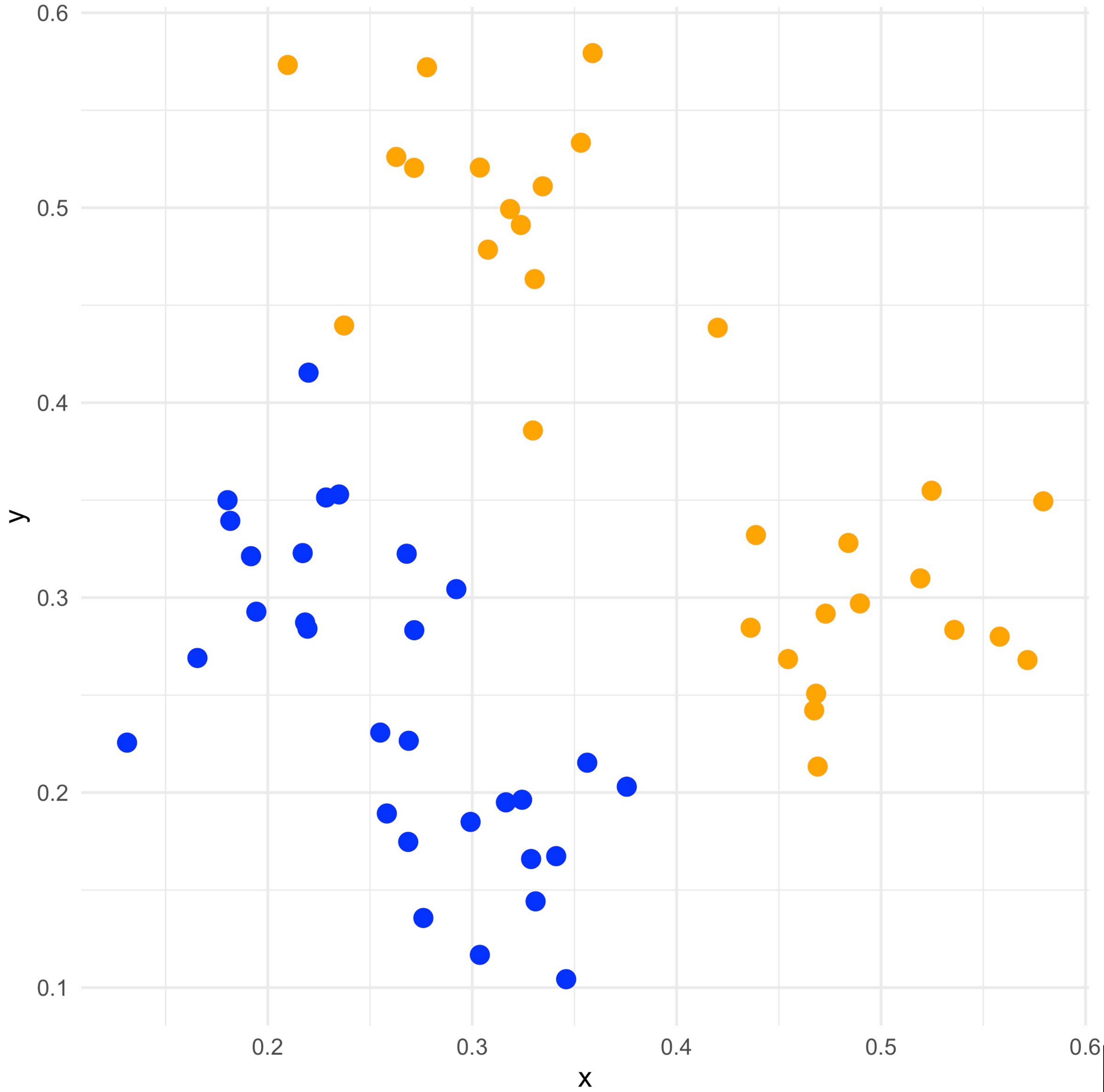
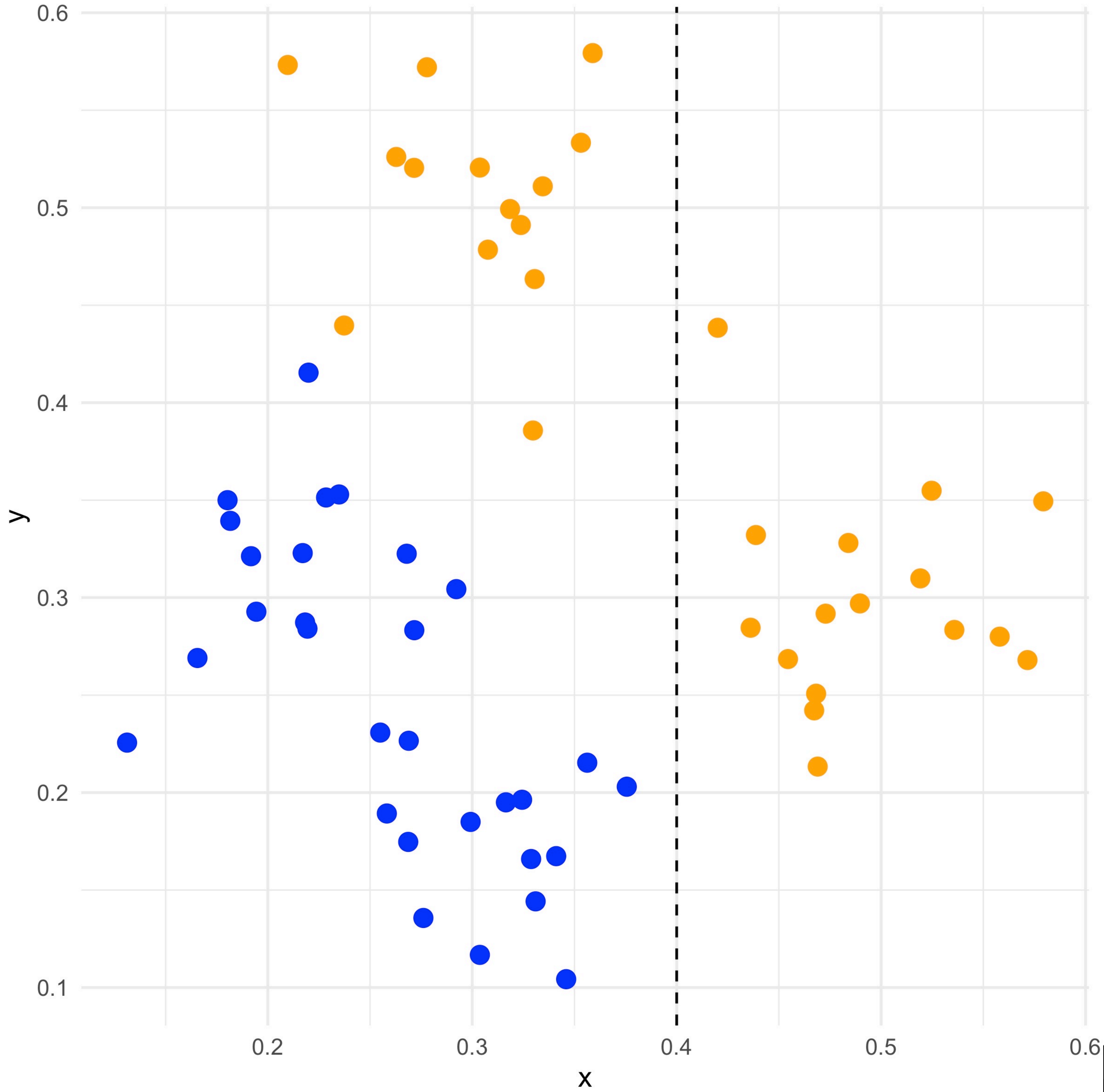
Decision Tree

