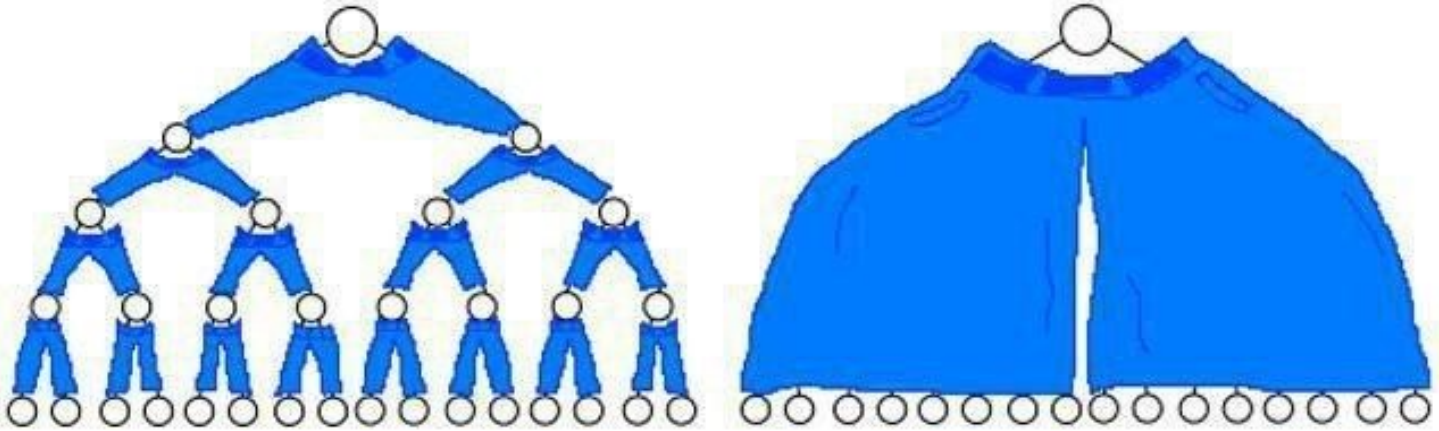
Bagging
Part A

Pavlos Protopapas

If a binary tree wore pants would he wear them

like this or like this?



Outline

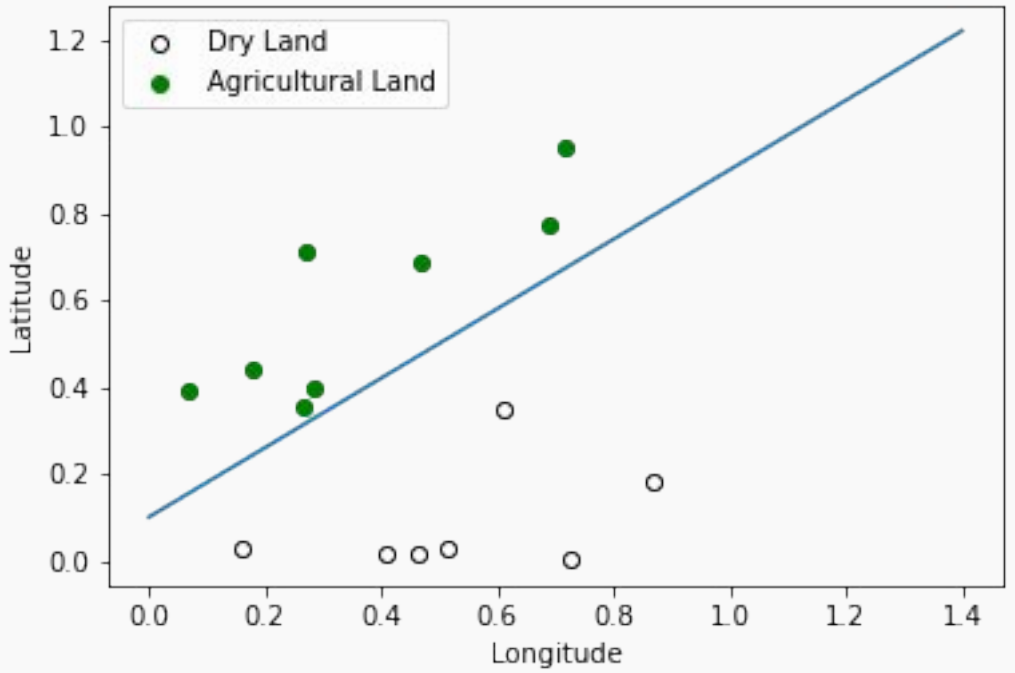
- Recap - Decision Trees
- Bagging
- Out of Bag Error (OOB)
- Variable Importance

Outline

- **Recap - Decision Trees**
- Bagging
- Out of Bag Error (OOB)
- Variable Importance

Recap: Geometry of Data

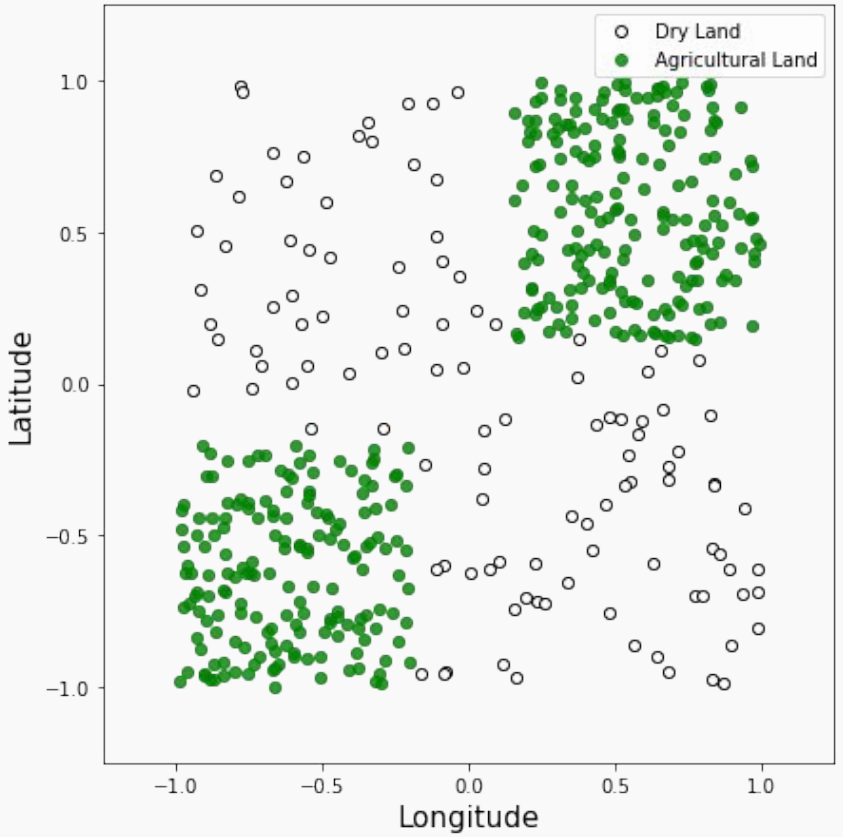
Question: Can you guess the equation that defines the decision boundary below?



$$Latitude = 0.8\ Longitude + 0.1$$
$$or$$
$$-0.8 Longitude + Latitude - 0.1 = 0$$

Recap: Geometry of Data

Complicated decision boundaries can not be explained with Log Regression.

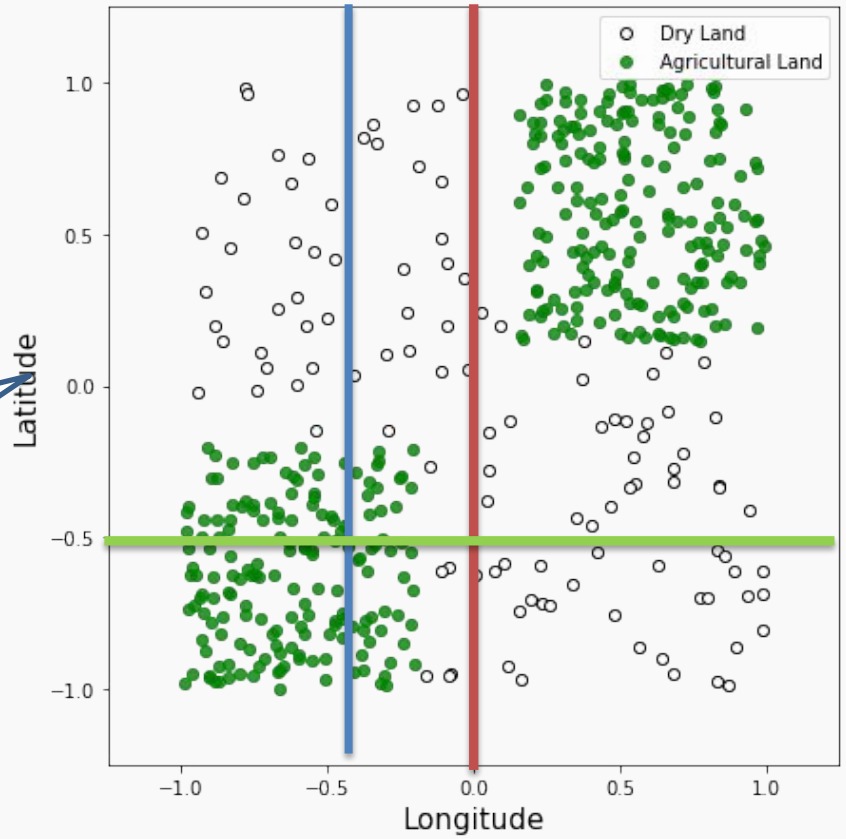


Recap: Geometry of Data

But can be described using decision trees.

Which one represents the first split of a decision tree?

It could be any of these lines – all three will give similar results!



Recap: Decision Trees

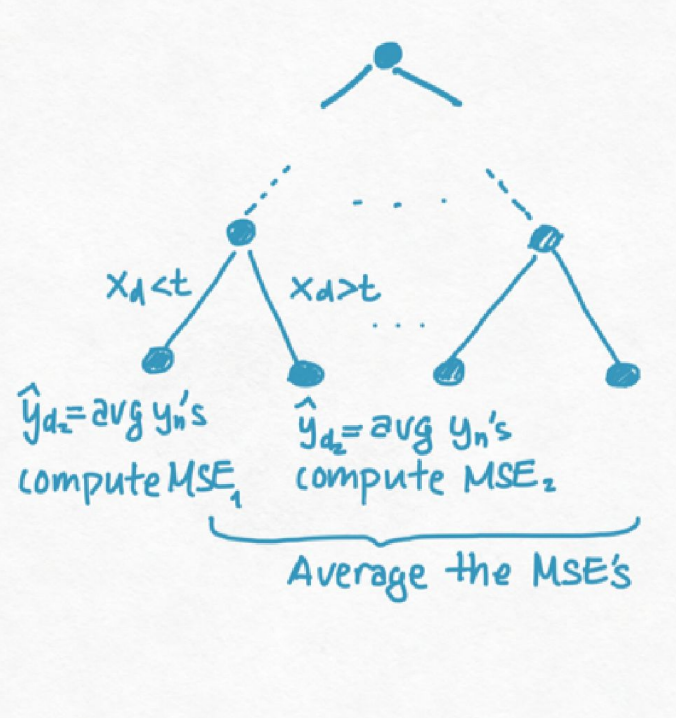
To learn a decision tree model, we take a greedy approach:

1. Start with a node containing all the data.

2. If **stopping condition** is not met:

   A. Choose the '**optimal**' predictor and threshold and divide the data in the node into two sets.

3. For each new node, repeat step 2.

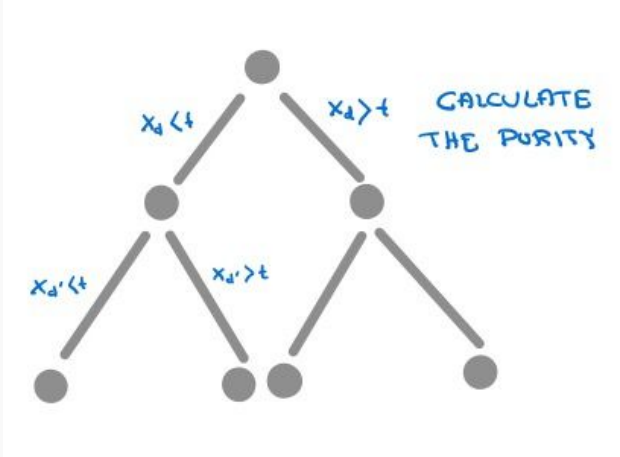


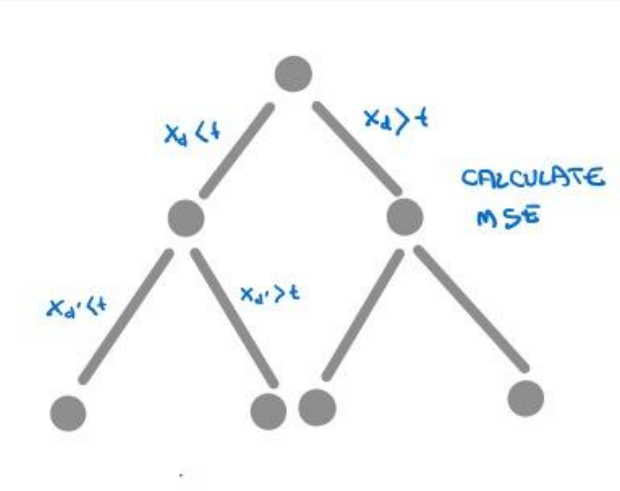
Recap: Splitting Criteria

For classification, purity of the regions is a good indicator the performance of the model.

