TREES

Outline

Trees

A decision tree is a non-parametric learning method used for both classification and regression. It works by recursively partitioning the feature space into a set of rectangular regions, based on splitting rules that minimize an impurity measure (such as Gini index, entropy, or residual sum of squares). Each split aims to create more homogeneous subgroups in terms of the outcome variable. This process continues until a stopping criterion is met (e.g., minimum node size or maximum depth), or the tree is pruned to avoid overfitting.

For categorical outcomes, the tree assigns the majority class of training observations within each region to new data.

For numeric outcomes, the tree predicts the mean of the response values in the region.

Use Cases

- Classifying customers as likely to churn or not
- Predicting default risk in credit scoring
- Diagnosing diseases based on clinical indicators
- Fraud detection in banking transactions
- Predicting equipment failure from sensor data

Trees vs. Regression

- Regression models are well-tested, extensively studied, robust and work even with moderate violations of the assumptions
- However, with large number of variables regression becomes difficult to use when
  - *Relationships are non-linear*
  - *Variables interact*

Strengths

- Easy to interpret and explain to others
- Requires minimal preprocessing
  - *No need to standardize numeric features*
  - *Can handle many types of predictors (sparse, skewed, continuous, categorical, etc.)*
  - *Handles missing values without imputation*
  - *Robust to outliers*
- Efficient
  - *Only uses features that contribute to prediction performance*
- Flexible
  - *Handles non-linear relationships and interactions*
- Feature Selection
  - *implicitly conduct feature selection. If a predictor is never used in a split, then the prediction equation is independent of it. However, this advantage is weakened for highly correlated predictors where the choice of which is used for a split is somewhat random.*

Weaknesses

- Less-than optimal predictive performance.
  - *Tree models partition the predictor space into rectangular regions. If the relationship between predictors and outcome is not adequately described by these rectangles, performance will be poor.*
  - *Number of possible predicted outcomes is finite and is determined by the number of leaves. This limitation is unlikely to capture all the nuances of the data.*
- Model instability
  - *Slight changes in the data can drastically change the structure of the tree, hence the interpretation. Thus, these models have high variance. Easy to overfit train data.*

Types

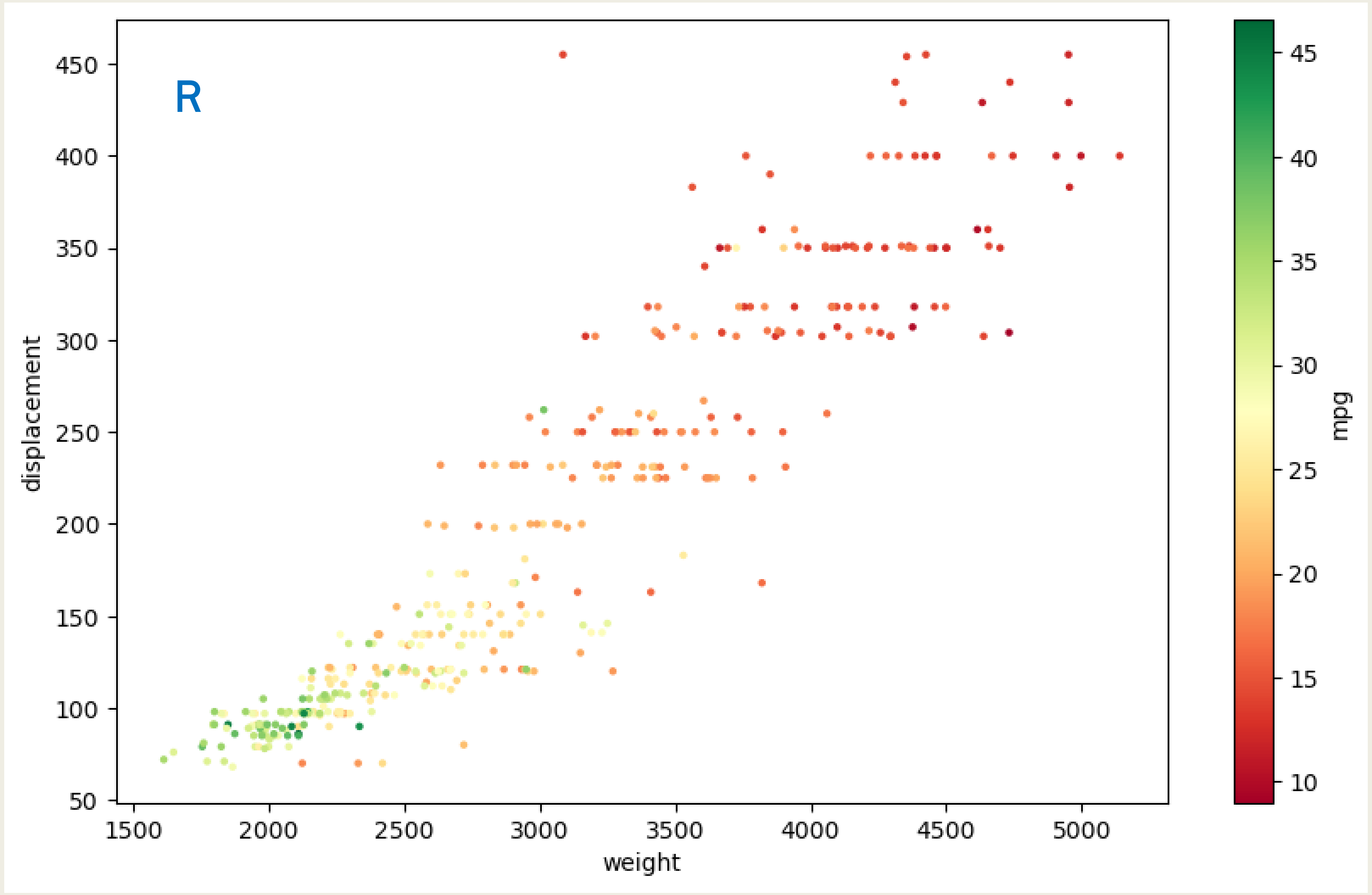
- Based on the nature of the outcome variable, trees can be categorized as
  - *Regression Trees: Outcome is numeric (e.g., price in $), or*
  - *Classification Trees: Outcome is categorical (e.g., whether a product is purchased or not)*

