ Impurity = \frac{37}{46} * 0.053 + \frac{9}{46} * 0.345 = 0.110$$

Calculating Gini Impurity. Image by author.

> ***Important note:*** *If the best weighted Gini Impurity for the two child nodes is* ***not lower*** *than Gini Impurity for the parent node, you should not split the parent node any further.*

Entropy

The entropy approach is essentially the same as Gini Impurity, except it uses a slightly different formula:

$$Entropy = -\sum_{i=1}^{n} p_i \, log_2(p_i)$$

To identify the best split, you would have to follow all the same steps outlined above. The split with the lowest entropy is the best one. Similarly, if the entropy of the two child nodes is not lower than that of a parent node, you should not split any further.

How to build CART Decision Tree models in Python?

We will build a couple of classification decision trees and use tree diagrams and 3D surface plots to visualize model results. First, let's do some basic setup.

Setup

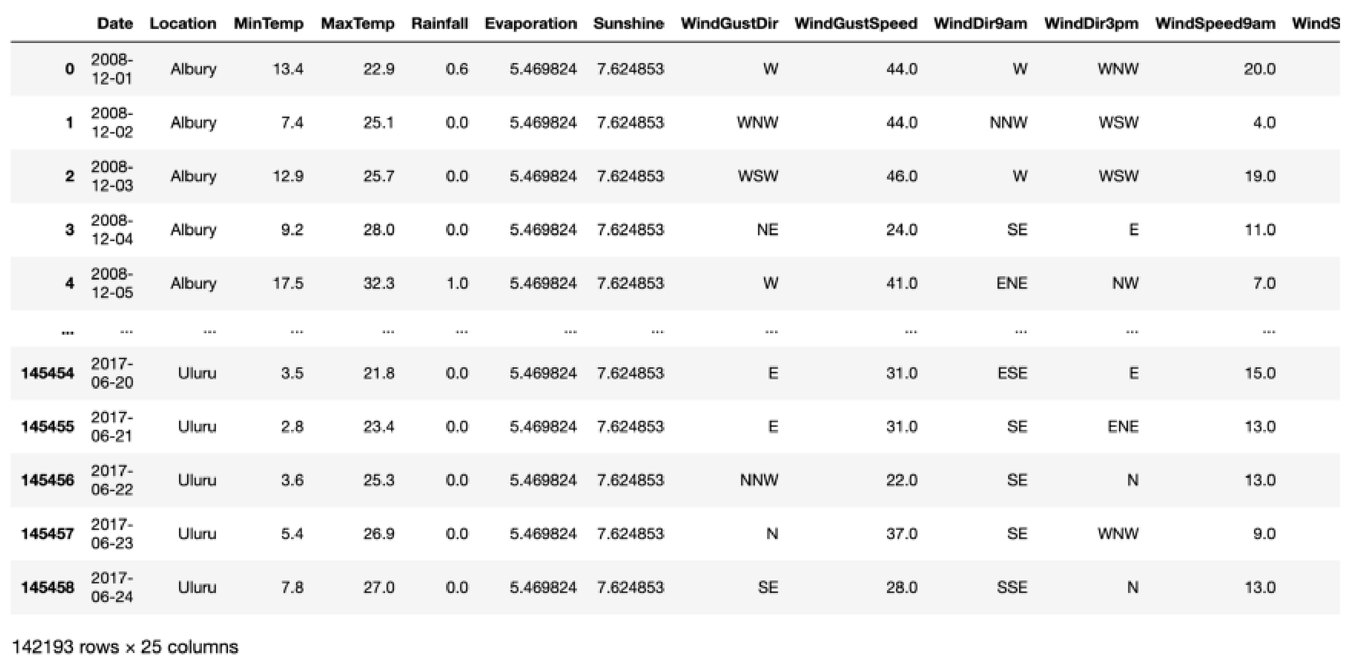
We will use the following data and libraries:

- Australian weather data from Kaggle
- Scikit-learn library for splitting the data into train-test samples, building CART classification models, and model evaluation
- Plotly for data visualizations
- Pandas and Numpy for data manipulation
- Graphviz library to plot decision tree graphs

Let's import all the libraries:

Then we get the Australian weather data from Kaggle, which you can download following this link: https://www.kaggle.com/jsphyg/weather-dataset-rattle-package.

Once you have saved the data on your machine, ingest it with the below code. Note, we also do some simple data manipulation and derive a few new variables for later use in our models.



	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	WindS
0	2008-12-01	Albury	13.4	22.9	0.6	5.469824	7.624853	W	44.0	W	WNW	20.0	
1	2008-12-02	Albury	7.4	25.1	0.0	5.469824	7.624853	WNW	44.0	NNW	WSW	4.0	
2	2008-12-03	Albury	12.9	25.7	0.0	5.469824	7.624853	WSW	46.0	W	WSW	19.0	
3	2008-12-04	Albury	9.2	28.0	0.0	5.469824	7.624853	NE	24.0	SE	E	11.0	
4	2008-12-05	Albury	17.5	32.3	1.0	5.469824	7.624853	W	41.0	ENE	NW	7.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
145454	2017-06-20	Uluru	3.5	21.8	0.0	5.469824	7.624853	E	31.0	ESE	E	15.0	
145455	2017-06-21	Uluru	2.8	23.4	0.0	5.469824	7.624853	E	31.0	SE	ENE	13.0	
145456	2017-06-22	Uluru	3.6	25.3	0.0	5.469824	7.624853	NNW	22.0	SE	N	13.0	
145457	2017-06-23	Uluru	5.4	26.9	0.0	5.469824	7.624853	N	37.0	SE	WNW	9.0	
145458	2017-06-24	Uluru	7.8	27.0	0.0	5.469824	7.624853	SE	28.0	SSE	N	13.0	

142193 rows × 25 columns

A snippet of Kaggle's Australian weather data with some modifications. Image by author.