Entropy as a splitting criterion minimizes the cross-entropy (greedy). Gini is also a splitting criteria.

For regression, we want to select a splitting criterion that promotes splits that improves the predictive accuracy of the model as measured by the MSE

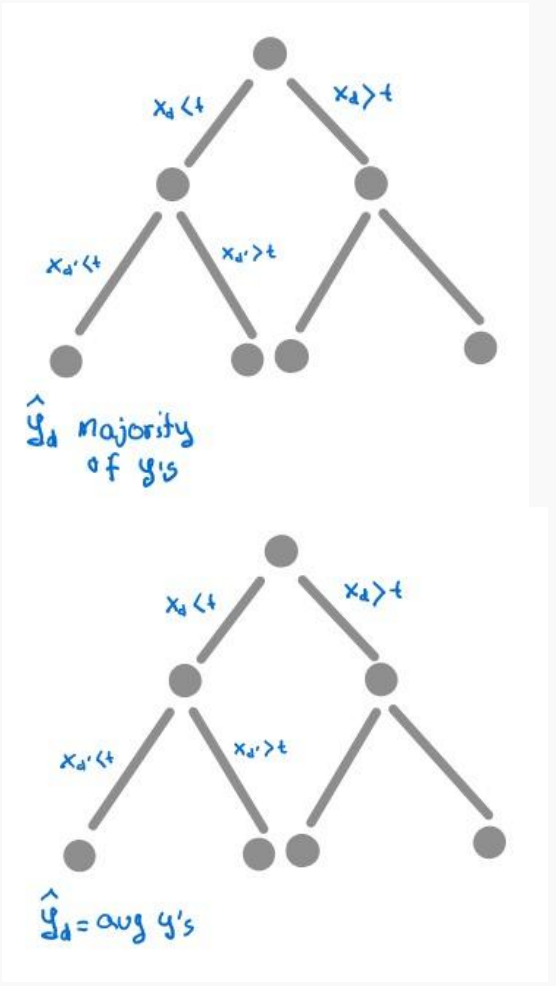




Recap: Prediction

For classification, we label each region in the model with the label of the class to which the **plurality** of the points within the region belong.

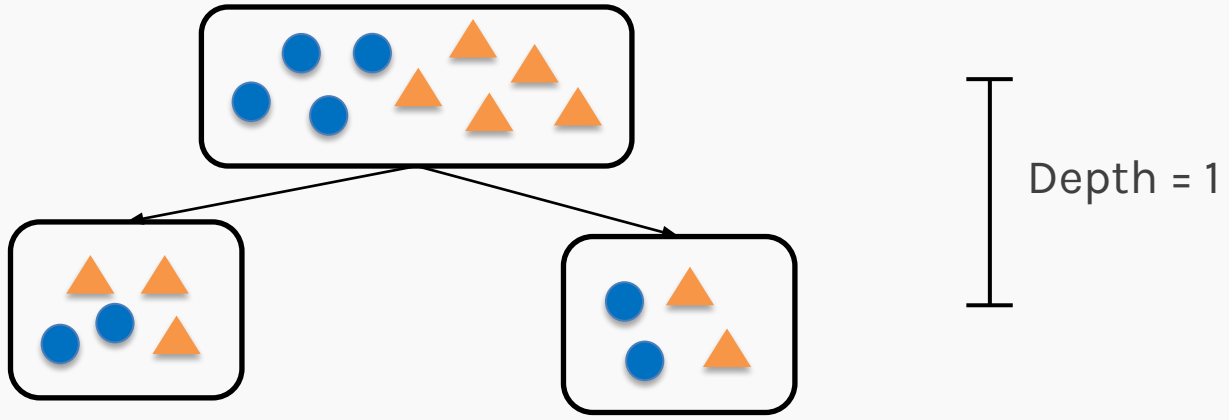
For regression, we predict with the **average** of the output values of the training points contained in the region.



Stopping Conditions

The most common stopping condition is to limit the maximum depth *(max_depth)* of the tree.

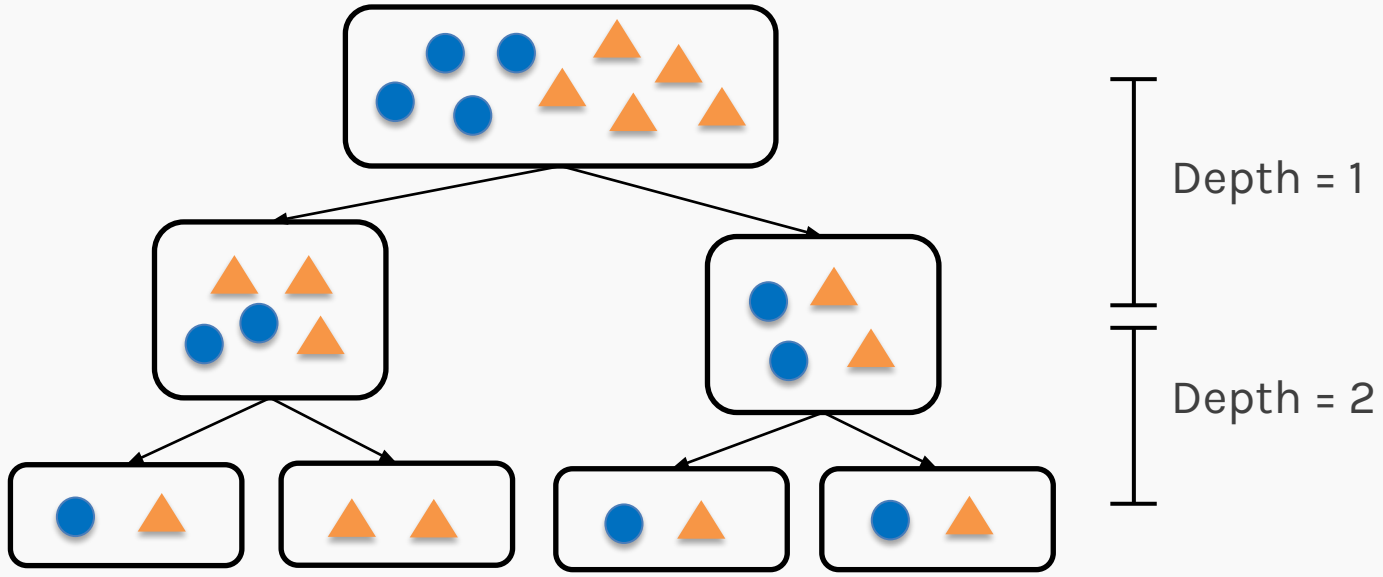
max_depth = 1



Stopping Conditions

The most common stopping condition is to limit the maximum depth *(max_depth)* of the tree.

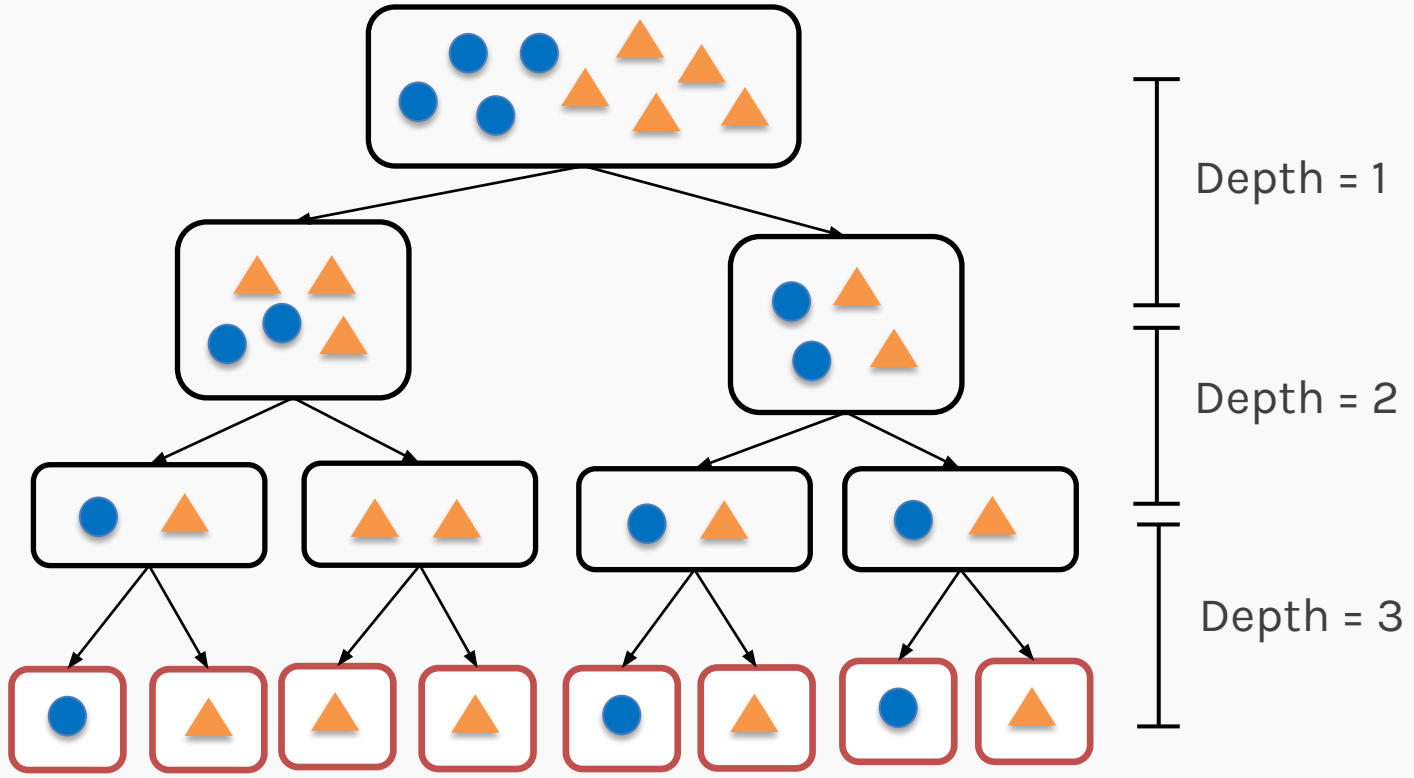
max_depth = 2



Stopping Conditions

The most common stopping condition is to limit the maximum depth *(max_depth)* of the tree.