REGRESSION TREES

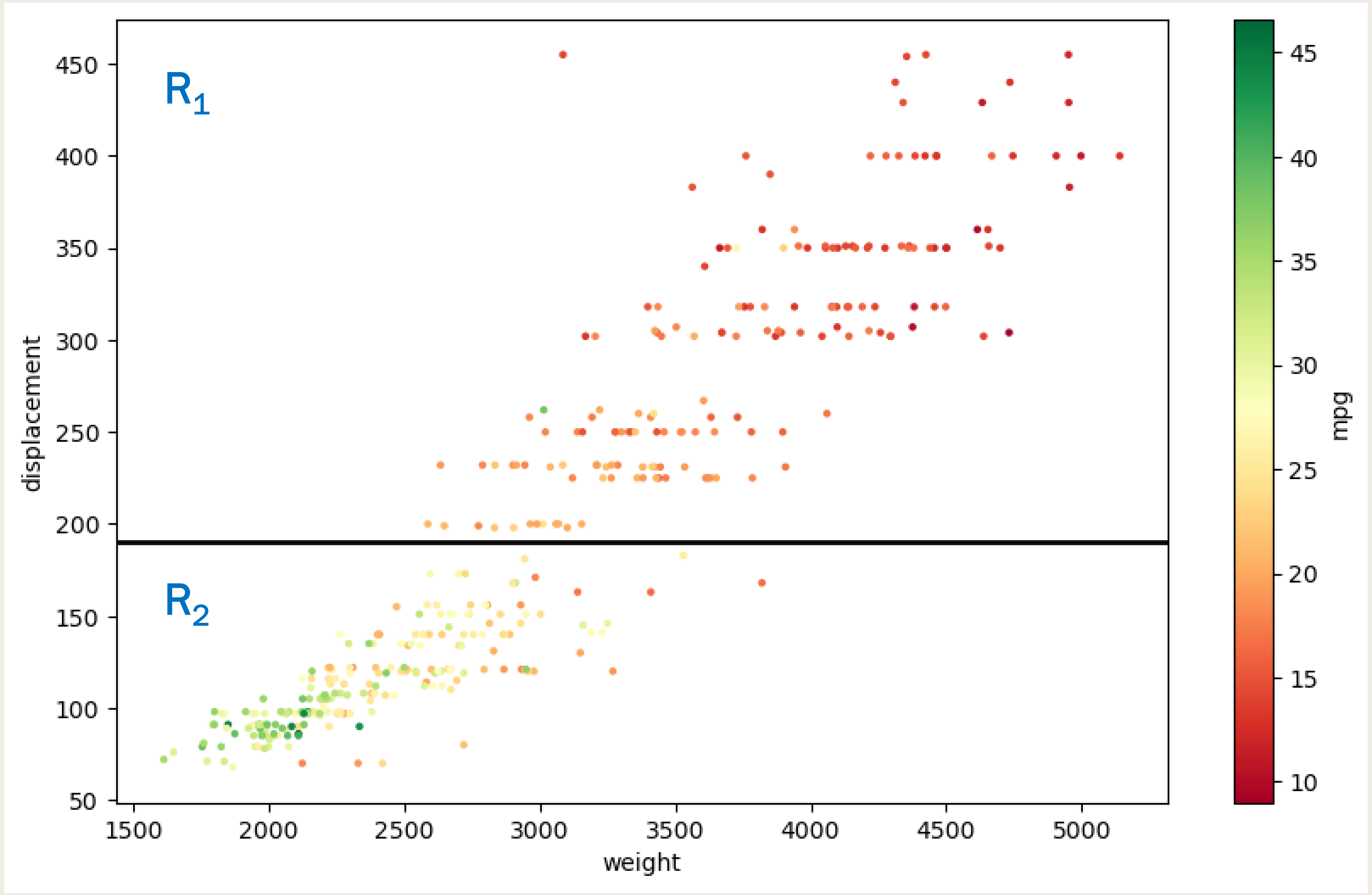
Illustration of Process

- In the next few slides, we will illustrate the process by which tree models stratify or segment predictor space to make predictions.
- Lets say we are interested in predicting gas mileage for cars (mpg) based on their weight (wt) and displacement in cu. in (disp).

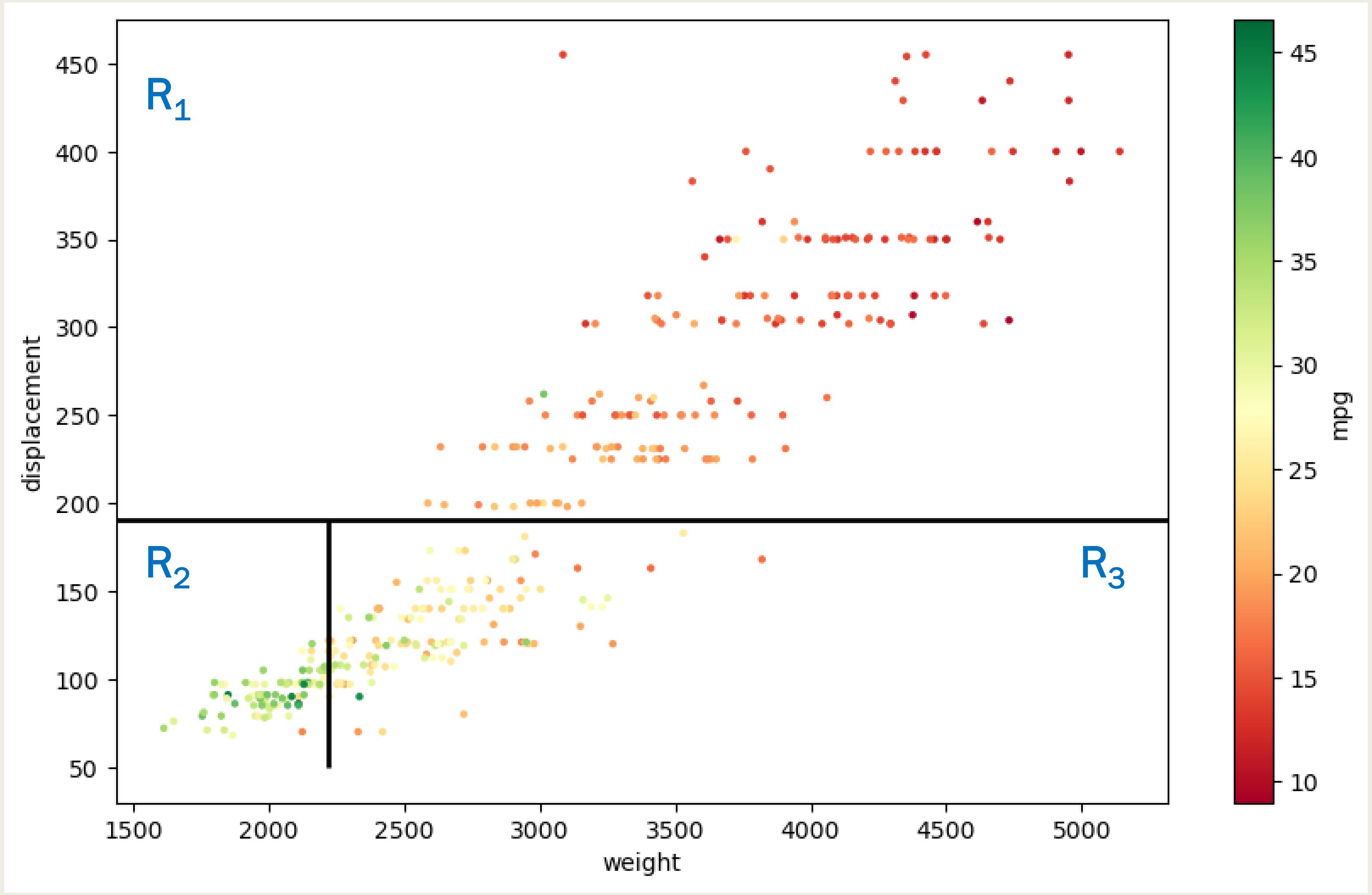
What is the “best” way to partition Predictor Space, R?



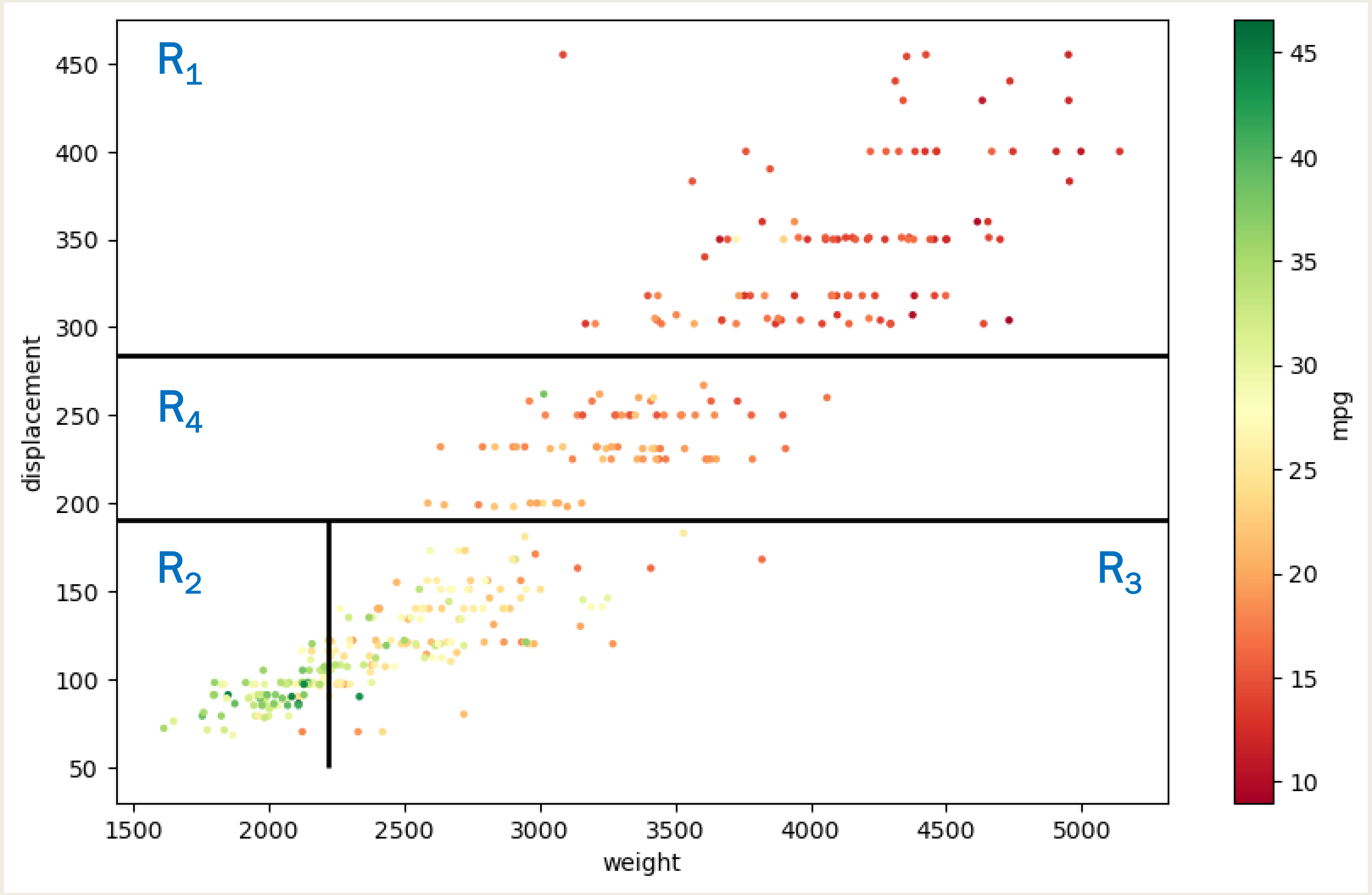
Which of the predictor spaces R_1 or R_2 should be segmented next and where?