To reduce the amount of repeated code, we will create a couple of functions that we can reuse throughout the analysis.

This first function performs the following actions:

- Splits the data into train and test samples
- Fits the model
- Predicts the label on a test set
- Generates model performance evaluation metrics
- Creates a decision tree graph

The second function will be used to plot 3D scatter graphs with the test data and model prediction surface:

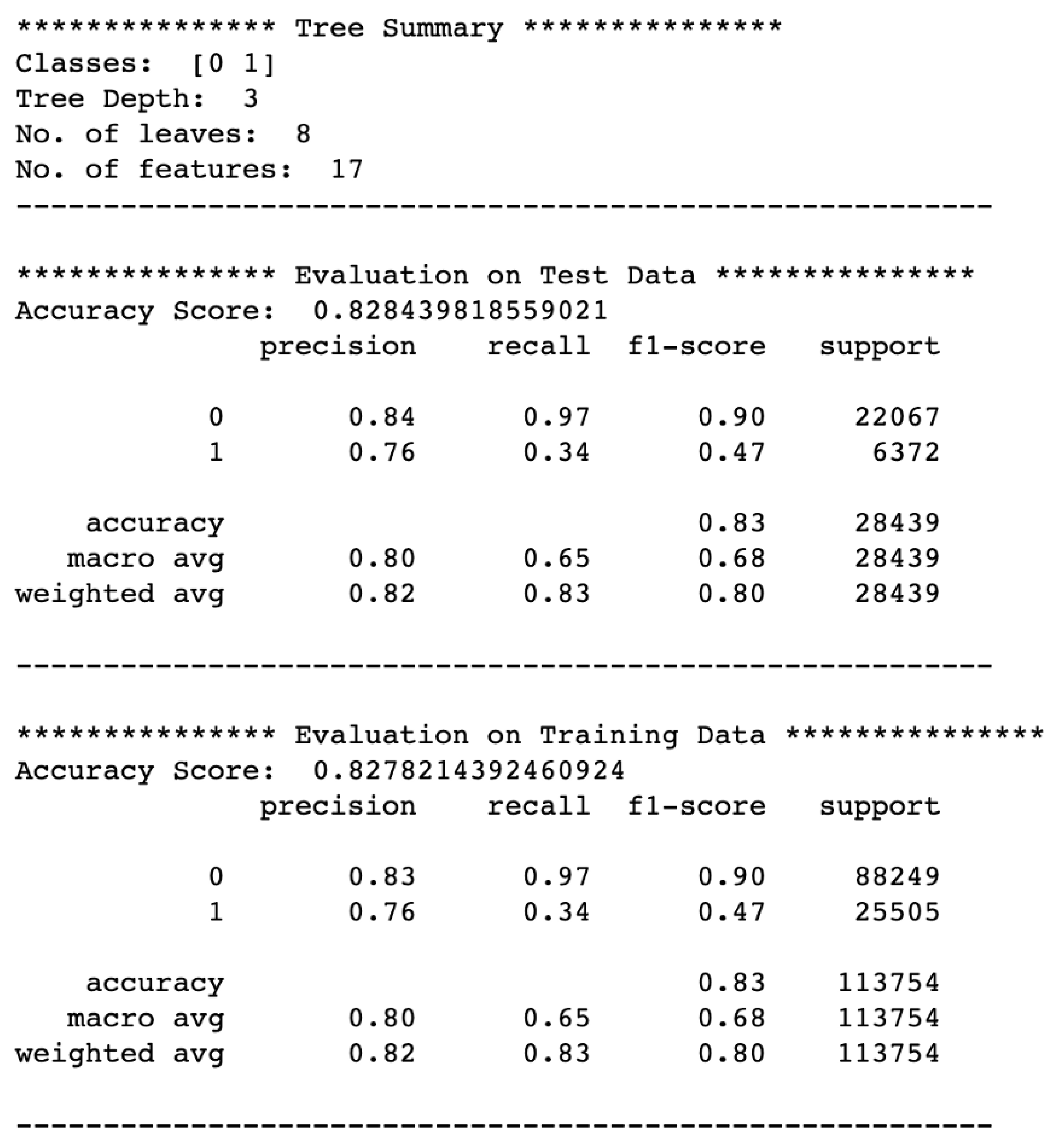
CART classification model using Gini Impurity

Our first model will use all numerical variables available as model features. Meanwhile, **RainTomorrowFlag** will be the target variable for all models.

Note, at the time of writing sklearn's **tree.DecisionTreeClassifier()** can only take numerical variables as features. However, you can also use categorical ones as long as you encode them with an encoding algorithm such as **sklearn's Ordinal Encoder** or any other appropriate method to convert categorical values into numerical ones.