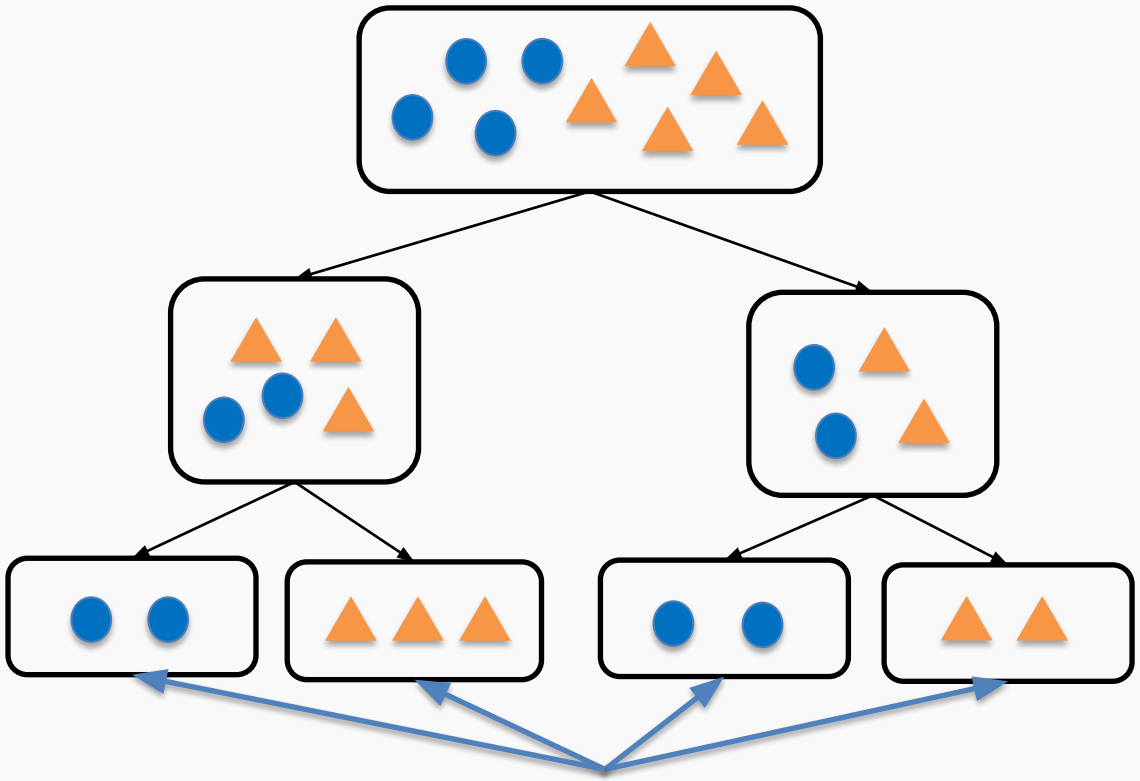
max_depth = 3



Stopping Conditions

Other common simple stopping conditions are:

- Don't split a region if all instances in the region belong to the same class.

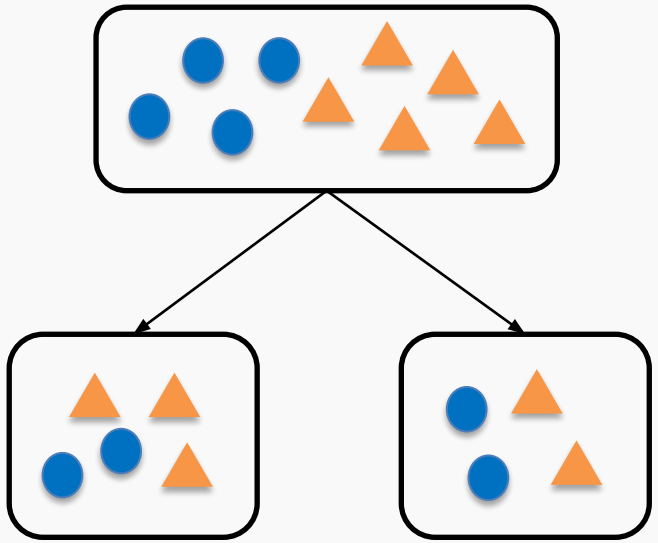


Stopping Conditions

Other common simple stopping conditions are:

- Don't split a region if the number of instances in any of the sub-regions will fall below pre-defined threshold *(min_samples_leaf)*.

min_samples_leaf = 4

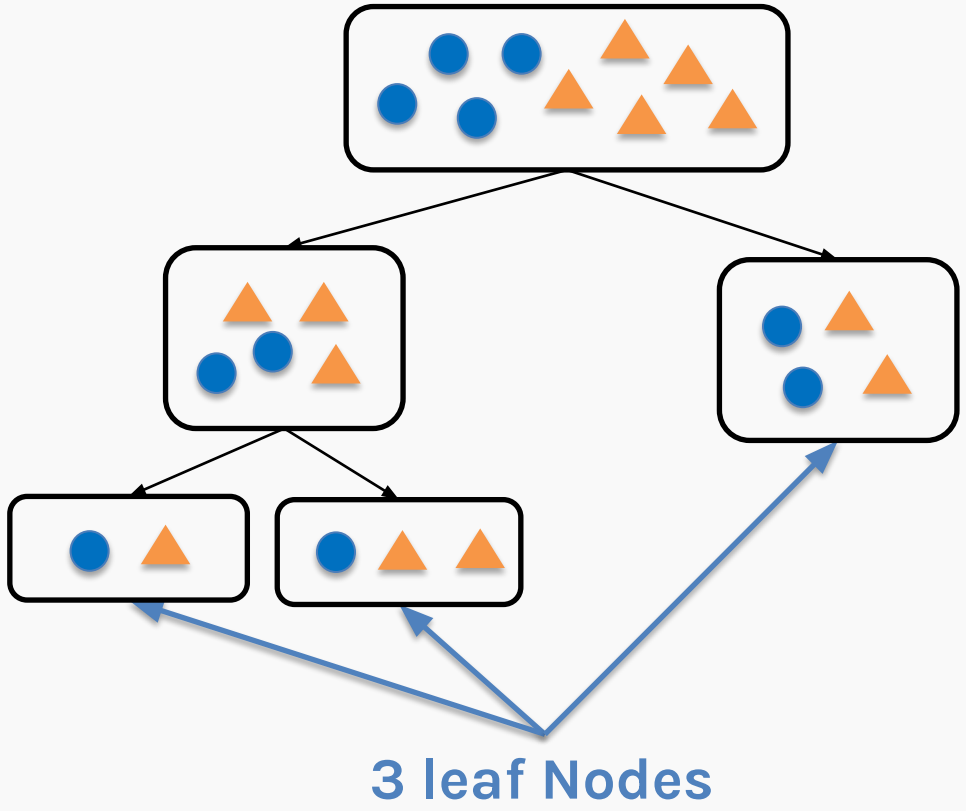


Stopping Conditions

Other common simple stopping conditions are:

- Don't split a region if the total number of leaves in the tree will exceed pre-defined threshold *(max_leaf_nodes)*.

max_leaf_nodes = 3



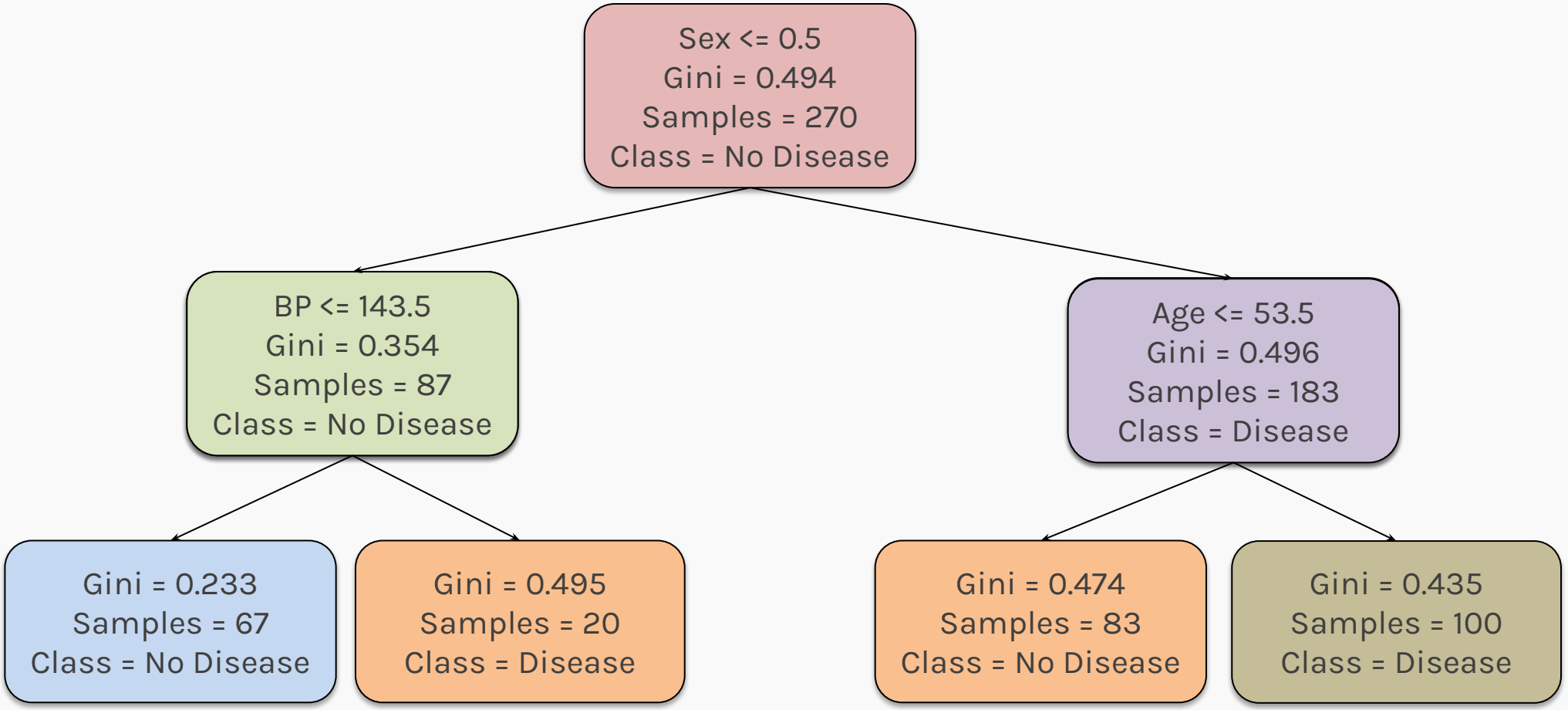
Stopping Conditions

Sklearn grows trees in **depth-first** fashion unless *max_leaf_nodes* is specified.
When *max_leaf_nodes* is specified, it is grown in a **best-first** fashion.

But what is depth-first and best-first fashion?

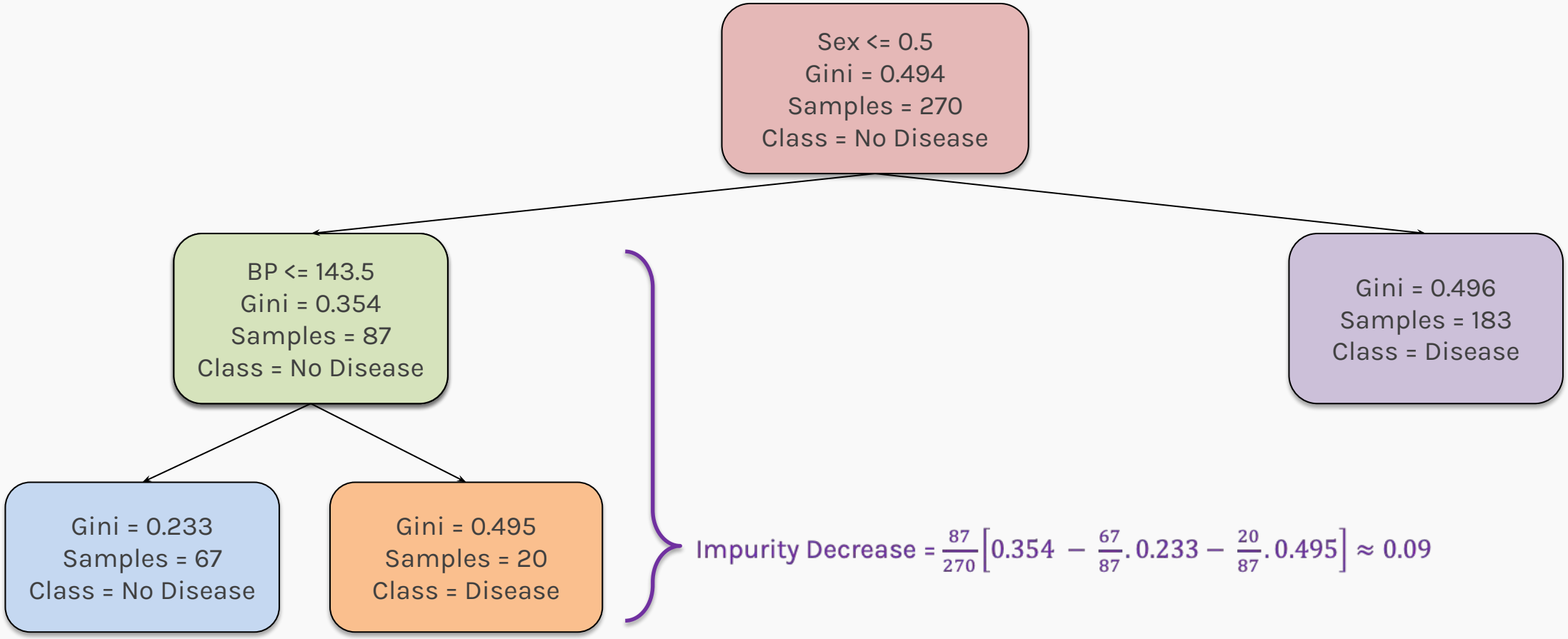
Example 1: Depth-first

Consider the following decision tree that predicts if a person has heart disease based on age, sex, BP and cholesterol:

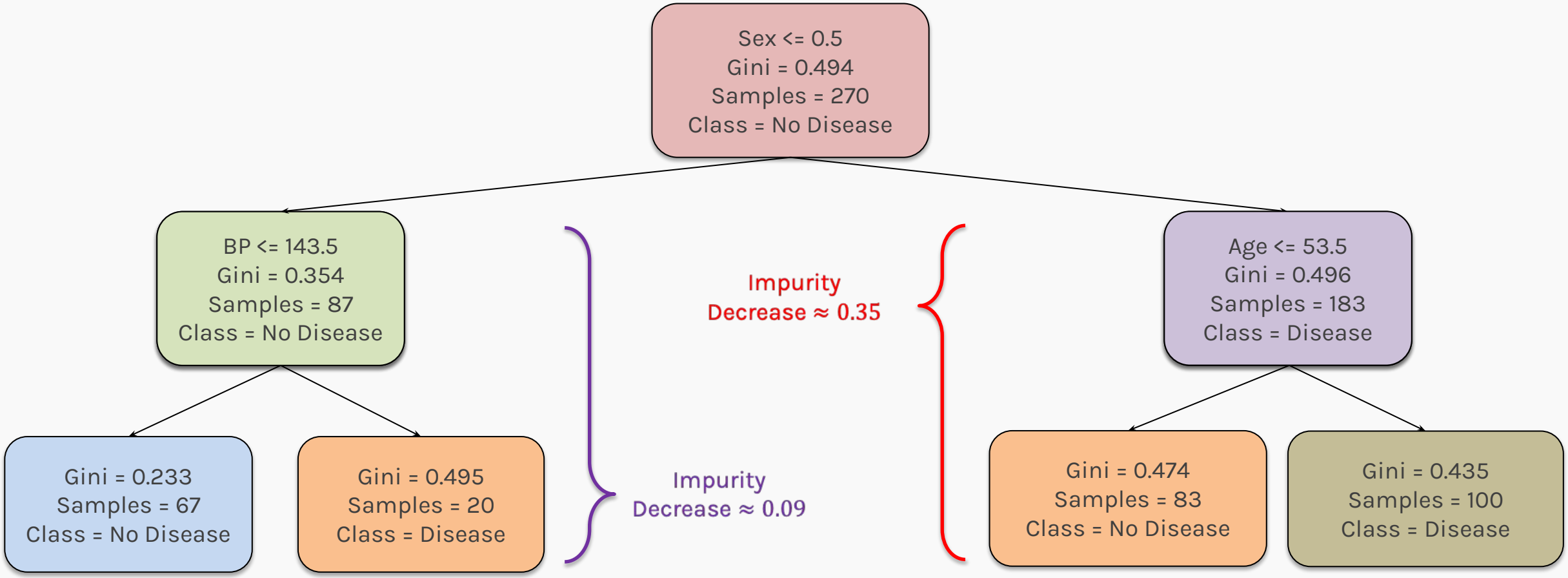


Example 1: Best-first growth

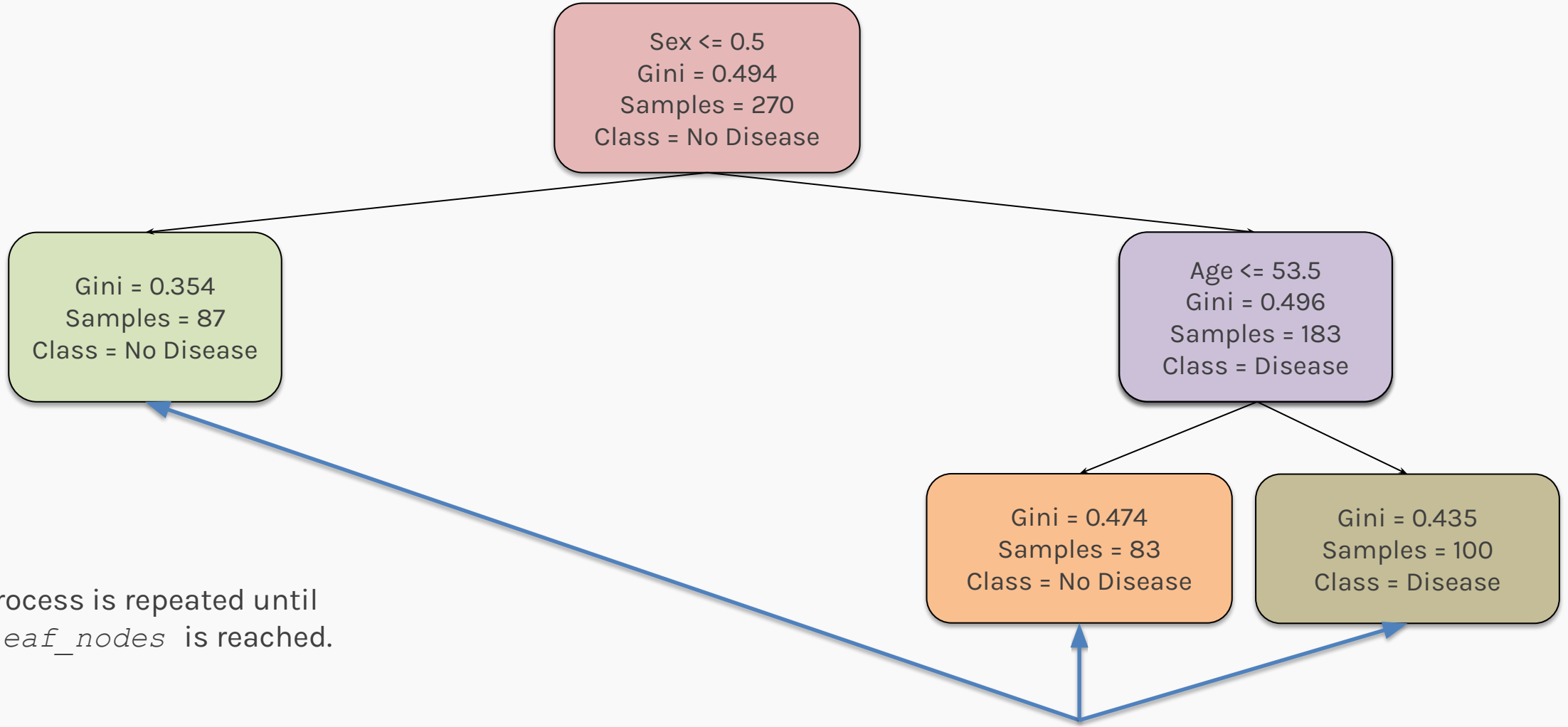
Sklearn determines the best split based on **impurity decrease**. The resulting tree will be the same when fully grown, just the order in which it is built is different.



Example 1: Best-first growth



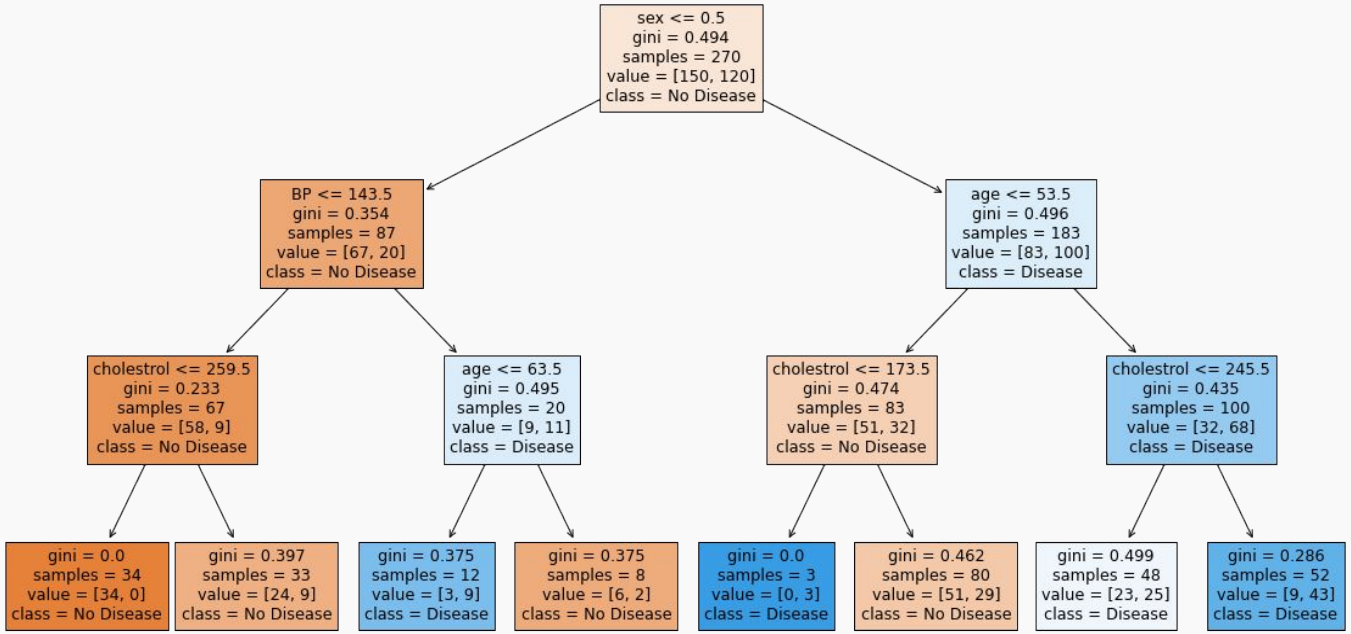
Example 1: Best-first growth



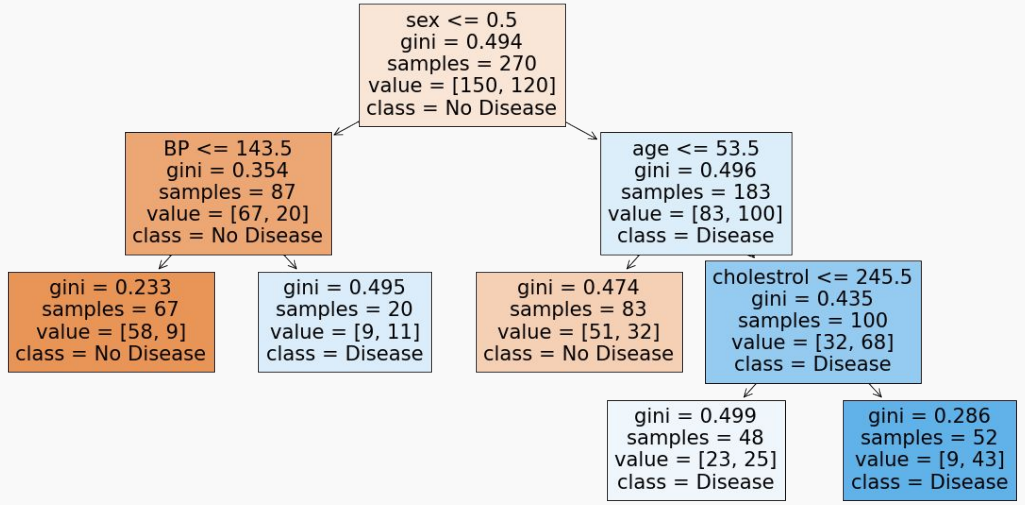
This process is repeated until *max_leaf_nodes* is reached.