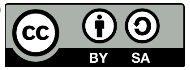
Let’s consider our two-class classification task once again.



Decision Tree

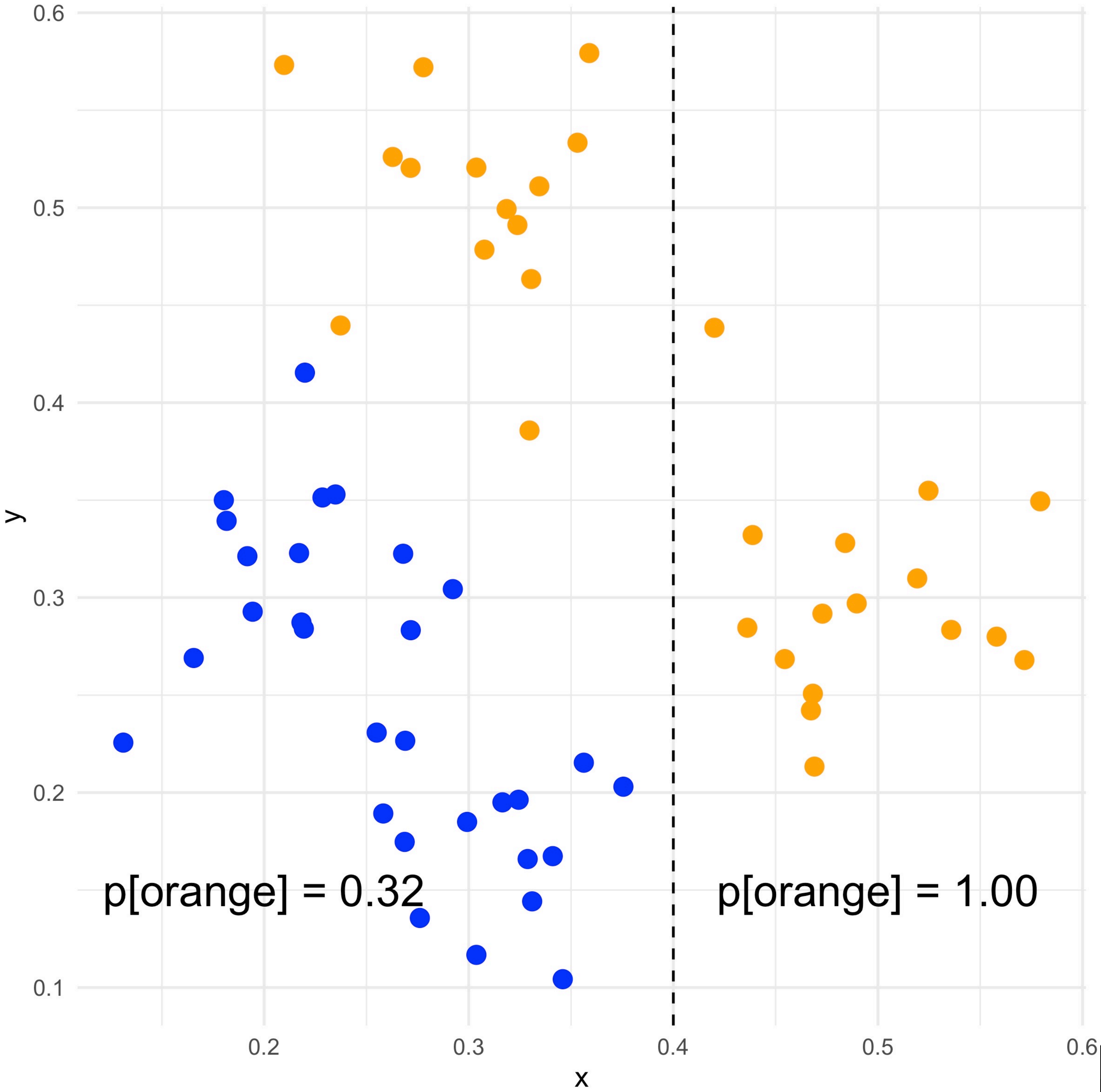
What if we return to the idea of a dividing line, but this time with two restrictions: it must be perpendicular to one axis and the line will define a fixed probability on each side of it.