Let's use our ***fitting*** function to build the model with the tree depth limited to 3 and minimum leaf size being 1,000 observations. Limiting tree depth and leaf size help us to avoid overfitting. In a later example, we will look at how much tree complexity increases once we remove some of those constraints.

Here is the output generated by the ***fitting*** function:



```
*************** Tree Summary ***************
Classes:  [0 1]
Tree Depth:  3
No. of leaves:  8
No. of features:  17
--------------------------------------------------------

*************** Evaluation on Test Data ***************
Accuracy Score:  0.828439818559021
              precision    recall  f1-score   support

           0       0.84      0.97      0.90     22067
           1       0.76      0.34      0.47      6372

    accuracy                           0.83     28439
   macro avg       0.80      0.65      0.68     28439
weighted avg       0.82      0.83      0.80     28439

--------------------------------------------------------

*************** Evaluation on Training Data ***************
Accuracy Score:  0.8278214392460924
              precision    recall  f1-score   support

           0       0.83      0.97      0.90     88249
           1       0.76      0.34      0.47     25505

    accuracy                           0.83    113754
   macro avg       0.80      0.65      0.68    113754
weighted avg       0.82      0.83      0.80    113754

--------------------------------------------------------
```

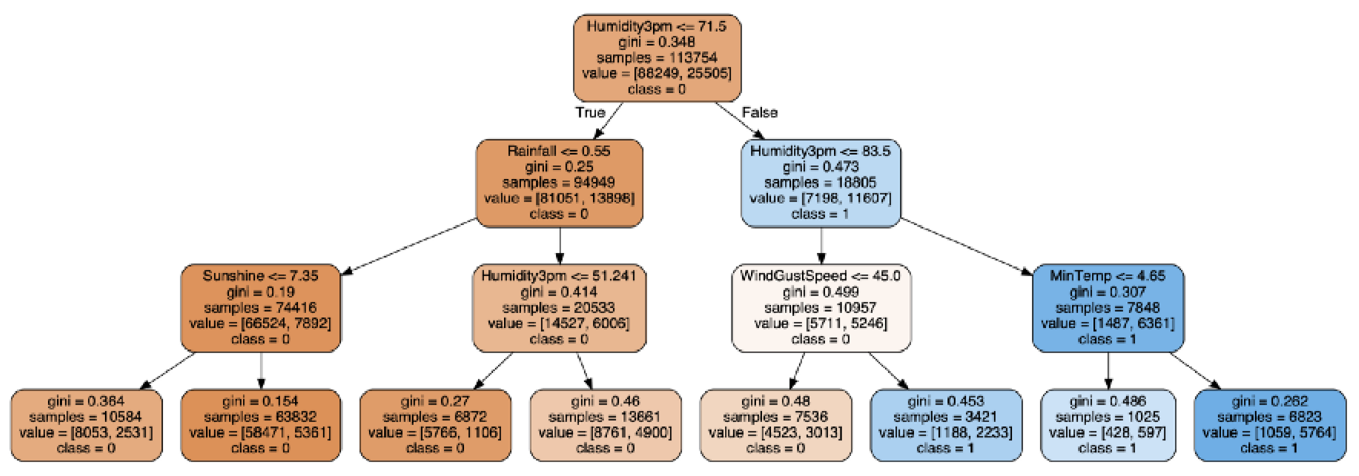
Model 1 — CART model performance metrics. Image by author.

We can see that the model performs relatively well in predicting dry days. However, the performance is worse on predicting rainy days, with precision on test data being 0.76 and recall 0.34.

- **Precision** means that it will actually rain tomorrow in 76% of those cases where the model predicts a rainy day.
- Meanwhile, **recall** means that for all the rainy days in the test data, the model only identified 34% of them.

The difference in performance across the two class labels is largely driven by an imbalance in the data, with many more dry days available than rainy days.

Next, let’s look at the **tree graph** generated by our ***fitting*** function:



Model 1 — CART model decision tree. Image by author.