Now we compare the impurity decrease of these 3 nodes!

Example 2: Depth-first vs Best-first growth



max_depth = 3



max_leaf_nodes = 5

Stopping Conditions

A more restrictive stopping condition is:

Compute the gain in purity, information or reduction in entropy of splitting a region R into R_1 and R_2:

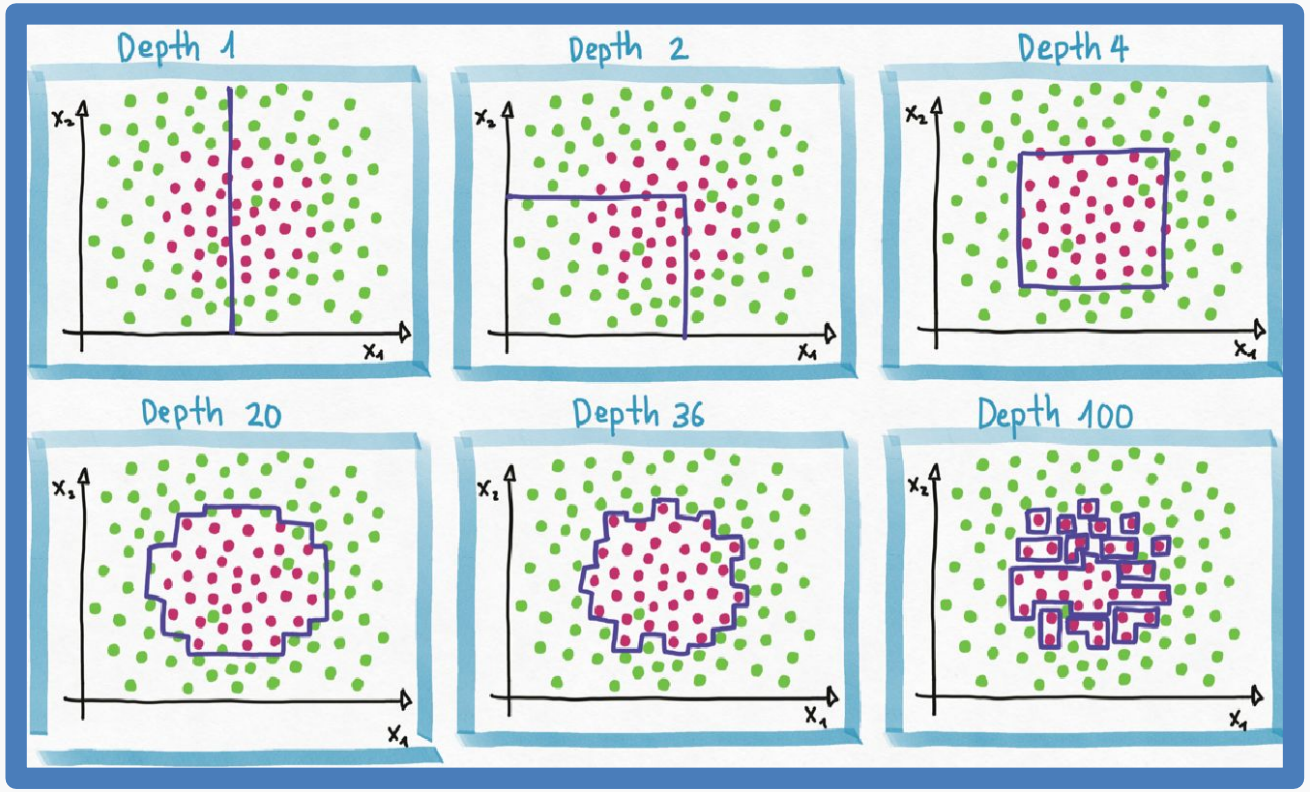
$$Gain(R) = \Delta(R) = m(R) - \frac{N_1}{N} m(R_1) - \frac{N_2}{N} m(R_2)$$

Classification Error/Gini Index/Entropy

Don't split if the gain is less than some pre-defined threshold (`min_impurity_decrease`).

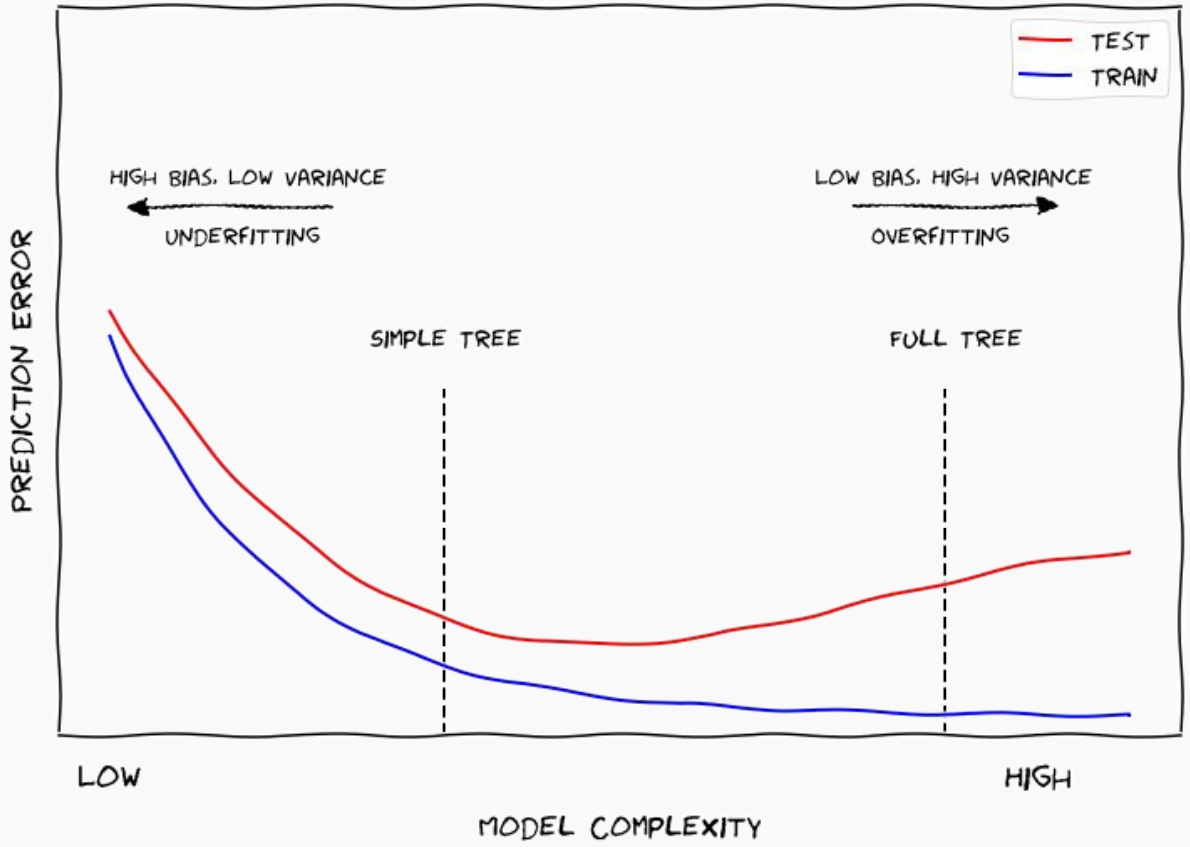
Overfitting

When a tree is too shallow, it cannot divide the input data into enough regions, so the model underfits. When the tree is too deep it cuts the input space into too many regions and fit to the noise of the data -> overfits.

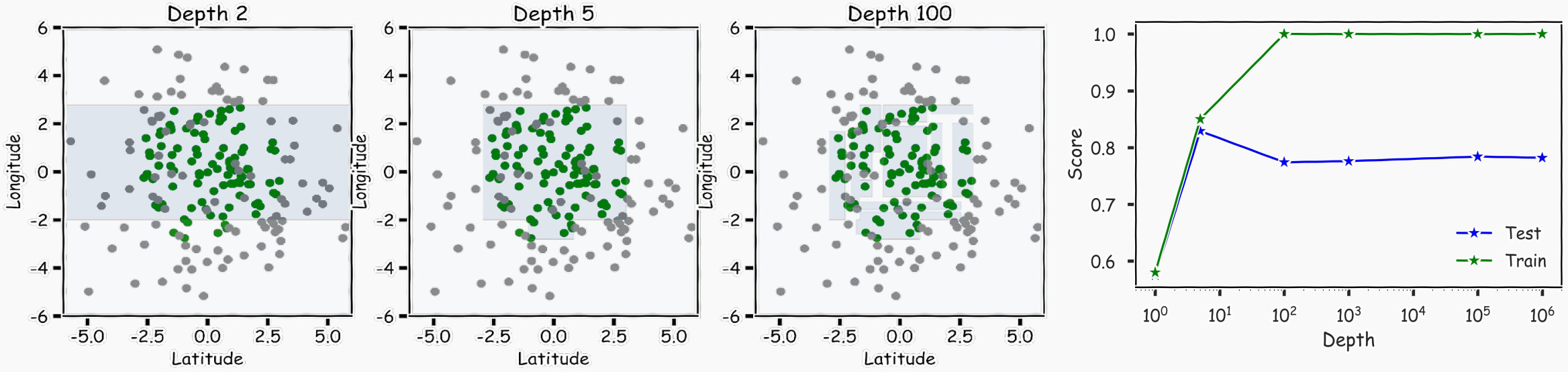


Overfitting

Avoid overfitting by pruning or limiting the depth of the tree and using CV.



Reduce the variance: Depth of the tree



We've seen that large trees have high variance and are prone to overfitting.

Use train/validation or cross validation to estimate the best depth.

Limitations of Decision Tree Models

- Decision trees models are highly interpretable and fast to train, using our greedy learning algorithm.
- However, to **capture a complex decision boundary** (or approximate a complex function), we need to use a large tree (since each time we can only do axis-aligned splits).
- We've seen that large trees have high variance and are prone to overfitting.

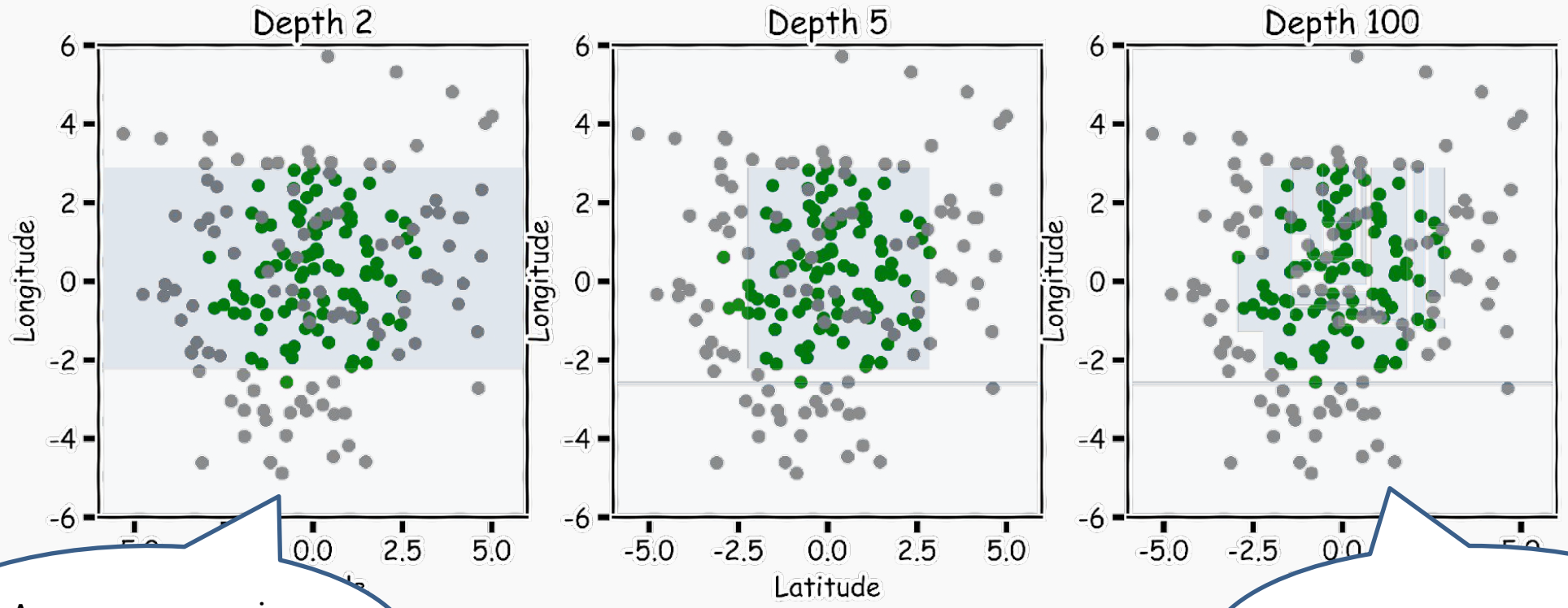
For these reasons, in practice, decision tree models often underperform when compared with other classification or regression methods.