Decision Tree

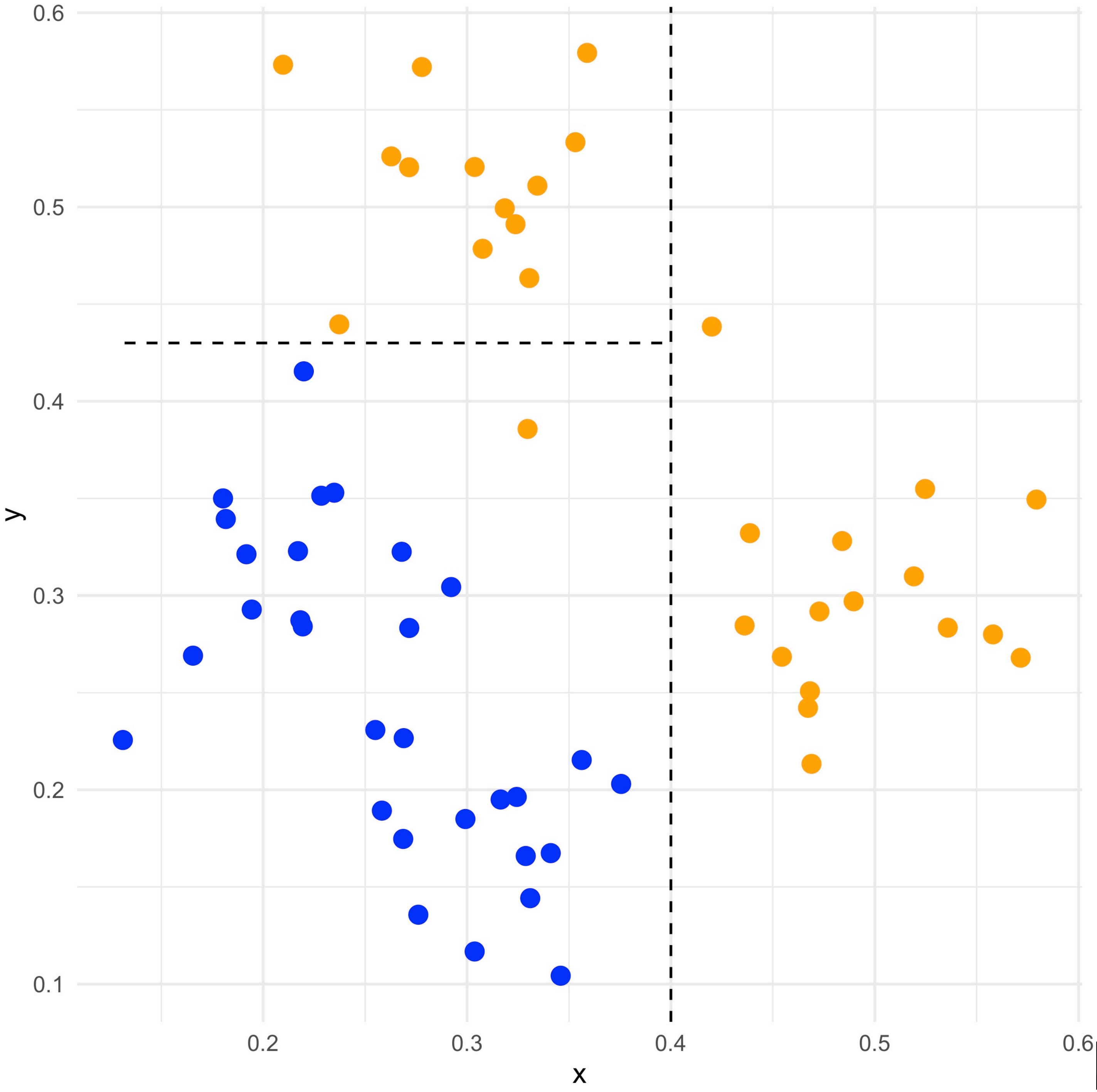
With this line, the probabilties will be given by the empircal probabilities.



Decision Tree

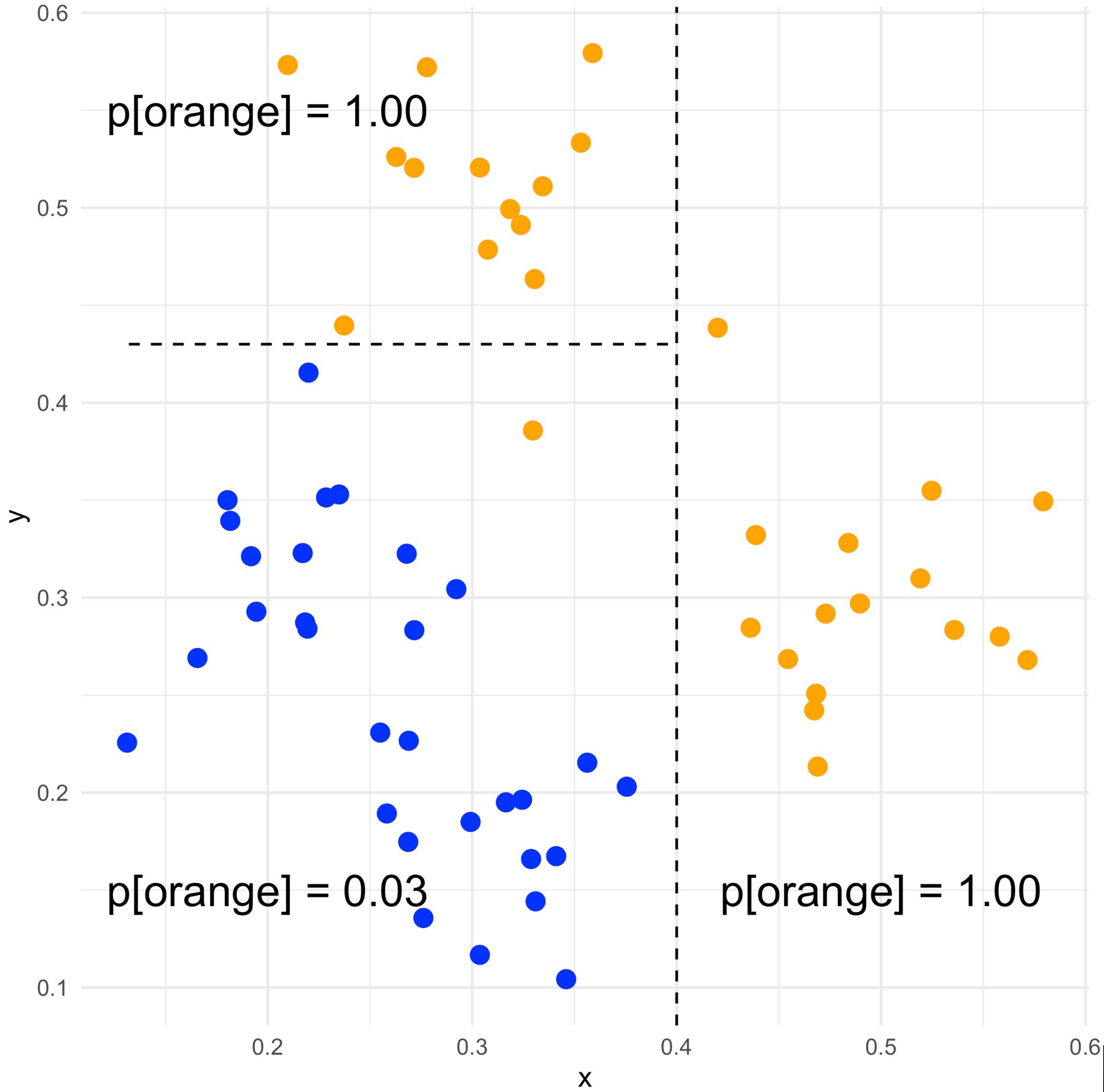
Now, we can split the data again with a line. This line, however, will only split on one side of the original line.

Like the first line, it will be perpendicular to the variable axes.



Decision Tree

Here are the probabilties for each of the three regions.