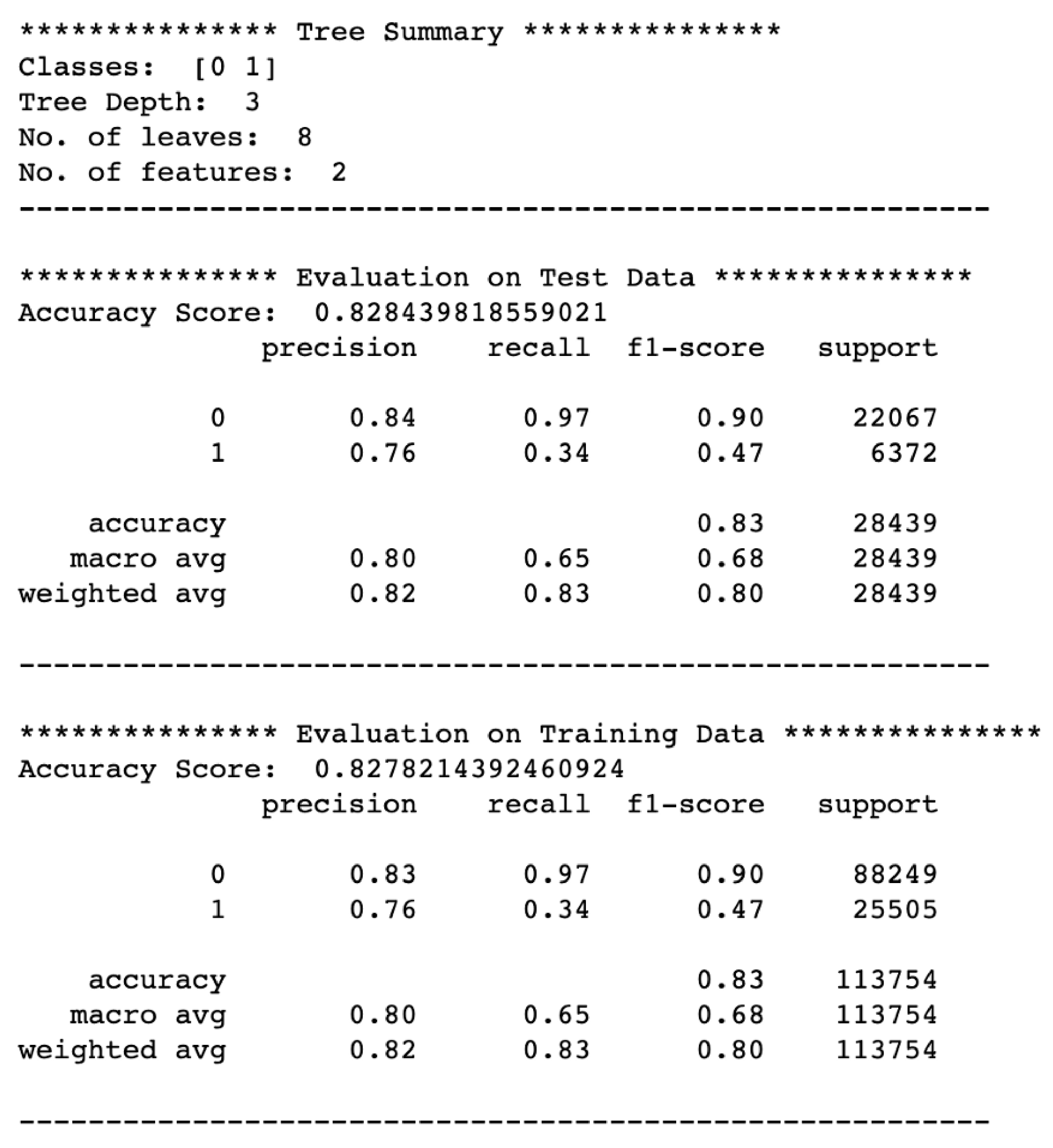
Looking at the above, we can see that while the algorithm used several different features, the two most important ones were “Humidity3pm” and “WindGustSpeed,” as these are the only two that influenced class label prediction being 0 or 1.

Hence, we can create a model with a similar performance by reducing the number of features, as shown in the next example.

CART classification model using Gini Impurity and 2 features

Let’s use our ***fitting*** function again:

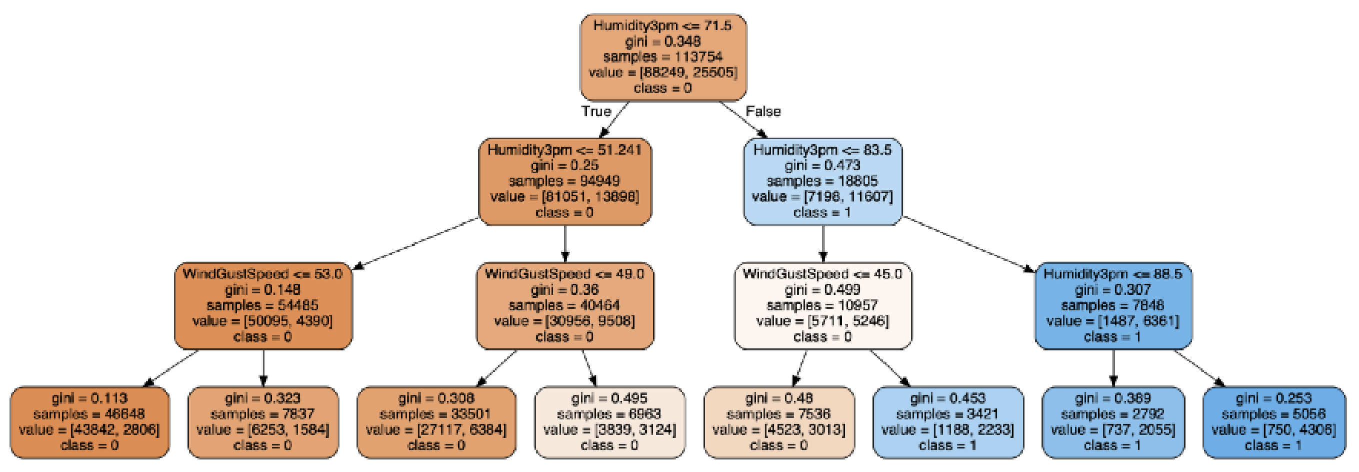
Model performance metrics:



```
*************** Tree Summary ***************
Classes:  [0 1]
Tree Depth:  3
No. of leaves:  8
No. of features:  2
--------------------------------------------------------

*************** Evaluation on Test Data ***************
Accuracy Score:  0.828439818559021
              precision    recall  f1-score   support

           0       0.84      0.97      0.90     22067
           1       0.76      0.34      0.47      6372

    accuracy                           0.83     28439
   macro avg       0.80      0.65      0.68     28439
weighted avg       0.82      0.83      0.80     28439

--------------------------------------------------------

*************** Evaluation on Training Data ***************
Accuracy Score:  0.8278214392460924
              precision    recall  f1-score   support

           0       0.83      0.97      0.90     88249
           1       0.76      0.34      0.47     25505

    accuracy                           0.83    113754
   macro avg       0.80      0.65      0.68    113754
weighted avg       0.82      0.83      0.80    113754

--------------------------------------------------------
```

Model 2 — CART model performance metrics. Image by author.