Outline

- Review of Decision Trees
- **Bagging**
- Out of Bag Error (OOB)
- Variable Importance

Motivation for Bagging

Decision tree models often underperform when compared with other classification or regression methods in situations of complex boundaries & variance.

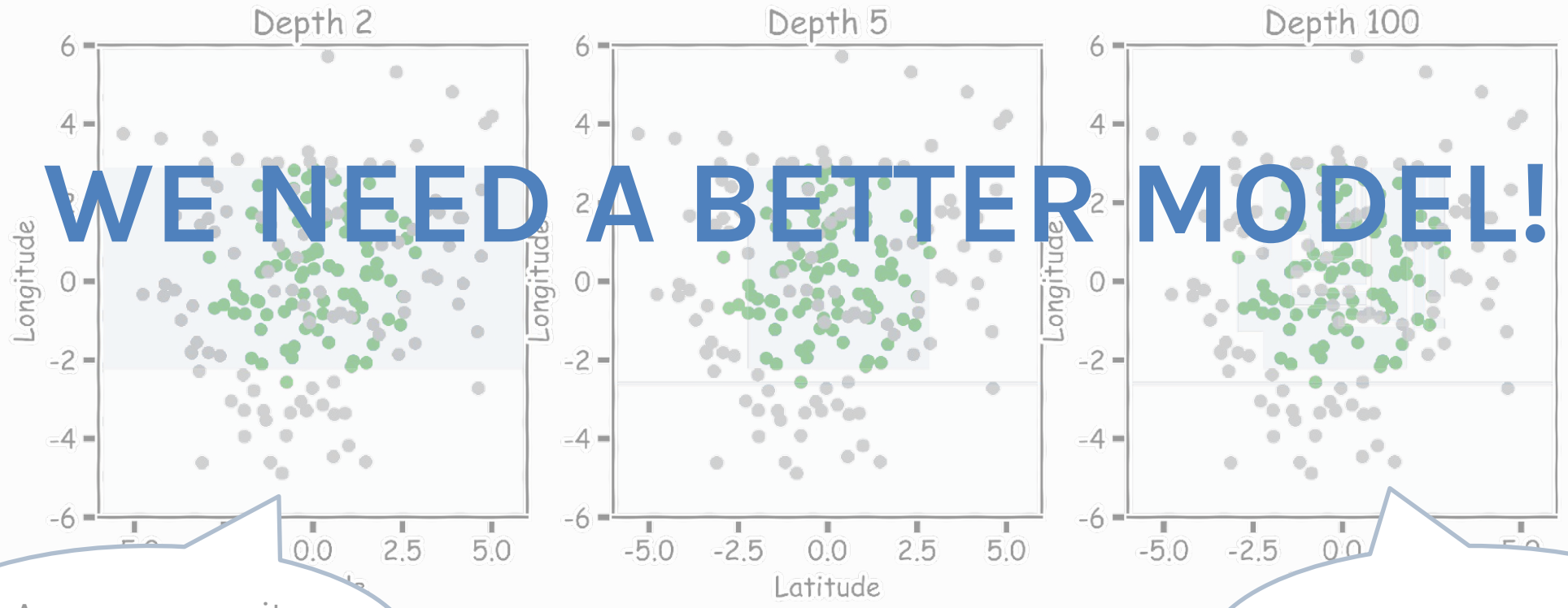


As you can see, in certain cases it underfits.

...and in certain cases it overfits!

Motivation for Bagging

Decision tree models often underperform when compared with other classification or regression methods in situations of complex boundaries & variance.



WE NEED A BETTER MODEL!

As you can see, it certain cases it underfits.

...and in certain cases it overfits!

ENSEMBLE LEARNING

And

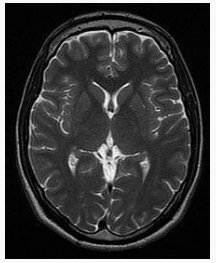
BOOTSTRAPPING!

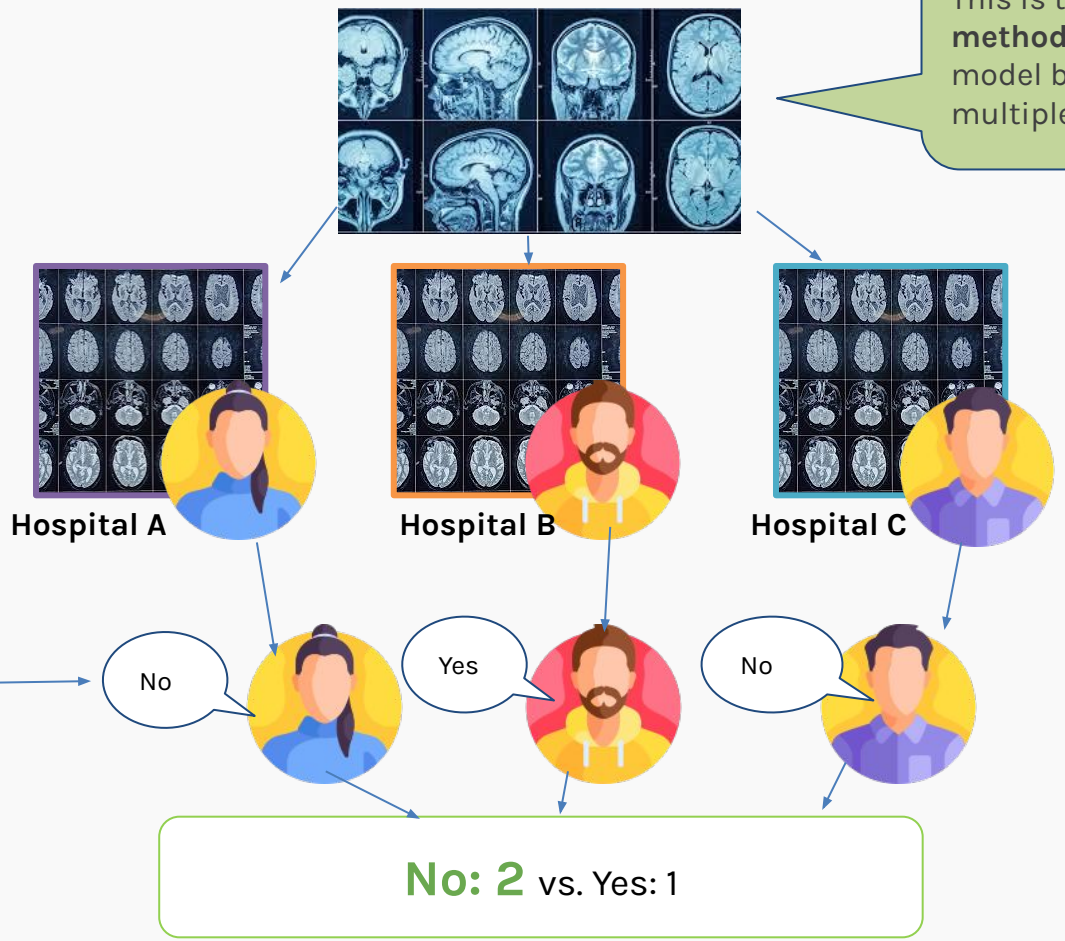
Ensemble Learning - Intuition

MRI Scans of Brain Tumor Patients

Doctors trained on various samples of Brain Tumor MRI Scans in the educational institutions they studied in.

Mr. X sends his MRI Scan forward to get the diagnosis from 3 doctors just to make sure that the results are more confident.

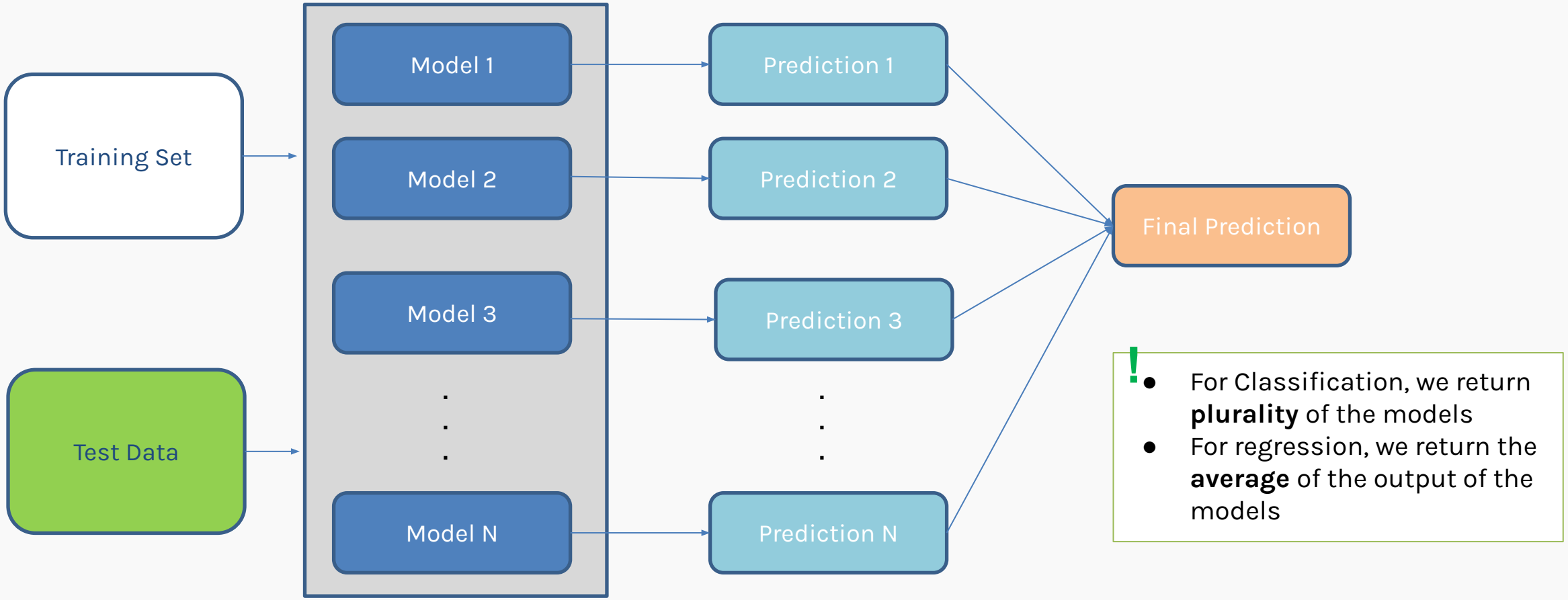




This is the intuition behind **ensemble method**, a method of building a single model by training and aggregating multiple models.