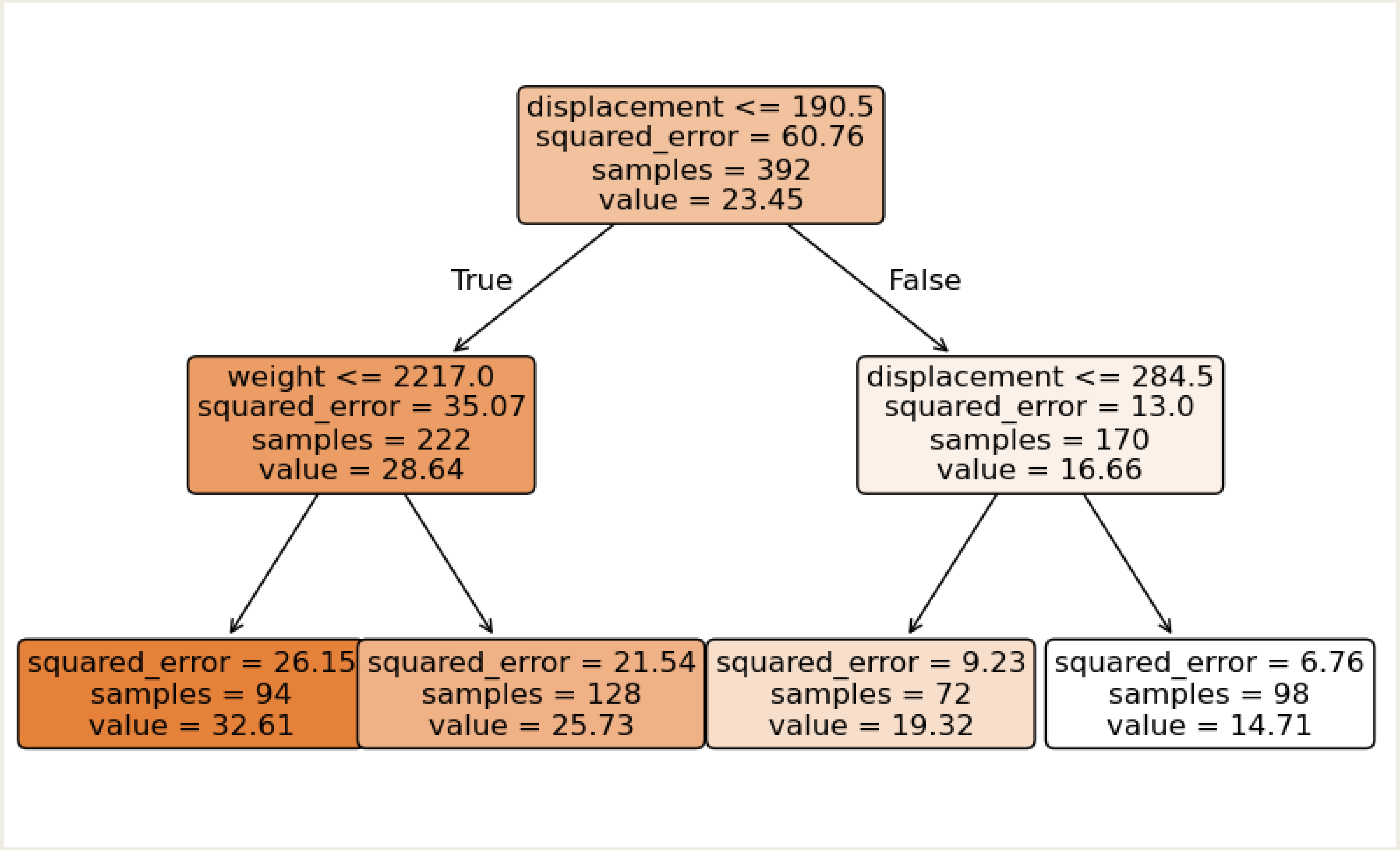
Which of the predictor spaces R_1,R_2, or R_3 should be segmented next and where?



What is the predicted “mpg” for each region? How accurate are these predictions?



Model Representation



Root Node

Interior Node

Leaf or
Terminal Node

Regression Trees

Regression trees recursively partition the data into smaller groups that are more homogeneous with respect to the outcome.

Tree Building Process

- The predictor space (i.e., set of possible values for $X_1, X_2, \ldots, X_p$) is divided into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
- Every observation that falls into the region R_j , gets the same prediction, which is simply the mean of the response values for the training observations in R_j .

Tree Building Process

- In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model.
- The goal is to find boxes $R_1, \ldots, R_J$ that minimize the SSE, given by

$$\text{SSE} = \sum_{j=1}^{J} \sum_{i \in Rj} (yi - \overline{y_{Rj}})^2$$

  - *where $\overline{y_{Rj}}$ is the mean outcome for training observations in the j*th *box*

Tree Building Process

- The process begins with the entire data, R, and searches every distinct value of every predictor to find the predictor and split value that partitions the data into two groups (R_1 and R_2) such that the overall sum of squared errors is minimized

$$\text{SSE} = \sum_{i \in R1} (yi - \overline{y_{R1}})^2 + \sum_{i \in R2} (yi - \overline{y_{R2}})^2$$

  - *Where $\overline{y_{R1}}$ and $\overline{y_{R2}}$ are mean training set outcomes within regions R_1 and R_2, respectively*

- Next, the process is repeated, looking for the best predictor and best split value (or cut point) in order to split the data further so as to minimize the SSE within each of the resulting regions.
- However, this time, instead of splitting the entire predictor space, one of the previously identified regions is split. This creates three regions, R_1, R_2 and R_3. Because of the recursive splitting nature of trees, this method is also known as recursive partitioning.
- This recursive partitioning process continues until a stopping criterion is reached such as number of observations in a region falls below 5.

Computational Shortcuts

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- This is the reason for the top-down, greedy approach of recursive binary splitting.
- The approach is top-down because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Pruning a Tree

- The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance.
- A smaller tree with fewer splits (i.e., fewer regions $R_1, \ldots, R_J$) might lead to lower variance and better interpretation at the cost of a little bias.
- There are two approaches to pruning the tree
  - *Pre-pruning*
  - *Post-pruning*

Pre-Pruning

- Parameters to control tree complexity are set before the tree is built. E.g., max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes
- Stops the tree from growing as soon as conditions are met
- This method is fast and computationally efficient but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split — that is, a split that leads to a large reduction in SSE later on.

Post-Pruning

- An alternative strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree.
- The goal of this process is to find a "right sized tree" that has the smallest error rate.
- Cost complexity pruning — also known as weakest link pruning — is used to do this
- We consider a sequence of trees indexed by a nonnegative tuning parameter, c_p (also known as a complexity parameter). For each value of c_p there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i \in Rm} (yi - \overline{y_{Rm}})^2 + c_p|T|$$

  - *is as small as possible. Here $|T|$ indicates the number of terminal nodes of the tree T, R_m is the rectangle (i.e. the subset of predictor space) corresponding to the m^{th} terminal node, and $\overline{y_{Rm}}$ is the mean of the training observations in R_m. Finally, note that Scikit-Learn refers to c_p as ccp_alpha.*

Choosing the Best Tree

- The tuning parameter c_p controls a trade-off between the subtree's complexity and its fit to the training data.
- Optimal value of c_p is chosen using using cross-validation.
- We then return to the full data set and obtain the subtree corresponding to c_p.