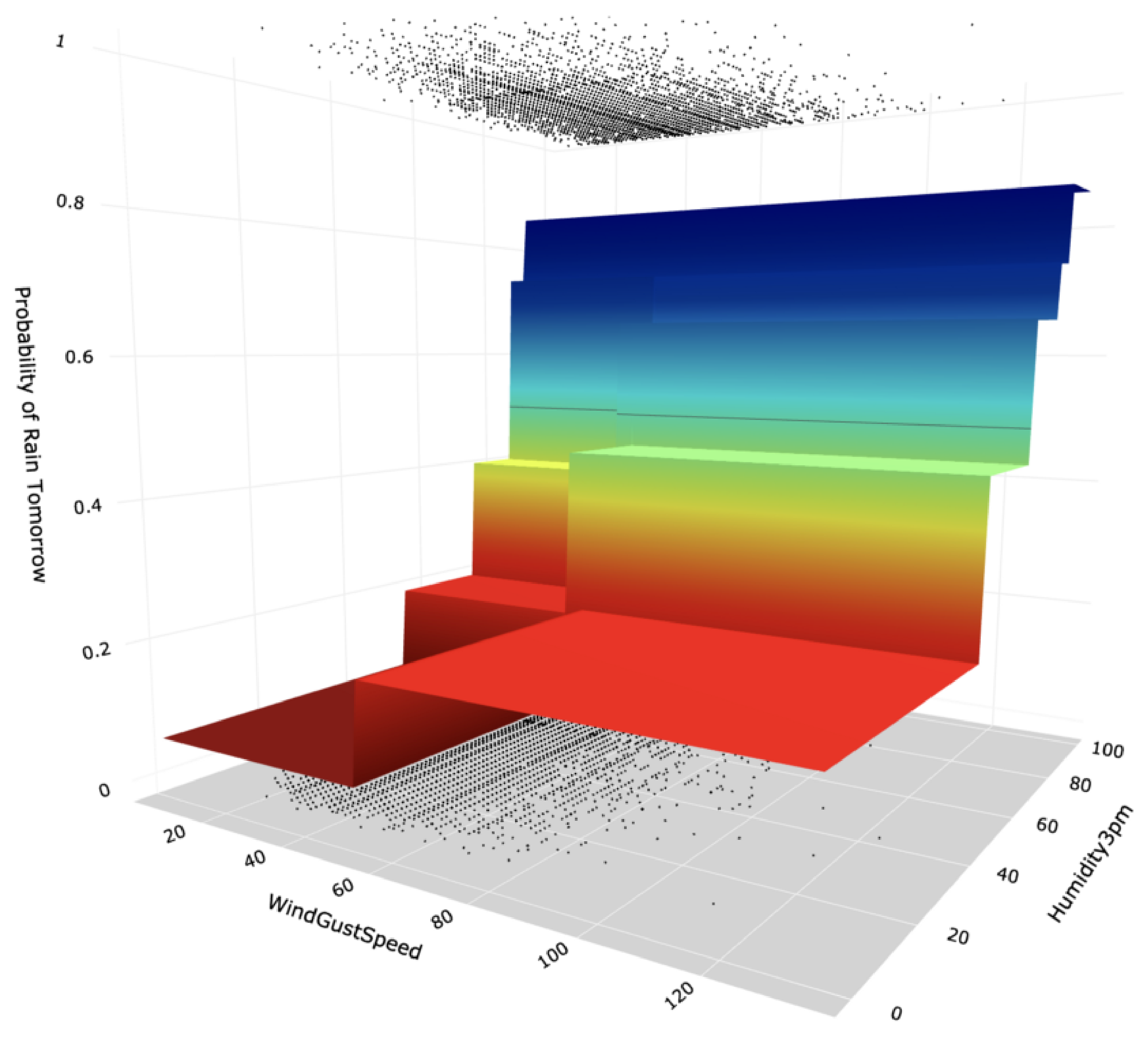
As expected, the model performance is identical to the first model. However, let's take a look at how the decision tree changed:



Model 2— CART model decision tree. Image by author.

So, while the tree is different in places, the key splits remained the same.

The best thing is that we can create a 3D chart to visualize the prediction plane since we only used two input features. That's where the second function, ***Plot_3D,*** comes in handy:



Model 2 — CART model prediction surface. Image by author.