Ensemble Learning

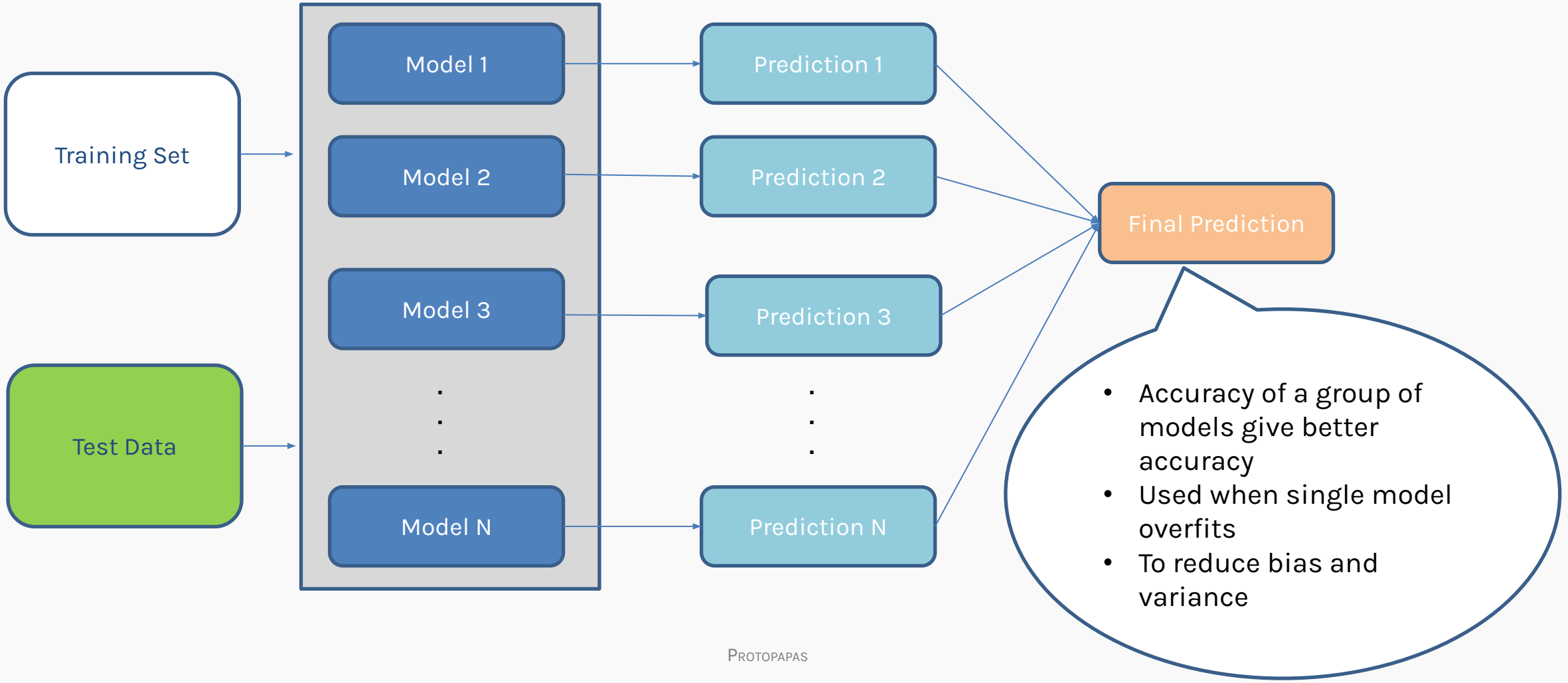
Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.



- For Classification, we return **plurality** of the models
- For regression, we return the **average** of the output of the models

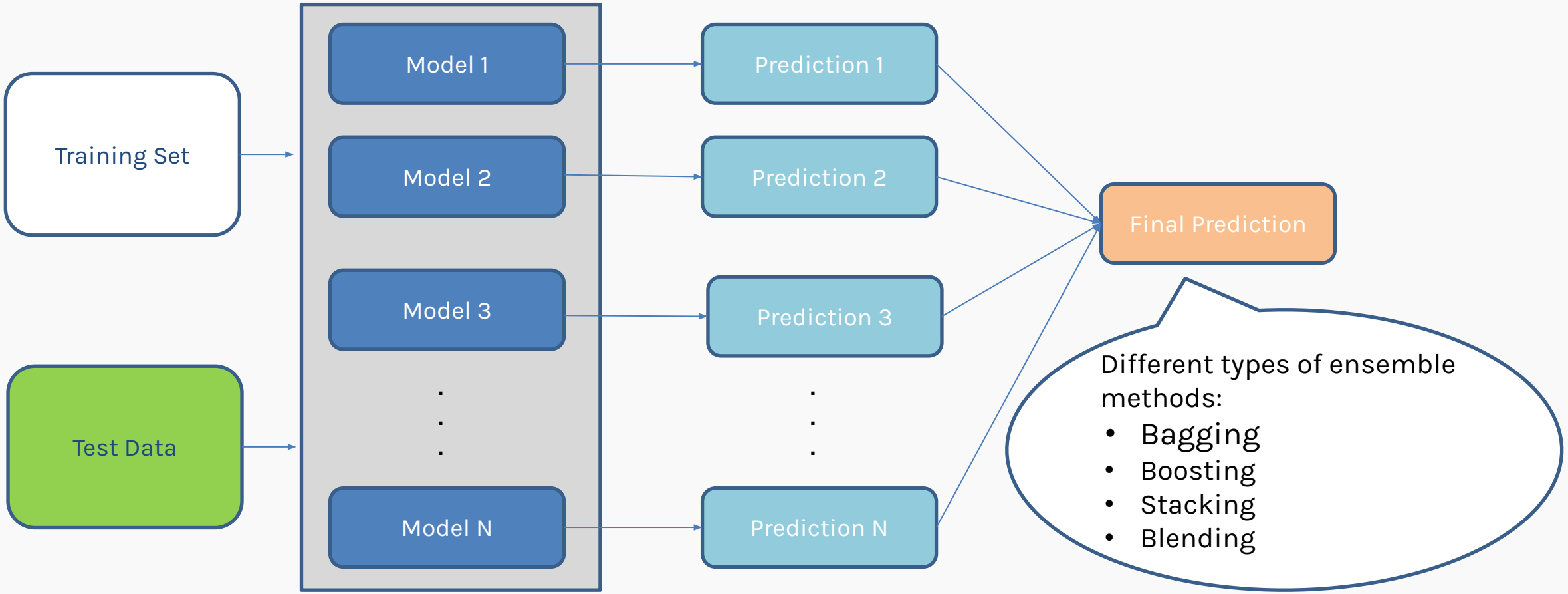
Ensemble Learning

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.



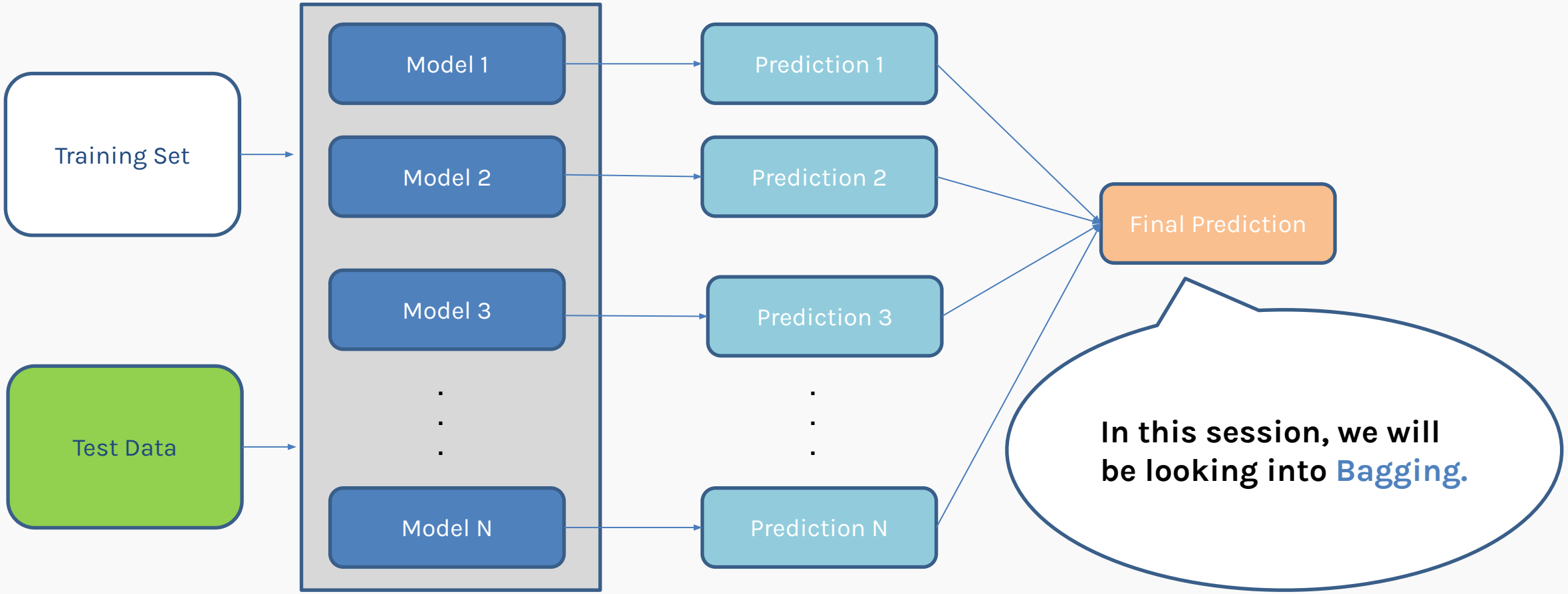
Ensemble Learning

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.



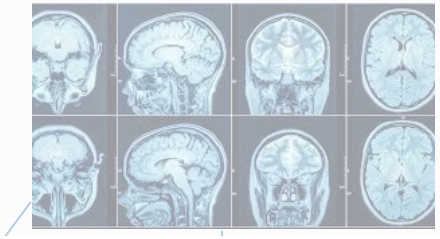
Ensemble Learning

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model.



Ensemble Learning - Intuition

MRI Scans of Brain Tumor Patients

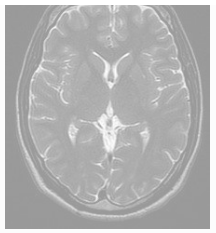


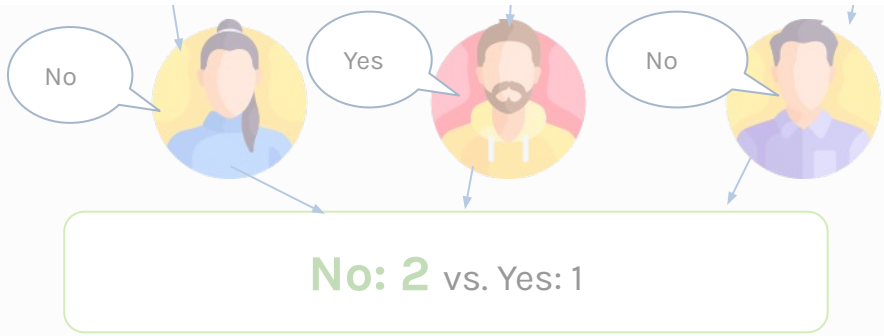
This is the intuition behind **ensemble method**, a method of building a single model by training and aggregating multiple models.

Docto
samp
Scans
instit

BUT WAIT, HOW DO THESE DOCTORS OR MODELS LEARN? WHAT DATASET DO THEY SEE?

Mr. X sends his MRI Scan forward to get the diagnosis from 3 doctors just to make sure that the results are more confident.





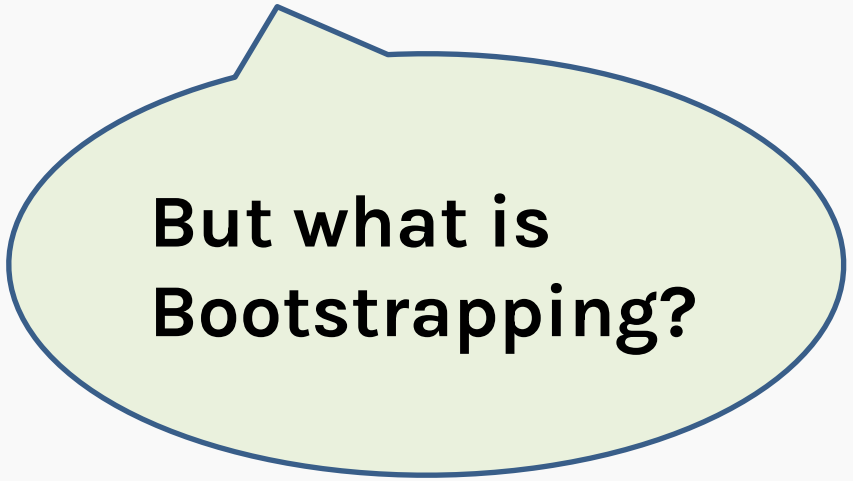
Bootstrap - Motivation

In practice, we do not have different datasets or different doctors.