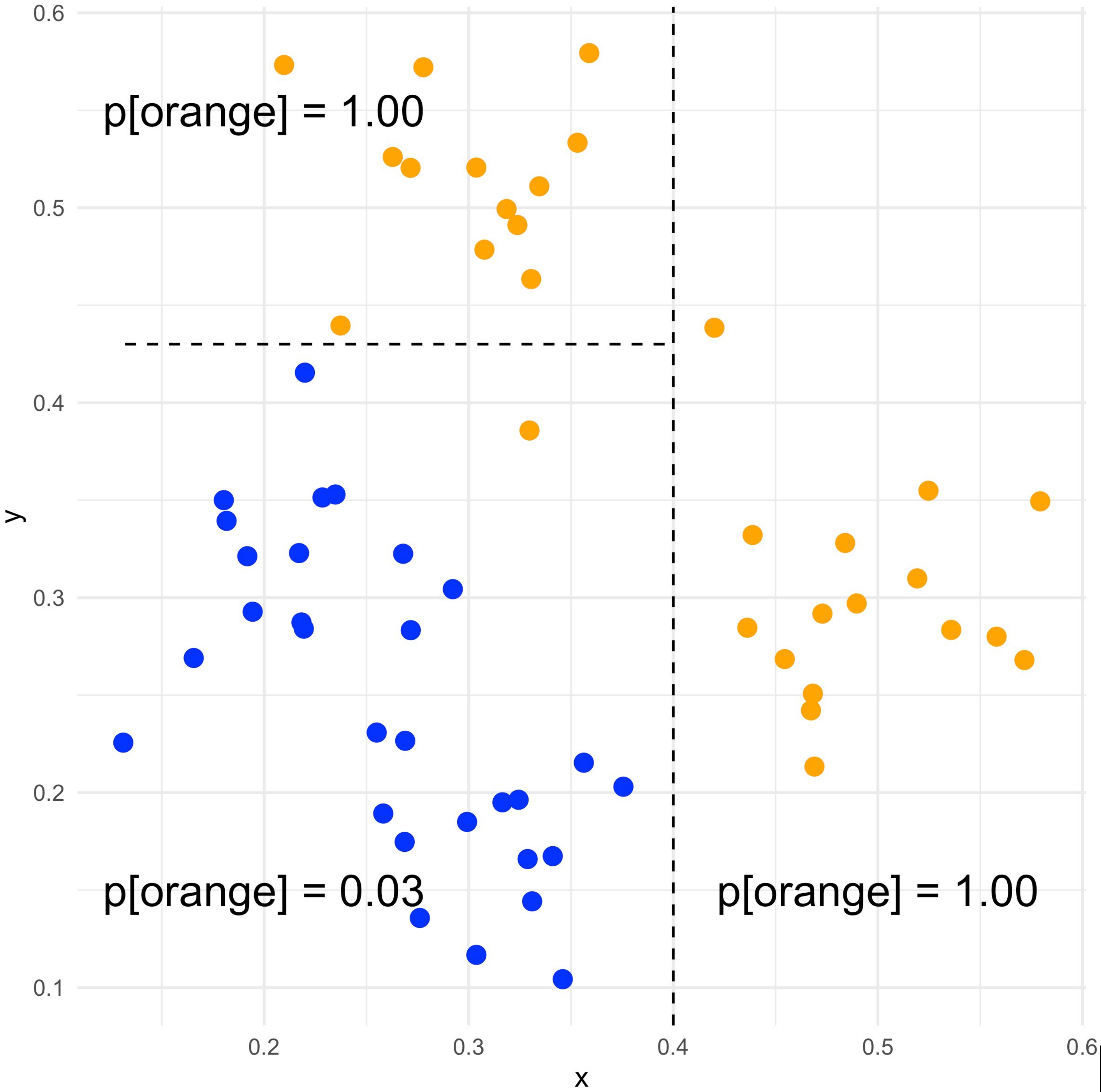


Decision Tree

How to determine the splits in the first place?

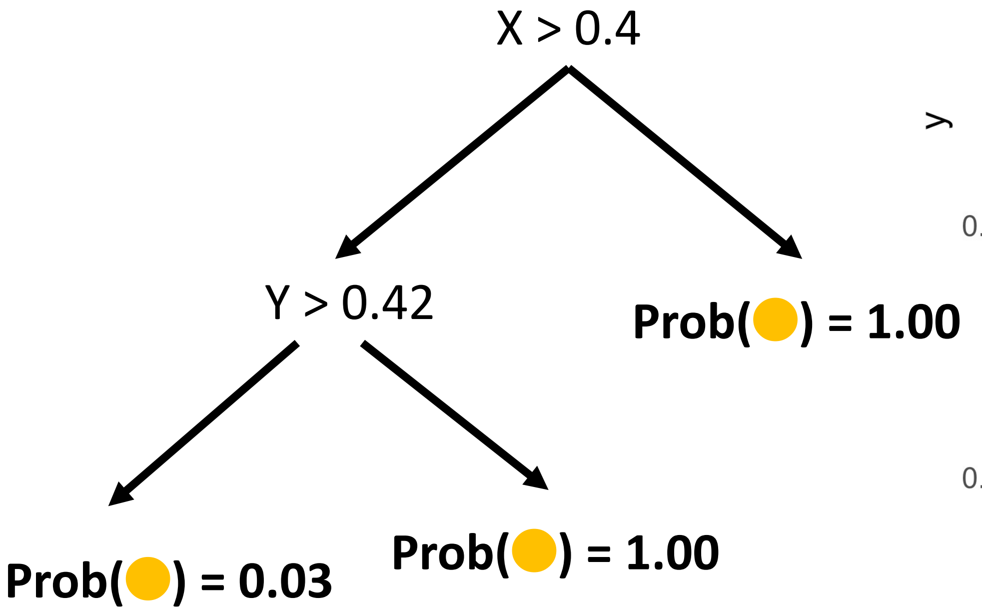
Pick the splits that maximise the probability of observing the training data, just as with logistic regression.