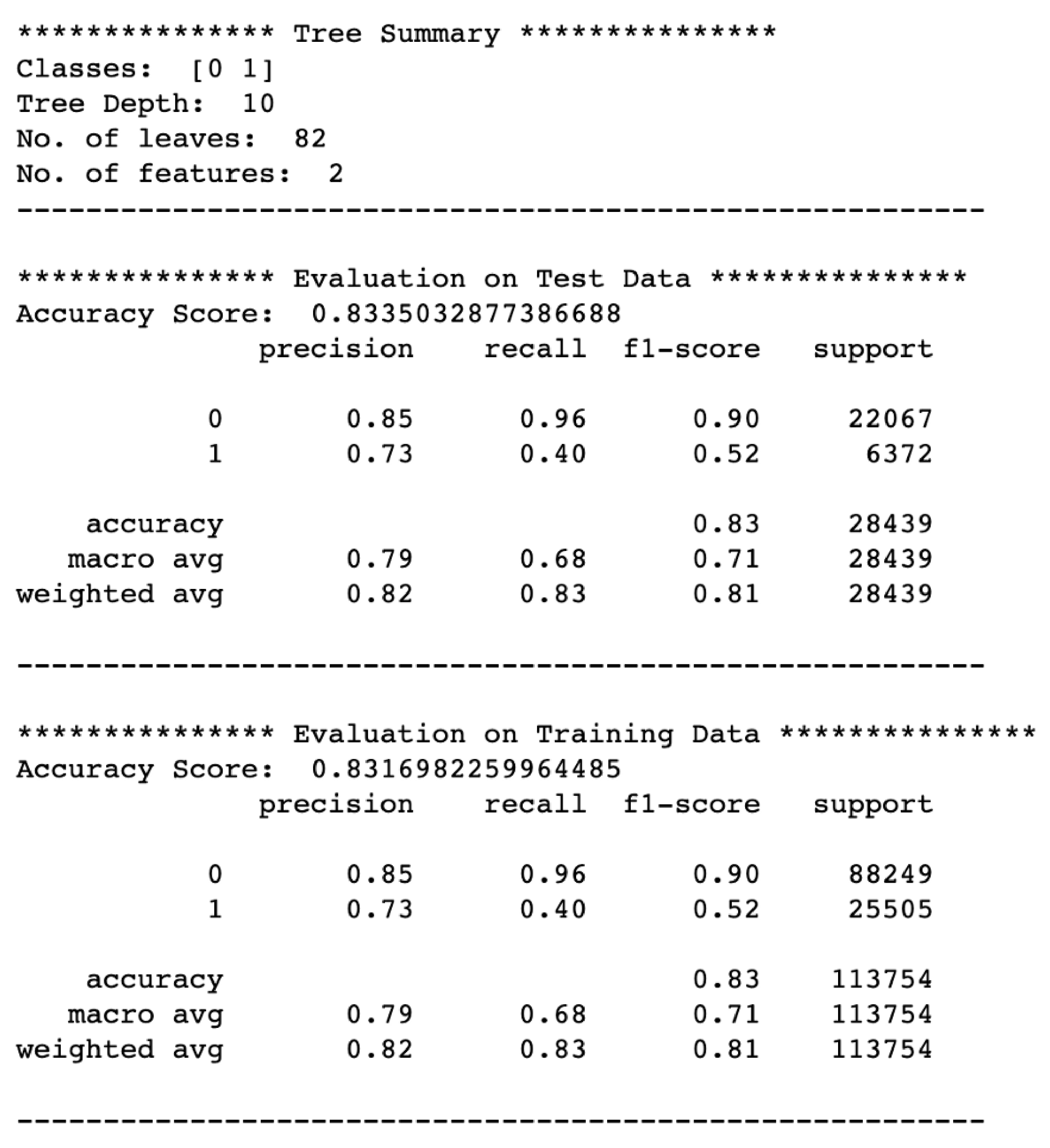
Note, black points at the top are instances of class=1 (Rain Tomorrow), and the ones at the bottom are instances of class=0 (No Rain Tomorrow). Meanwhile, the surface is the probability of rain tomorrow based on the model's prediction. Finally, the thin line in the middle of the graph is probability=0.5, which denotes the decision boundary.

To little surprise, the prediction plane looks like a set of stairs. This is because the prediction probability follows step changes at specific values used to split the tree nodes. E.g., the lowest rain probability (bottom step — dark red) is bounded by "Humidity3pm = 51.241" and "WindGustSpeed = 53.0."

CART classification model with unlimited tree depth

Now let's see what happens when we do not restrict the tree depth. We use our ***fitting*** function again:

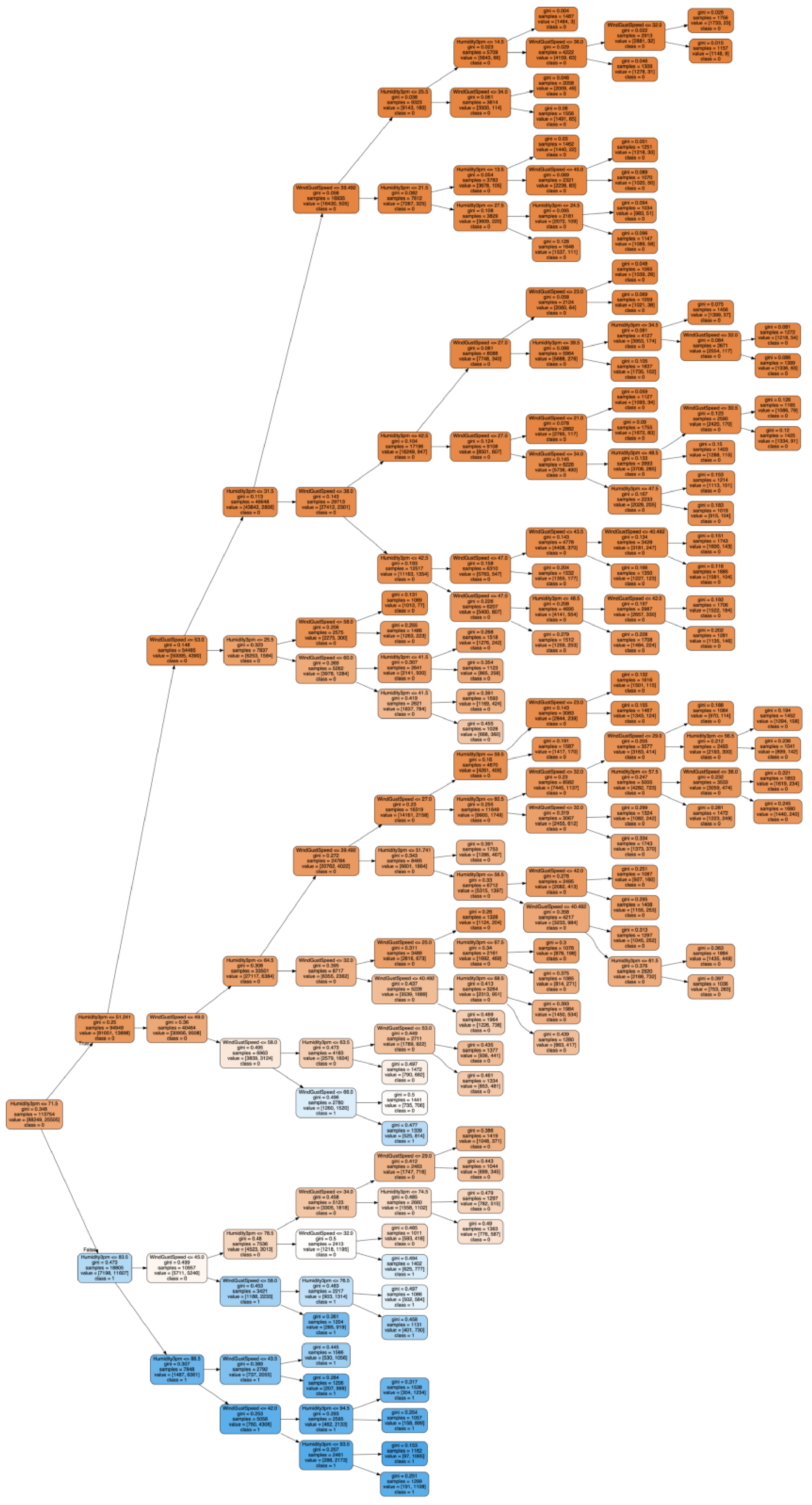
Here is the resulting model performance:



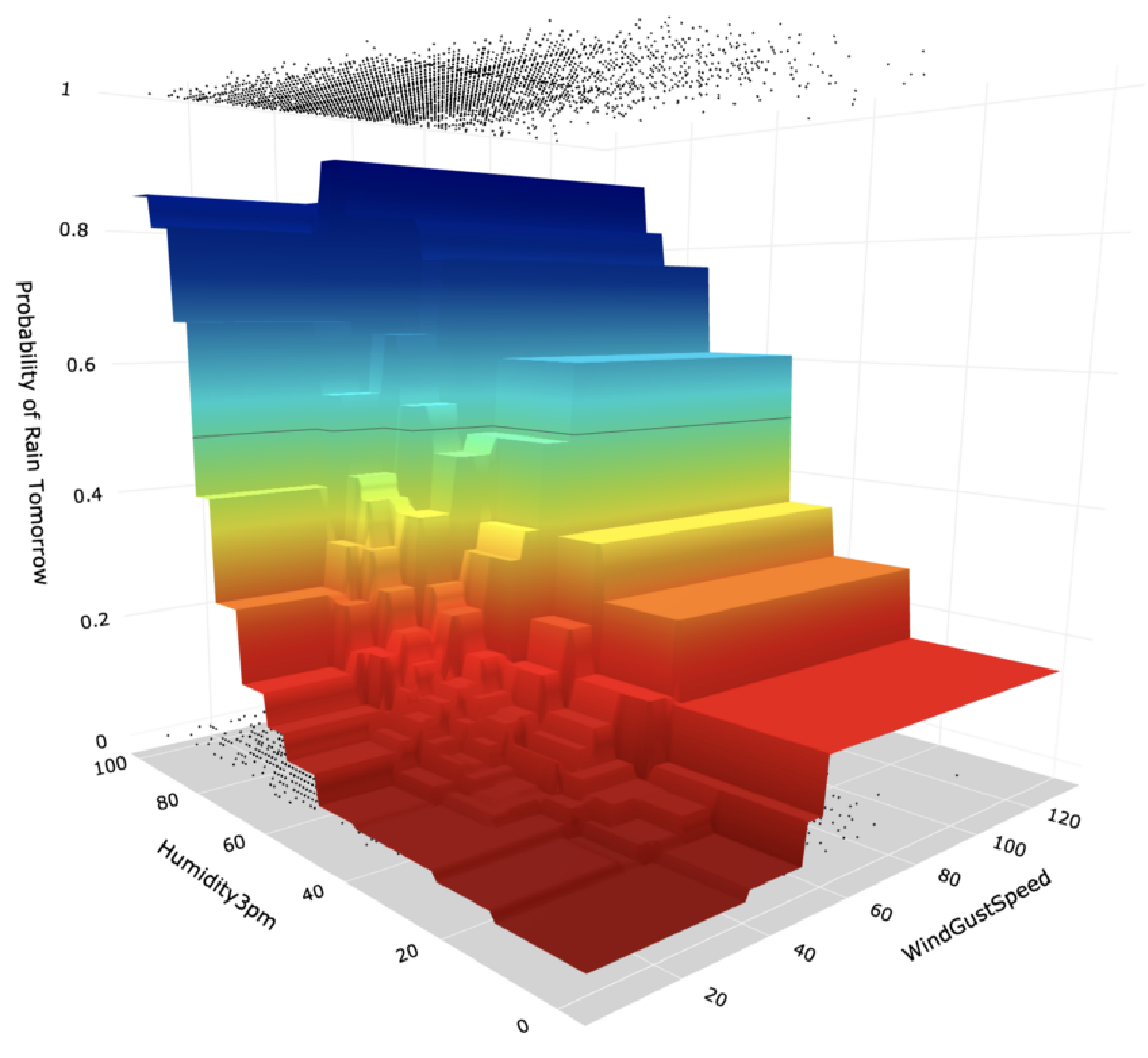
```
*************** Tree Summary ***************
Classes:  [0 1]
Tree Depth:  10
No. of leaves:  82
No. of features:  2
--------------------------------------------------------

*************** Evaluation on Test Data ***************
Accuracy Score:  0.8335032877386688
              precision    recall  f1-score   support

           0       0.85      0.96      0.90     22067
           1       0.73      0.40      0.52      6372

    accuracy                           0.83     28439
   macro avg       0.79      0.68      0.71     28439
weighted avg       0.82      0.83      0.81     28439

--------------------------------------------------------

*************** Evaluation on Training Data ***************
Accuracy Score:  0.8316982259964485
              precision    recall  f1-score   support

           0       0.85      0.96      0.90     88249
           1       0.73      0.40      0.52     25505

    accuracy                           0.83    113754
   macro avg       0.79      0.68      0.71    113754
weighted avg       0.82      0.83      0.81    113754

--------------------------------------------------------
```

Model 3— CART model performance metrics (with max depth). Image by author.

Decision tree (note, the tree has been rotated for a better fit on the page):


Model 3 — CART model decision tree (with max depth). Image by author.

Finally, the 3D graph:



Model 3— CART model prediction surface (with max depth). Image by author.

As you can see, with no limit on tree depth, the algorithm has created a much more complex tree, which can be seen in both the tree diagram and the amount of “steps” on a 3D prediction surface. Simultaneously, the model's performance is only marginally better (accuracy=0.83).

Whenever you build decision tree models, you should carefully consider the trade-off between complexity and performance. In this specific example, a tiny increase in performance is not worth the extra complexity.

Other things to explore

There are many ways you can further fine-tune your CART models. A few to mention:

- You can change 'gini' to 'entropy' to build a model using an entropy-based algorithm.
- You can use a 'random' splitter instead of 'best.' 'Best' always takes the feature with the highest importance to produce the next split. Meanwhile, 'random' would select a random feature (although weighted by the feature importance distribution).
- As demonstrated above, you can change the maximum allowed depth for the tree.
- You can adjust 'class_weight' (named ***clweight*** in our ***fitting*** function) by passing a dictionary with weights for each class or simply putting in 'balanced' for the algorithm to balance out class samples using weighting.
- Finally, you can also try adjusting the minimum leaf size.

Conclusion

CART is a powerful algorithm that is also relatively easy to explain compared to other ML approaches. It does not require much computing power, hence allowing you to build models very fast.

While you need to be careful not to overfit your data, it is a good algorithm for simple problems. If you are looking to improve your models' performance and robustness, you can also explore ensemble methods, such as a **random forest.**

As always, drop me a line if you enjoyed learning about the decision trees or had any questions or suggestions.

Cheers 👏
Saul Dobilas

If you have already spent your learning budget for this month, please remember me next time. *My personalized link to join Medium is:*

Join Medium with my referral link - Saul Dobilas
As a Medium member, a portion of your membership fee goes to writers you read, and you get full access to every story...
solclover.com

Other classification algorithms you may be interested in:

Random Forest Models: Why Are They Better Than Single Decision Trees?
A detailed explanation of how random forest machine learning algorithm works, what makes it superior to decision trees...
towardsdatascience.com

Gradient Boosted Trees for Classification — One of the Best Machine Learning Algorithms
A step by step guide to how Gradient Boosting works in classification trees
towardsdatascience.com

XGBoost: Extreme Gradient Boosting — How to Improve on Regular Gradient Boosting?
A detailed look at differences between the two algorithms and when you should choose one over the other
towardsdatascience.com