Predictor Importance

- Relative importance of predictors in trees can be determined by examining the reduction in SSE attributed to each split.
- Intuitively, predictors that appear higher in the tree, or that appear multiple times in the tree will be more important than predictors that appear lower in the tree or not at all

Summary: Tree Algorithm

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of c_p.
3. Use k-fold cross-validation to choose α. For each k = 1, . . . , K:
   a) *Repeat Steps 1 and 2 on the $\frac{K-1}{K}$ th fraction of the training data, excluding the kth fold.*
   b) *Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of c_p.*
   c) *Average the results, and pick c_p to minimize the average error.*
4. Return the subtree from Step 2 that corresponds to the chosen value of c_p.

CLASSIFICATION TREES

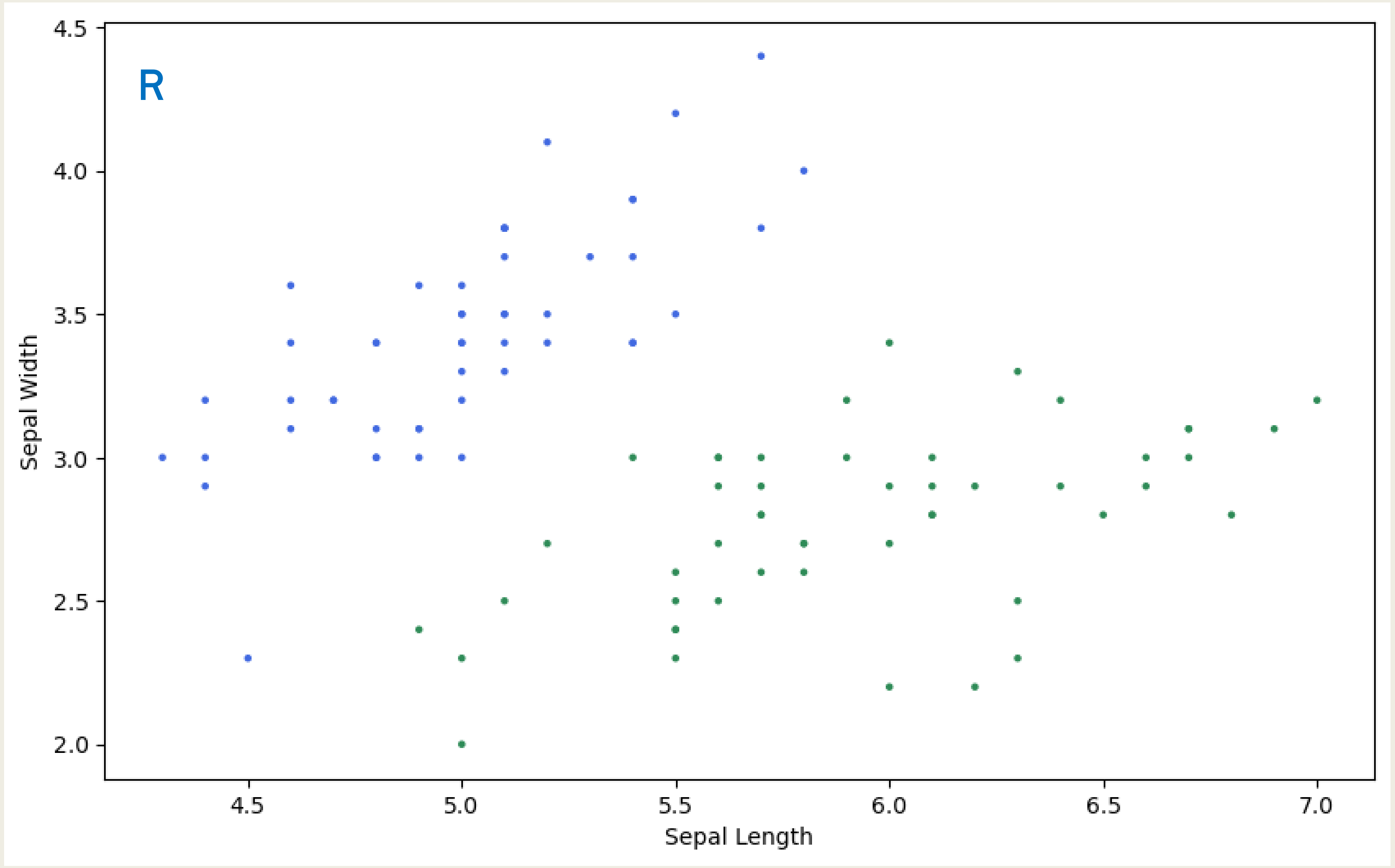
Classification Trees

- Very similar to a regression tree, except that it is used to predict a qualitative response rather than a quantitative one.
- For a classification tree, predicted value is based on the most commonly occurring class of training observations in the region to which it belongs.

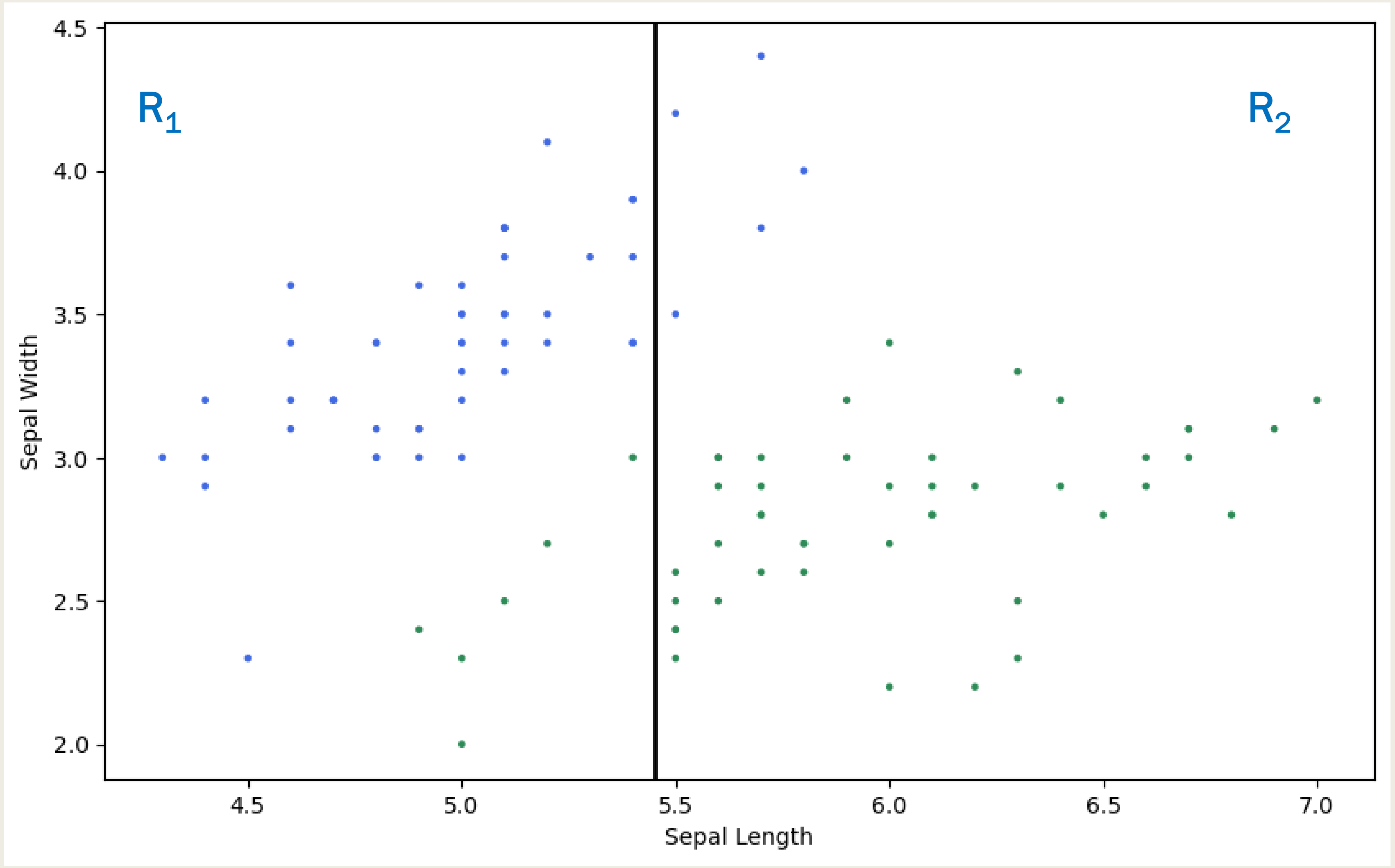
Simple Illustration

- In the next few slides, we illustrate the process by which a classification tree stratifies or segments predictor space to make class predictions.
- Lets say we are interested in predicting flower type (setosa or versicolor) based on sepal length and sepal width.