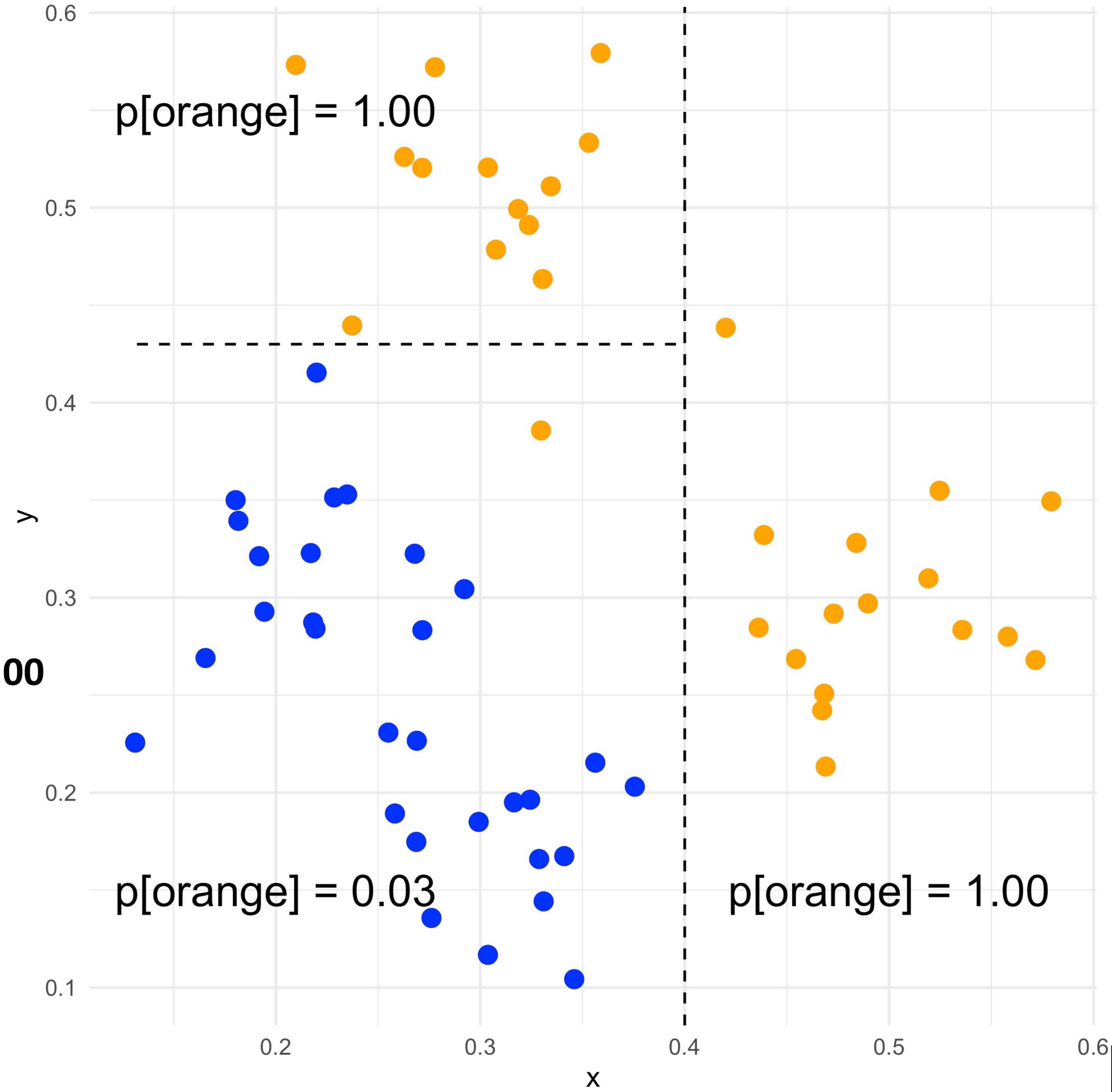
The tree is built greedily. We determine the best first split, and then the best second split, and so on.



Decision Tree

Notice that the algorithm can also be described by a tree-like structure, such as the one that follows.





Decision Trees: Assessement

Benefits

- Easy to understand
- Can extend to categorical features
- Can deal with missing values
- Naturally deals with interactions and non-linearity
- Fast to build and fast to apply to new data
- Invariant to the scale of input variables

Potential Challenges

- Unstable; High variability with a different data set
- Predictions have a amount of variability in the input space
- Can be difficult to built a model with a low error rate
- Trouble with boundaries that are not parallel to the axes

Gradient Boosted Trees

The trick to getting around most of the challenges is to harness the variability of the model by creating a collection of trees. There are two basic techniques: random forests and gradient boosted trees. We'll work with the latter.

Gradient Boosted Trees

Let's describe the algorithm for a two-class problem. Start by selecting a starting probability for each training data point, which we will call P0[i]. Usually this will be a constant.

Now, pick a random subset of the training data and build a decision tree on the residuals of P0[i]. This tree, when applied to the entire training data, will have predictions for the residuslas for each point which we will call T1[i].

Next, compute the following probabilities for each point for some small constant **$0 < \eta < 1$**:

$$P1[i] = P0[i] + (\eta \times T1[i])$$

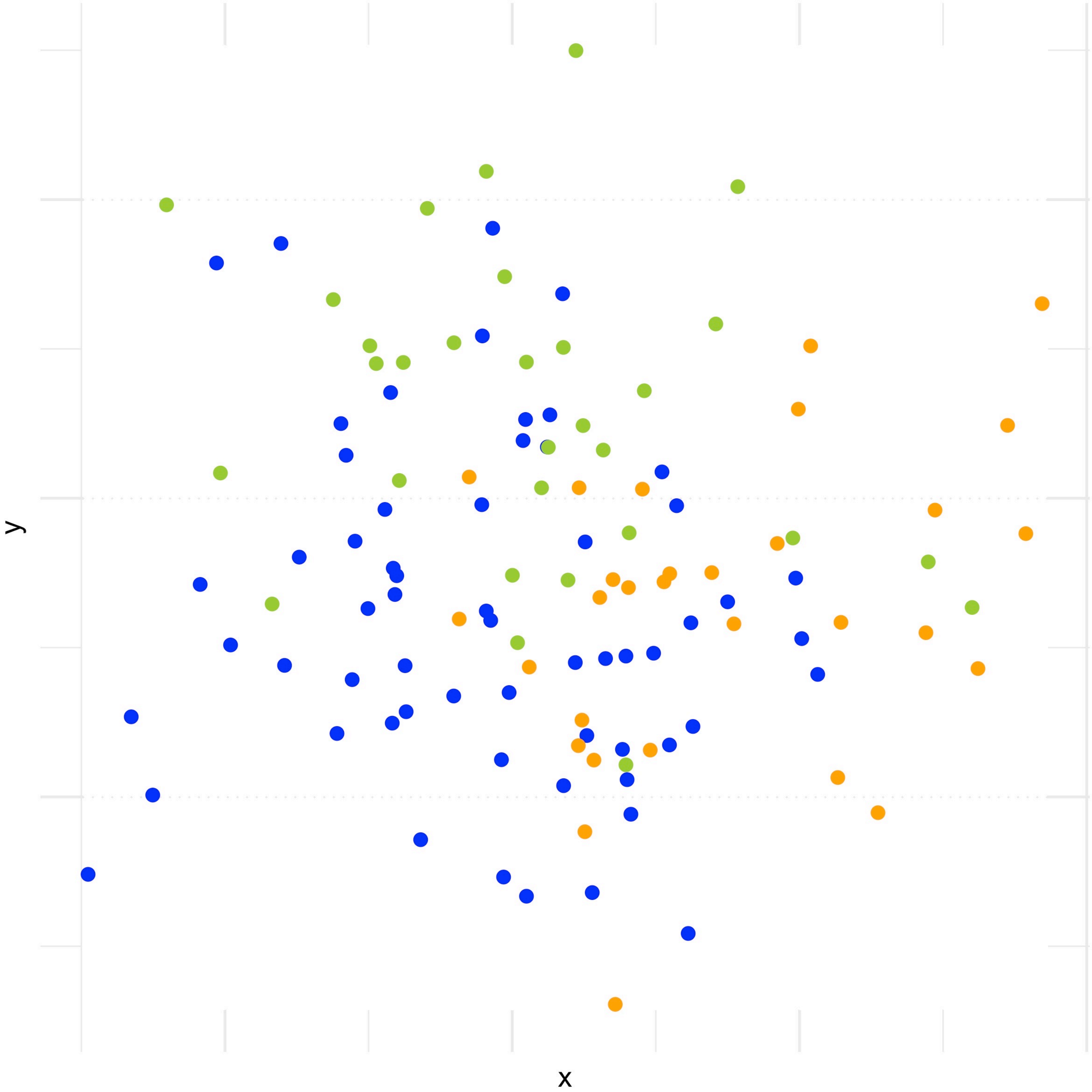
Then, fit another tree on another random subset of the training data that tries to compute the residuals from the probabilties P1[i]. The residuals T2[i] of these predictions are used to create new predictions:

$$P2[i] = P1[i] + (\eta \times T2[i])$$

We repeat this for to K trees, using the final PK[i] as the predictions.

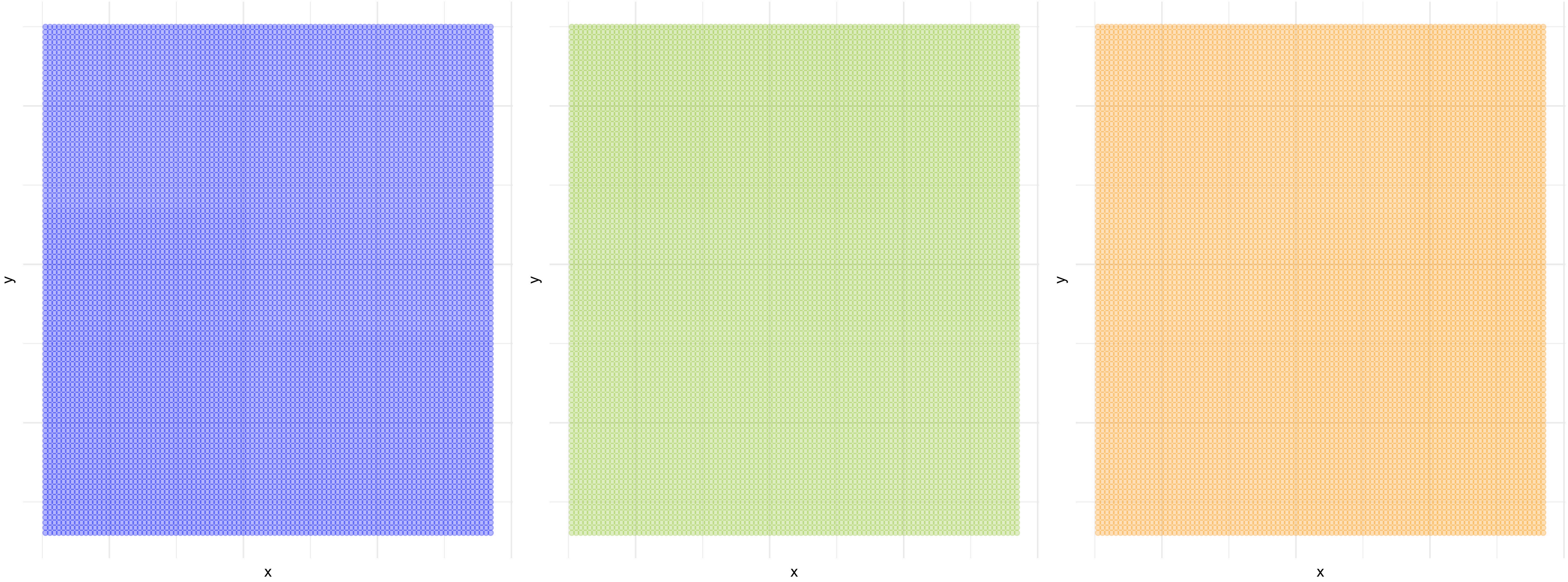
Gradient Boosted Trees

Let's see how this model works with some data.