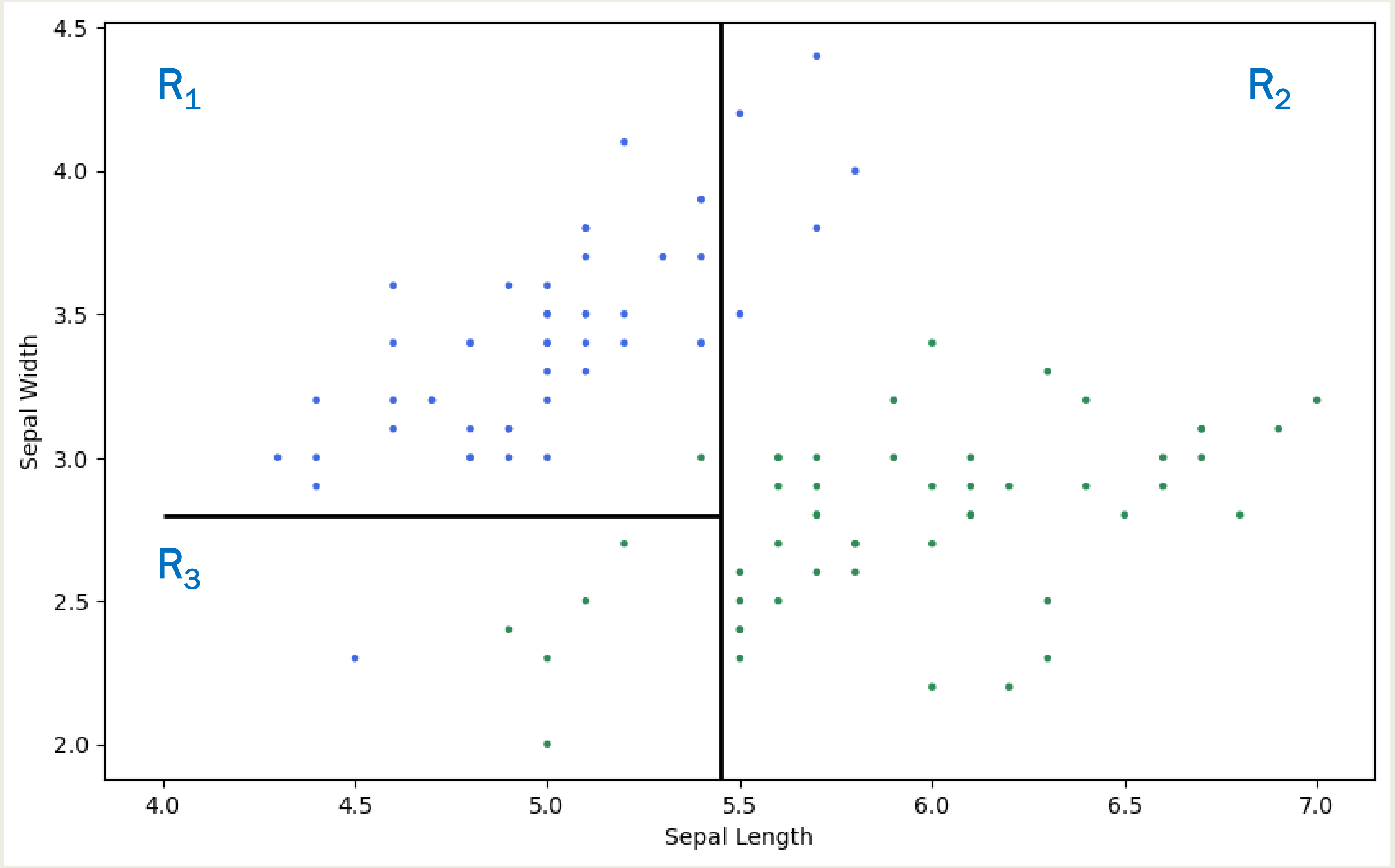
What is the “best” way to partition Predictor Space, R?



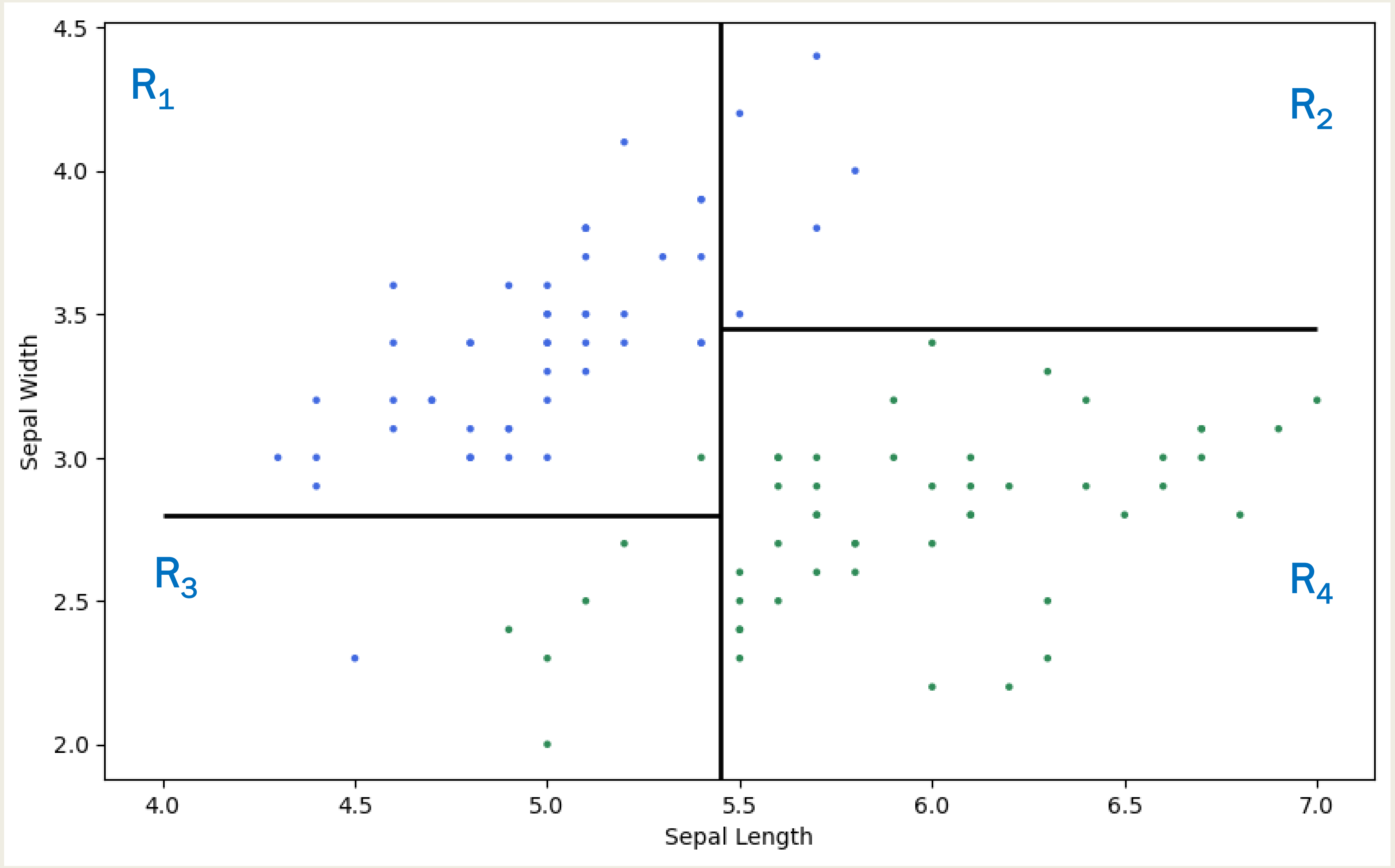
Which of the predictor spaces R_1 or R_2 should be segmented next and where?