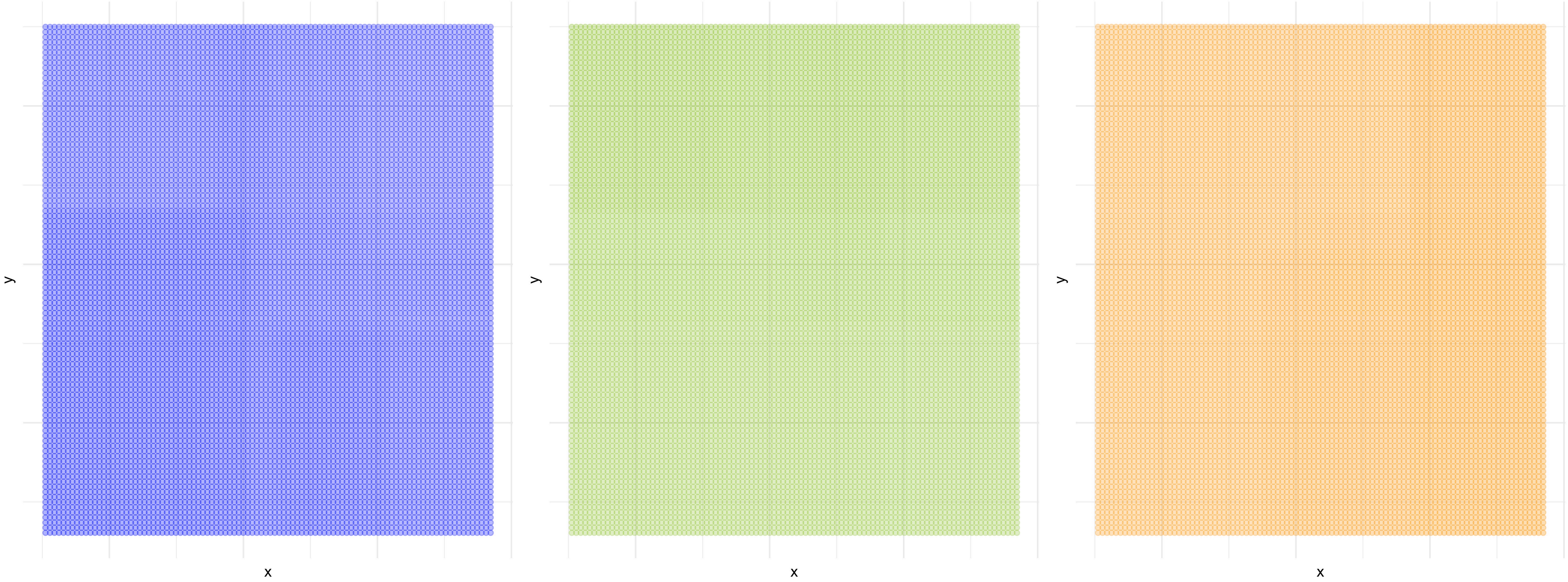
Gradient Boosted Trees

After just 1 tree (η = 0.05).



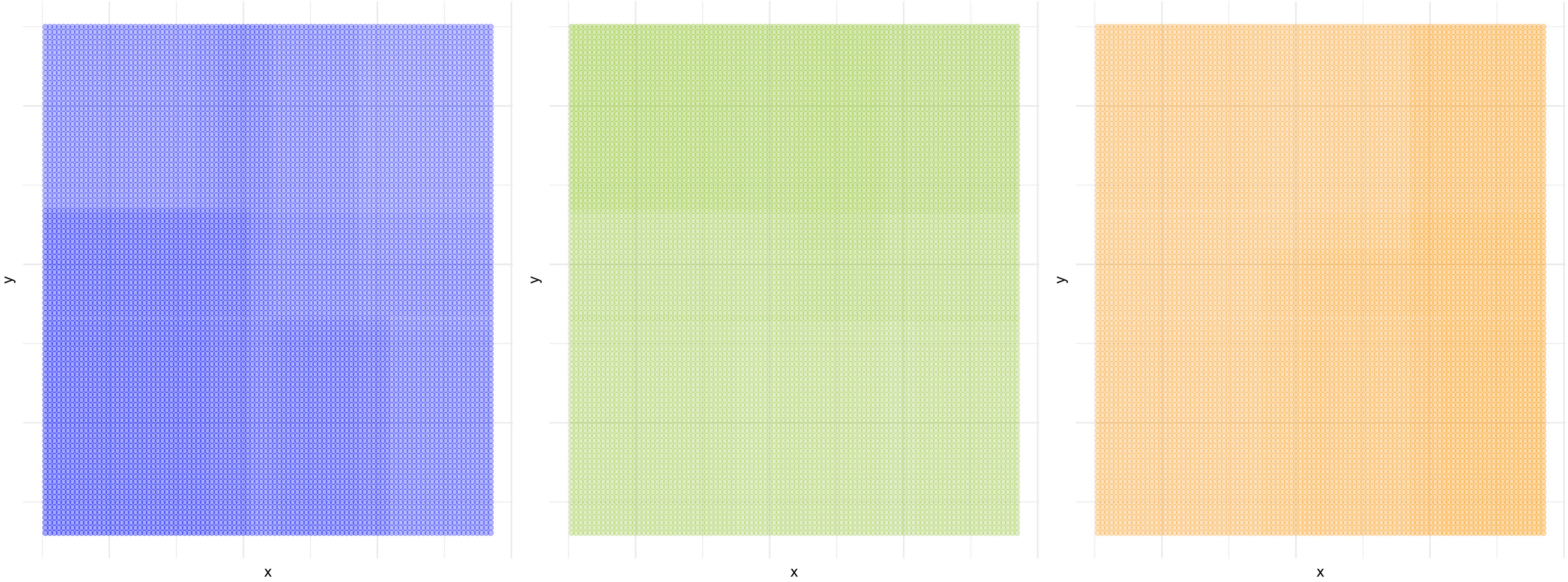
Gradient Boosted Trees

After 2 trees (η = 0.05).



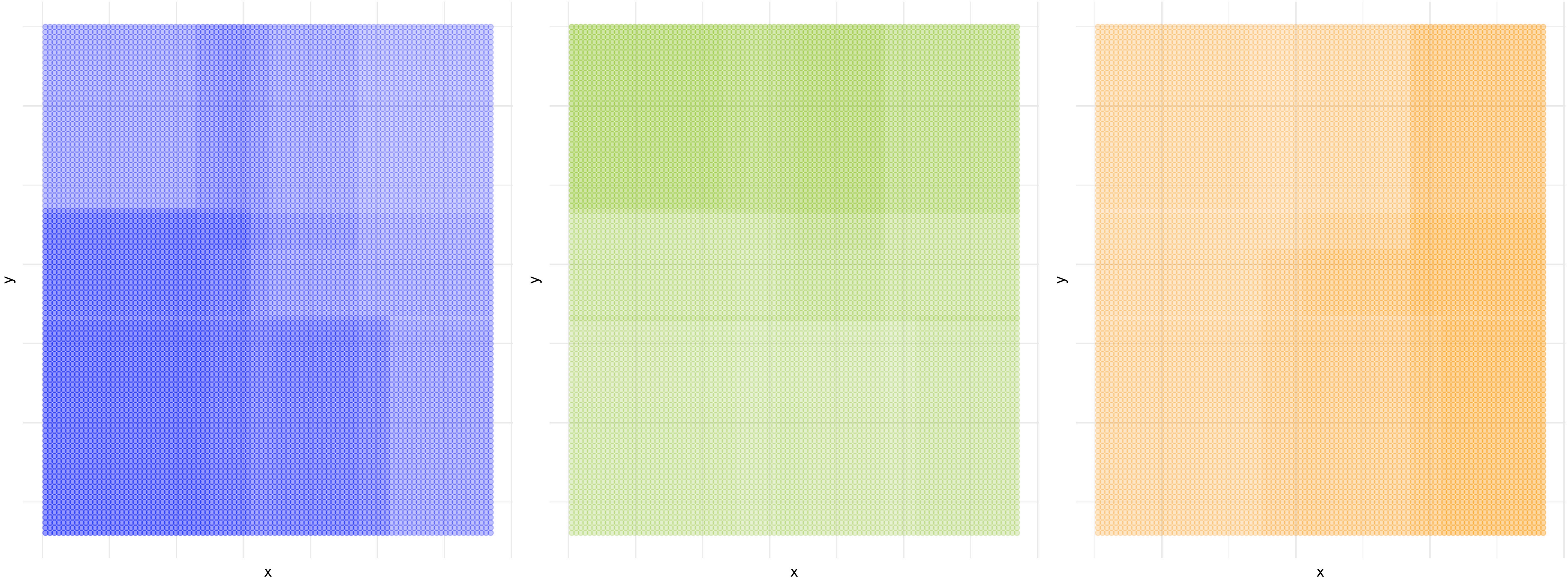
Gradient Boosted Trees

After 5 trees ($\eta = 0.05$).