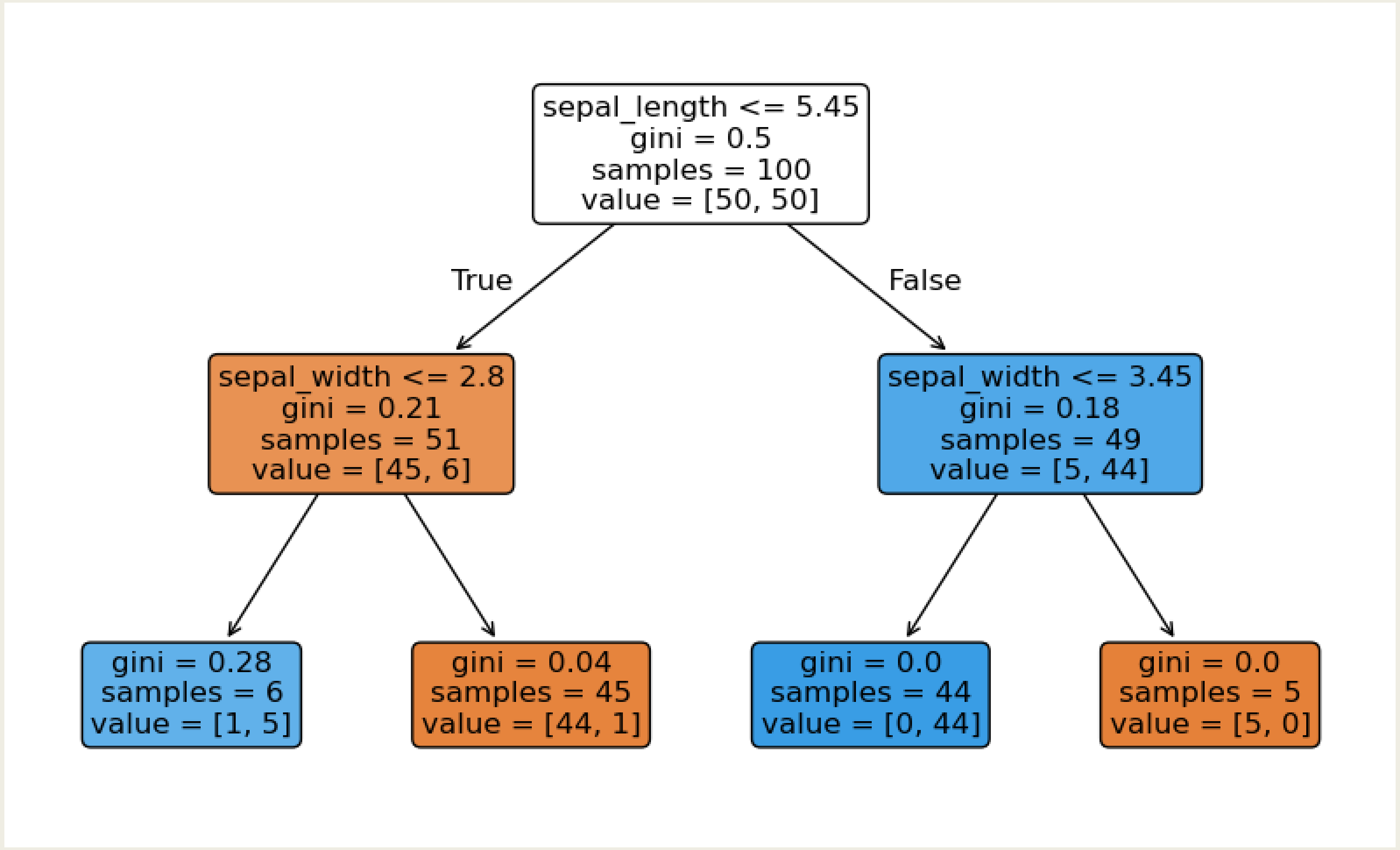
Which of the predictor spaces R_1,R_2, or R_3 should be segmented next and where?



What is the predicted class for each region? How accurate are these predictions?



Model Representation



Root Node

Interior Node

Leaf or
Terminal Node

Classification Tree Algorithm

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, SSE cannot be used as a criterion for making the binary splits
- A natural alternative to RSS is the misclassification rate, which is simply the fraction of the training observations in that region that do not belong to the most common class:

$$E = 1 - \max(p_{mk})$$

Here p_{mk} represents the proportion of training observations in the mth region that are from the kth class.

- However misclassification rate is not sufficiently sensitive for tree-growing, and in practice two other measures are preferable.

Classification Tree Algorithm

- The Gini index is defined by

  $G = \sum_{k=1}^{K} p_{mk}(1 - pmk)$,

  - *a measure of total variance across the K classes. The Gini index takes on a small value if all of the p_{mk}'s are close to zero or one.*

- For this reason the Gini index is referred to as a measure of node purity — a small value indicates that a node contains predominantly observations from a single class.

- An alternative to the Gini index is cross-entropy, given by

  $G = \sum_{k=1}^{K} p_{mk}\log(pmk)$,

- It turns out that the Gini index and the cross-entropy are very similar numerically.

CROSS-VALIDATION

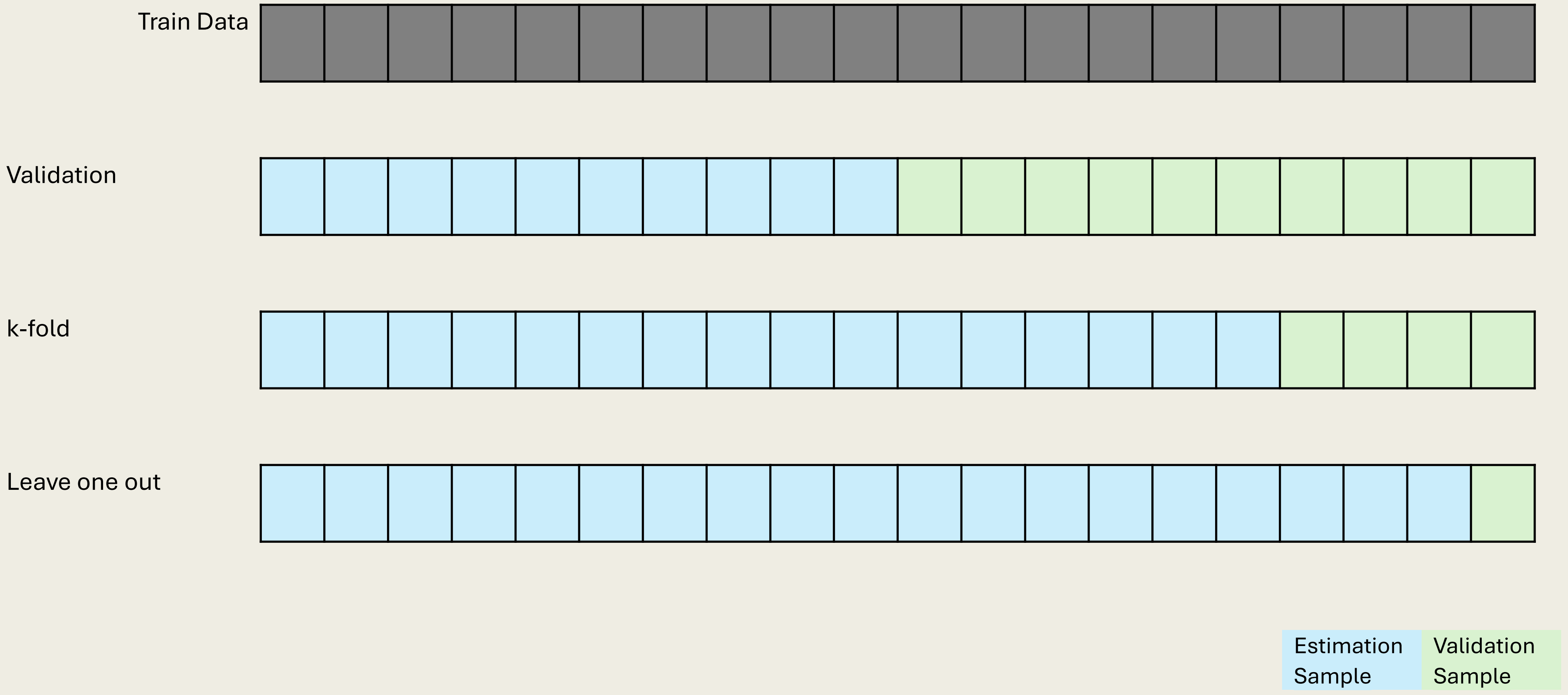
Cross-Validation

- Models are fitted on a train sample and evaluated on a test sample. This would be okay if one were only fitting one model. In practice, the analyst may fit a number of different models to find the optimal hyperparameters. But, the test sample was designed to be used only for the final model.
- This issue is addressed through cross-validation, wherein each model is fit on a subsample of the train sample and evaluated across the observations that were held out from the original subsample.