Gradient Boosted Trees

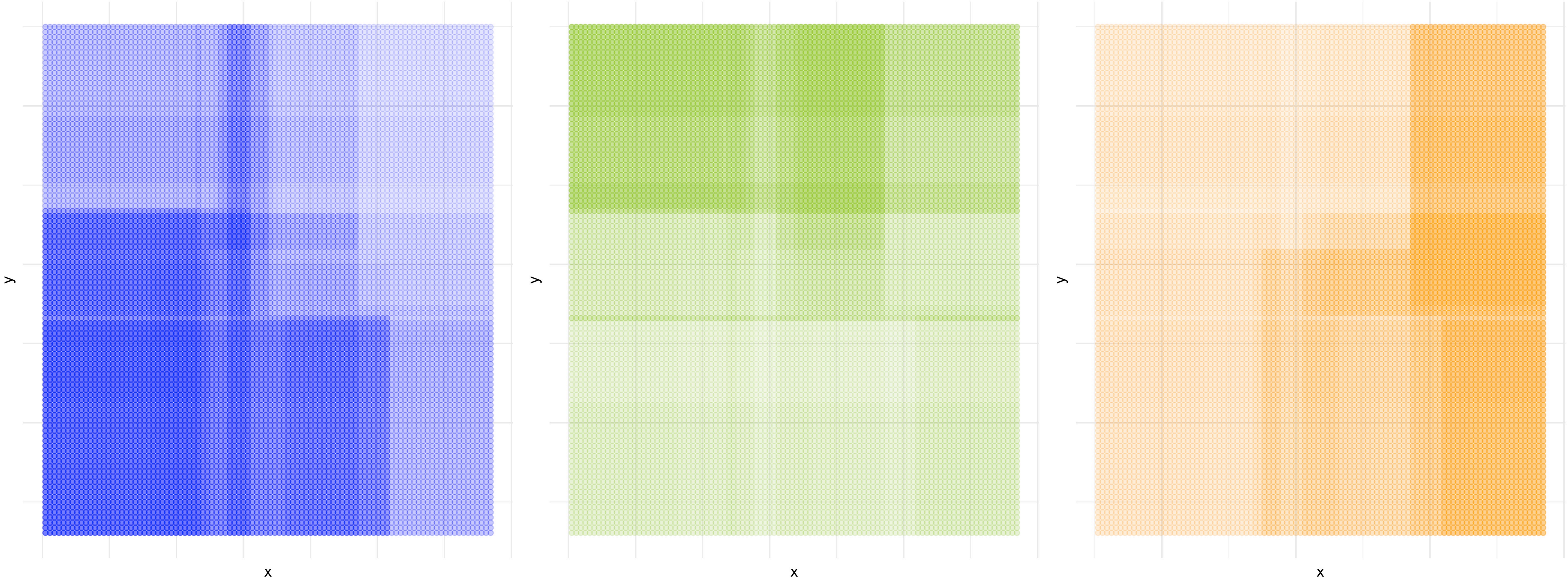
After 10 trees ($\eta = 0.05$).



T. ARNOLD

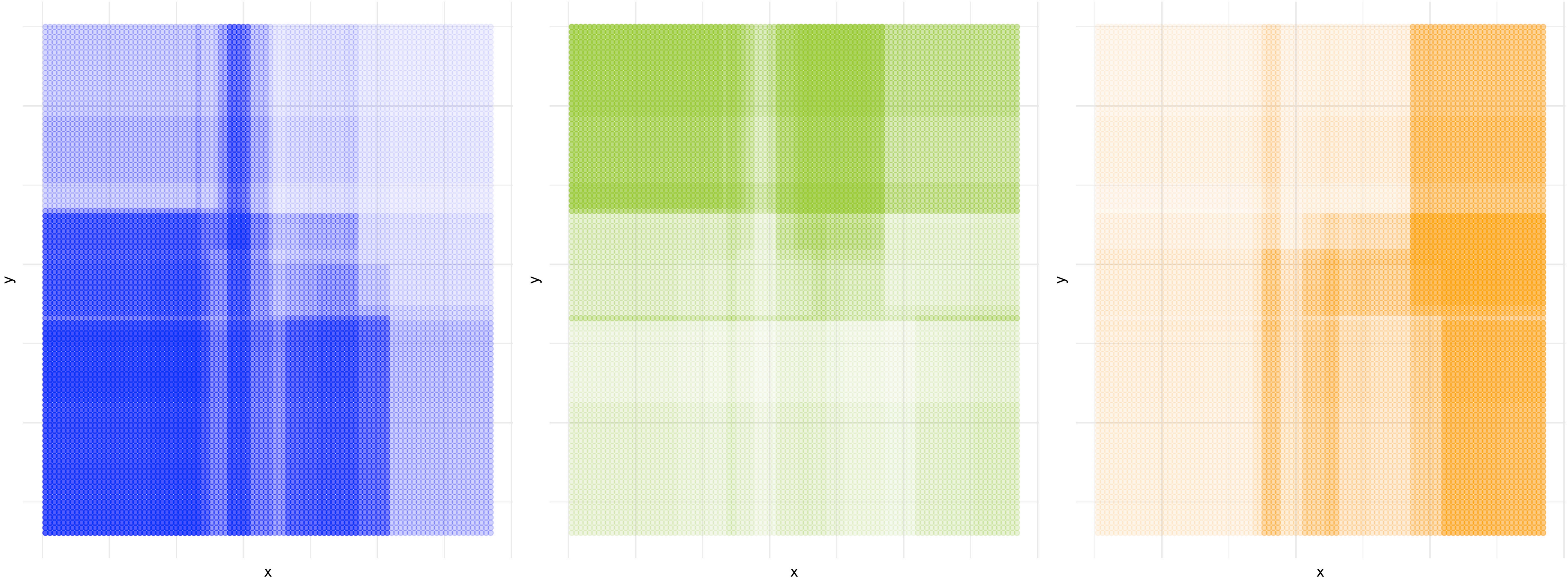
Gradient Boosted Trees

After 25 trees ($\eta = 0.05$).



Gradient Boosted Trees

After 50 trees (η = 0.05).



Gradient Boosted Trees

After 100 trees ($\eta = 0.05$).