Cross-validation

- Cross-validation is part of a general technique called resampling. Bootstrapping, another resampling approach will be examined later.
- Resampling methods involve repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain more information about the fitted model
  - *Model Assessment: estimate test error rates*
  - *Model Selection: select the appropriate level of model flexibility*
- Resampling method in general are computationally expensive! But these days we have powerful computers.
- Common Cross-validation methods include
  - Validation Set Approach
  - K-fold Cross-Validation
  - Leave One Out Cross-Validation (LOOCV)

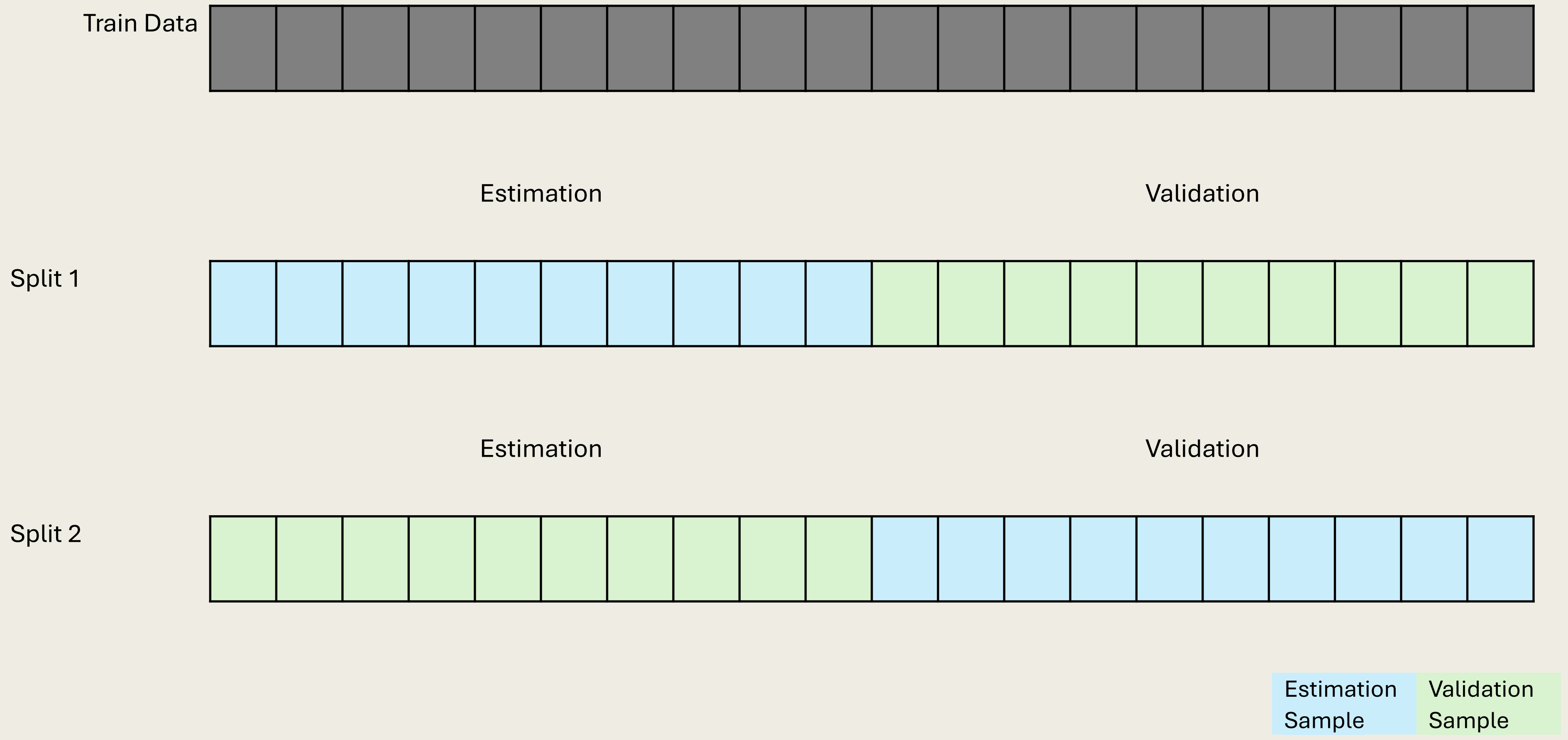
Cross-Validation



Validation Set Approach

1. Randomly split the data into a estimation set and a validation set.
2. Estimate the model using the estimation sample.
3. Apply the estimated model to the validation sample
4. Error in the validation-set provides an estimate of error in a new sample.

Cross-Validation: Validation Set



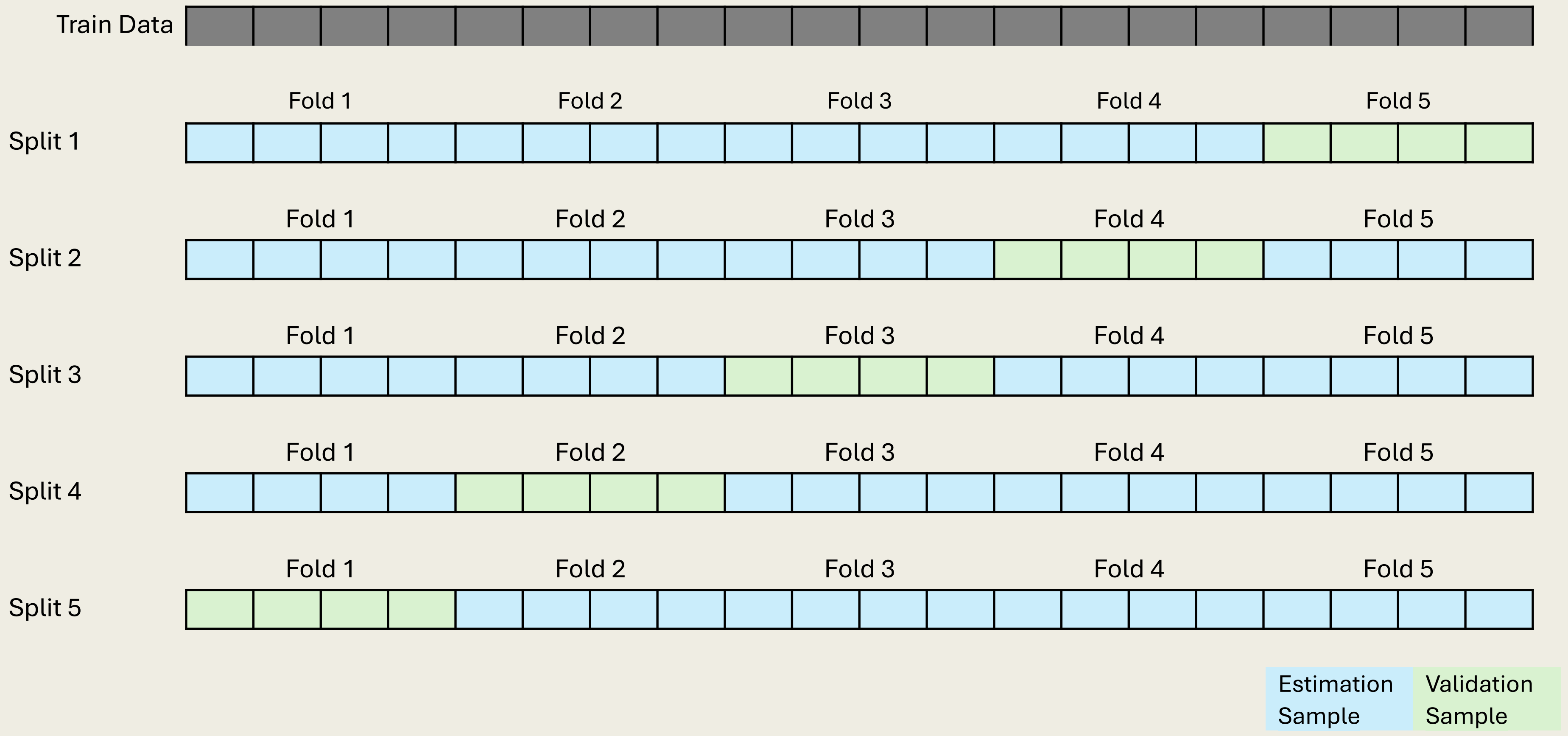
Cross-Validation: Validation Set

- Advantages
  - *Simple approach.*
  - *Not computationally intensive*
- Drawbacks
  - *The validation estimate of the test error can be highly variable, depending on precisely which observations are included in the training set and which observations are included in the validation set.*
  - *In the validation approach, only a subset of the observations — those that are included in the training set rather than in the validation set — are used to fit the model. This suggests that the validation set error may tend to overestimate the test error for the model fit on the entire data set.*

K-Fold Cross-Validation

- Widely used approach for estimating test error.
- Idea is to randomly divide the data into k equal-sized parts. We leave out part k, fit the model to the other k − 1 parts (combined), and then obtain predictions for the left-out kth part.
- This is done in turn for each part k = 1, 2, . . . K, and then the results are combined.

Cross-Validation: k-fold



Computation

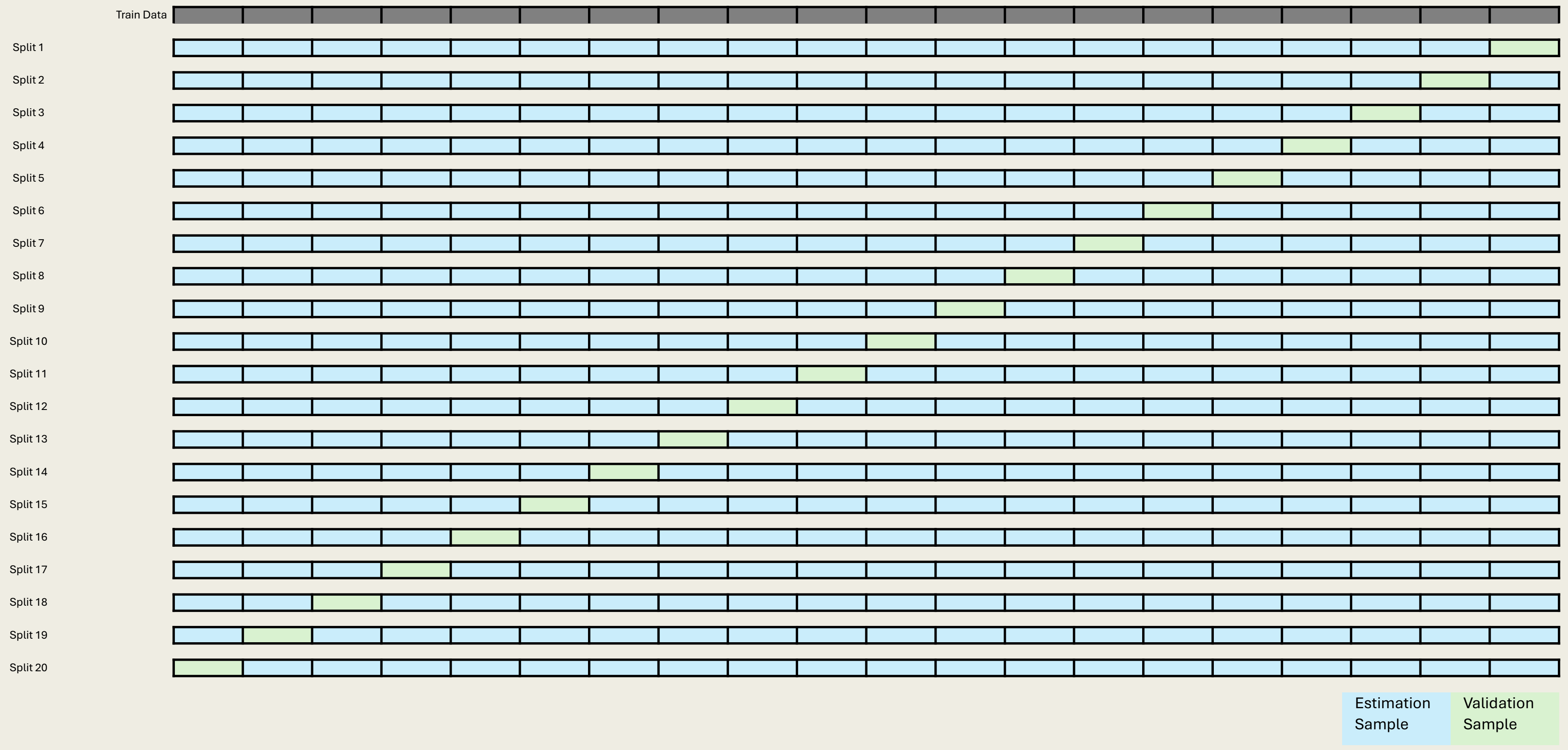
- Let the K parts be C_1, C_2, . . . C_K, where C_k denotes the indices of the observations in part k. There are n_k observations in part k: if N is a multiple of K, then n_k = n/K.
- Compute
  - $CV_k = \sum_{k=1}^{k} \frac{n_k}{n} MSE_k$

- When k = n, we get n-fold or leave-one-out cross validation

Evaluation of k-fold cross Validation

- Advantages
  - *Less bias than Validation because model is trained on a larger sample (for 5-fold, 80% compared to 50% for Validation)*
  - *Less variance because test statistic is averaged k times.*
- Drawbacks
  - *Computationally more intensive than Validation but less than LOOCV*
  - *Works best for k=5 or k=10*

Cross-Validation: Leave-one-out



LOOCV

1. Randomly divide the train sample into two halves:
    1. *Estimation Sample: n-1*
    2. *Validation Sample: 1*
2. Fit the model on estimation sample
3. Test model on validation sample
4. Repeat Steps 1-3, n times
5. MSE is average of n models.

$$CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i .$$

Evaluation of LOOCV

- Advantages
  - *Less bias because almost the entire sample is used for training the model*
  - *Generally less variance than Validation because of n repeats.*
- Disadvantages
  - *LOOCV is computationally intensive*
    - mostly because algorithms use the same approach as k-fold cross validation instead of using a computational shortcut. See Elements of Statistical Learning, Chapter 7 for details
  - *LOOCV sometimes useful, but typically doesn't shake up the data enough. The estimates from each fold are highly correlated and hence their average can have high variance.*
  - *A better choice is k = 5 or 10.*

TUNING HYPERPARAMETERS

Model Hyperparameters

- Parameters are estimated during model training. On the other hand, hyperparameters control the learning process.
- Decision Trees have a number of hyperparameters such as complexity parameter (ccp_alpha), depth of tree (max_depth) and minimum samples to split (min_samples_split).
- Default values for model hyperparameters provide a reasonable starting point for modeling. However, tuning these hyperparameters can greatly improve model performance.

Process of Tuning Model Hyperparameters

1. Identifying hyperparameter(s) to tune
2. Specifying range of hyperparameter values to evaluate.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
   - *One or more hyperparameters may be selected. E.g., max_depth, min_samples_leaf, max_features*
2. Specify range of hyperparameter values to evaluate.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specify range of hyperparameter values to evaluate.

   {'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

   'min_samples_leaf':[5, 10, 20, 40, 100, 150, 200],

   'max_features': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]}
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specify range of hyperparameter values to evaluate.
3. Train a model on each combination of hyperparameters
   - *Grid Search performs an exhaustive search while Randomized Search methods do a randomized search*
4. Measure performance
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specify range of hyperparameter values to evaluate.
3. Train a model on each combination of hyperparameters
4. Measure performance
   - *Training sample error is a poor guide for test error. Test sample can only be used for evaluating the final model, not a series of models with different combinations of hyperparameters.*
   - *So, models are evaluated based on cross-validation error.*
5. Choose hyperparameter values for model with best performance

Process of Tuning Model Hyperparameters

1. Identify hyperparameter(s) to tune
2. Specify range of hyperparameter values to evaluate.
3. Train a model on each combination of hyperparameters
4. Measure performance
5. Choose hyperparameter values for model with best performance
   - *Use the hyperparameter values for the model with lowest cross-validation error*

Tree Performance

- Trees are seldom the best models
- But, they are the simplest to understand
- Trees can be greatly improved by tuning model hyperparameters

Summary

- In this module, we
  - *defined Trees*
  - *examined regression Trees*
  - *examined classification Trees*
  - *discussed strengths and weaknesses of Trees*
  - *reviewed types of cross-validation*
  - *examined the virtues of tuning model hyperparameters*