We want multiple models (doctors) train on different datasets, however we have only one dataset.

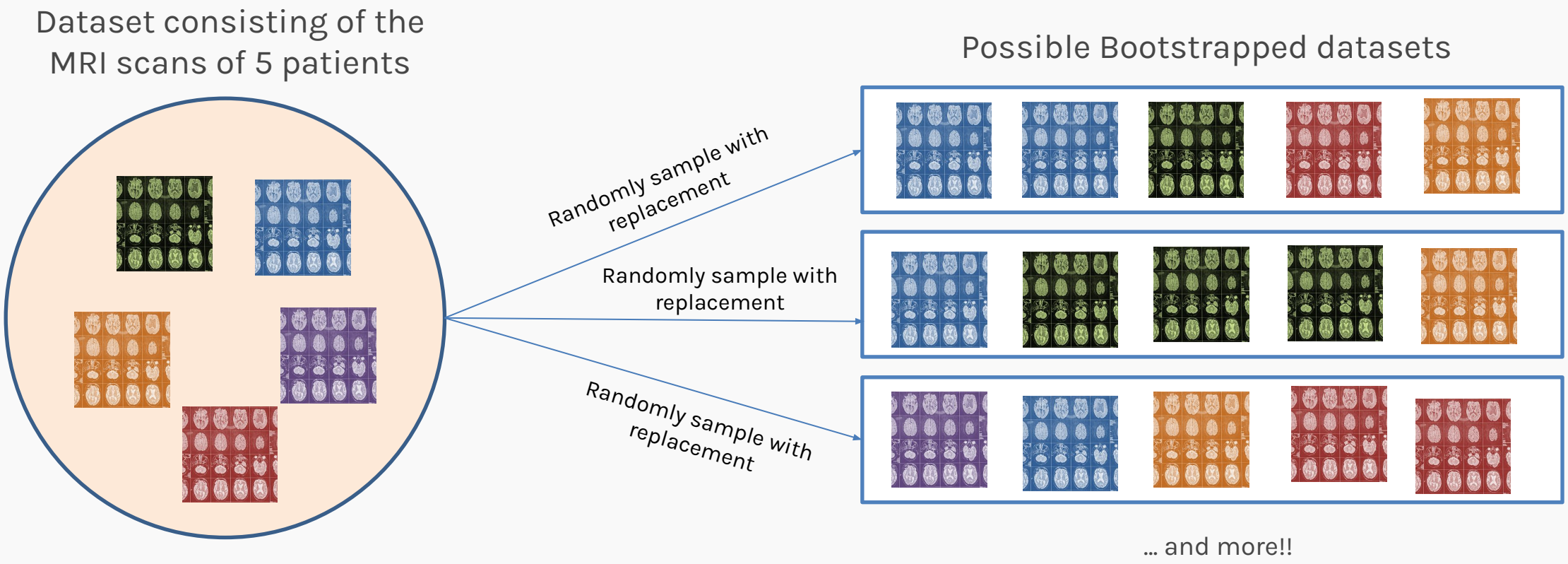
How can we generate datasets?

Bootstrapped datasets!



Bootstrapping

Bootstrapping is the process of **sampling with replacement** from a data set and performing calculations on such multiple bootstrapped datasets to get an overall aggregate inference.



Bootstrapping

Bootstrapping is the process of **sampling with replacement** from a data set and performing calculations on such multiple bootstrapped datasets for aggregate inference.

Dataset consisting of the MRI scans of 5 patients

Possible Bootst

Randomly choosing data and allowing for duplication is called **Sampling with replacement!**

Note that the bootstrapped dataset consists of the same amount of data as the original dataset.

... and more!!

Combining both Ensemble and Bootstrapping together …

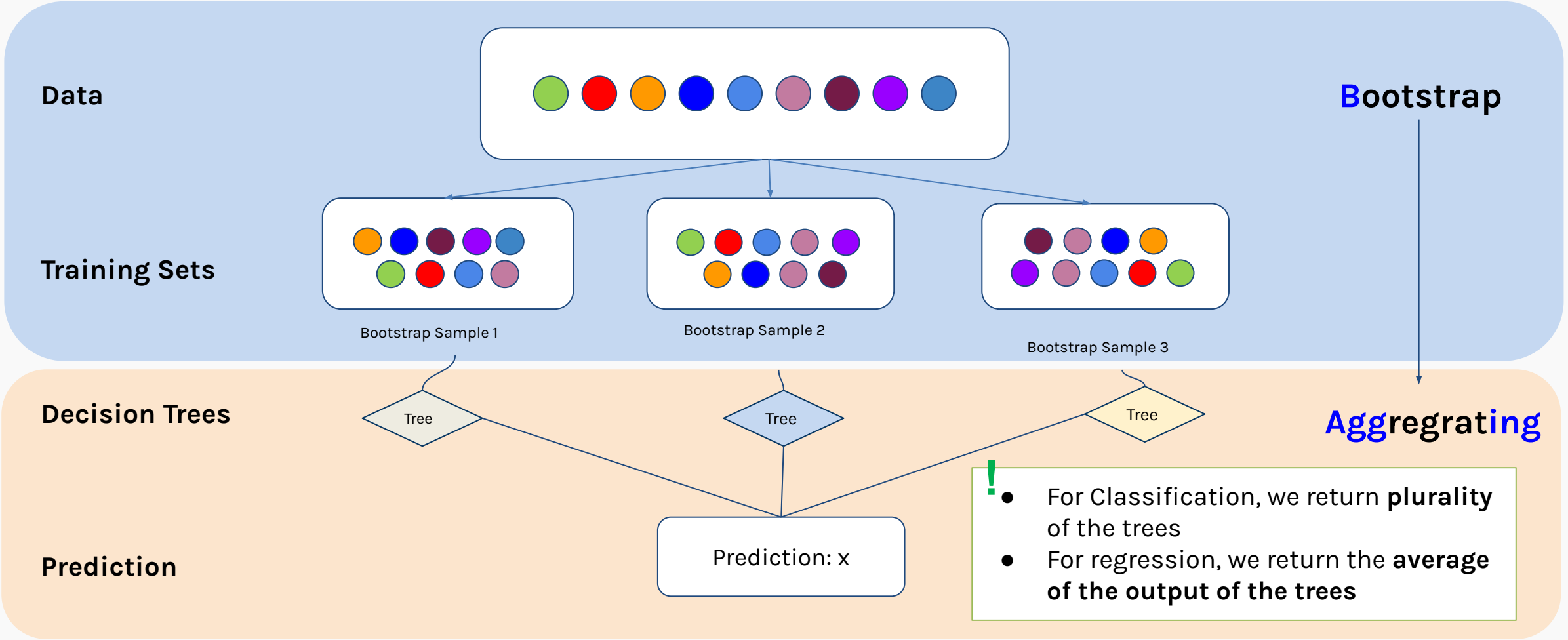
Bootstrap

Aggregation

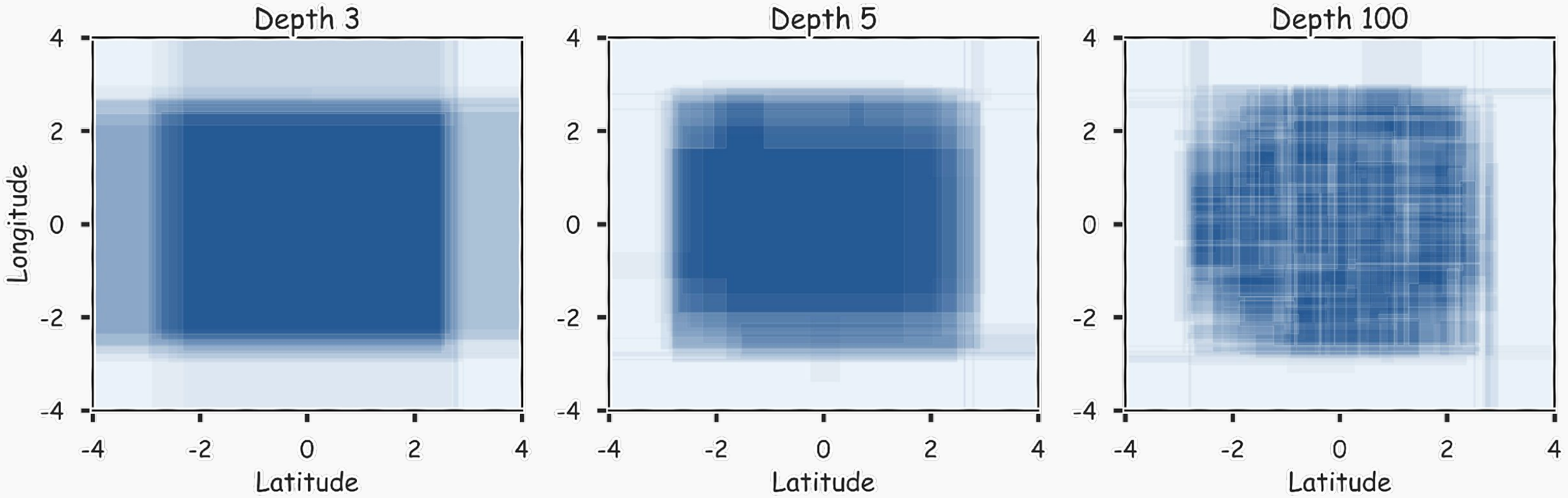
Bagging!

We get Bagging!

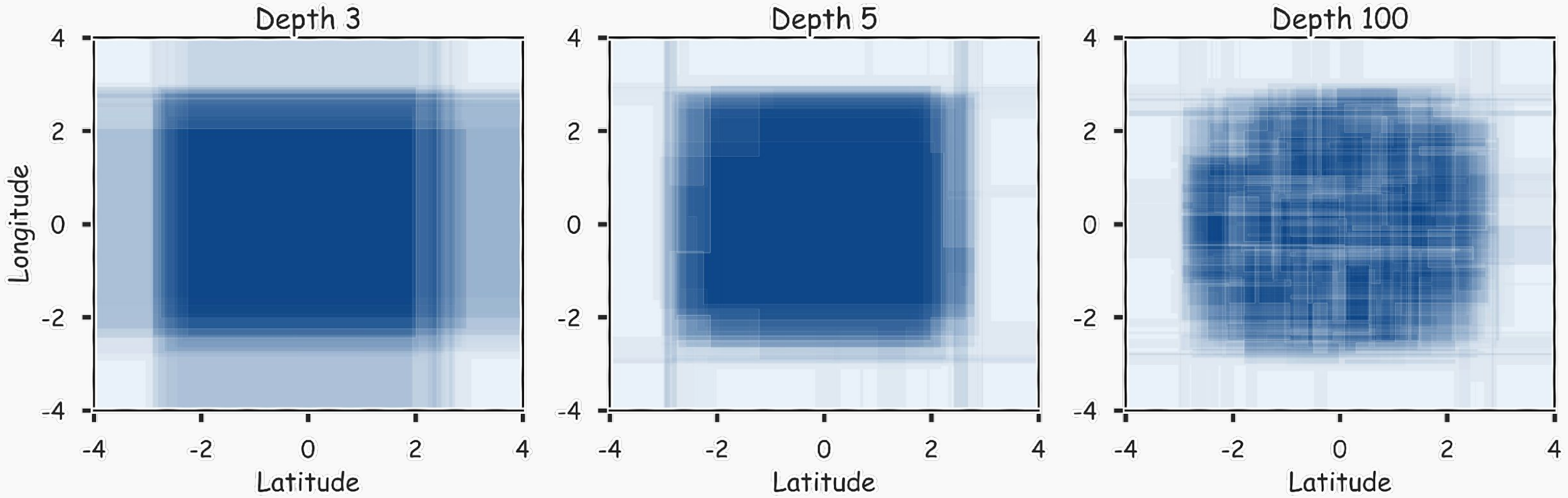
This method is called ***Bagging*** (Breiman, 1996), short for, of course, **B**ootstrap **Agg**regating.



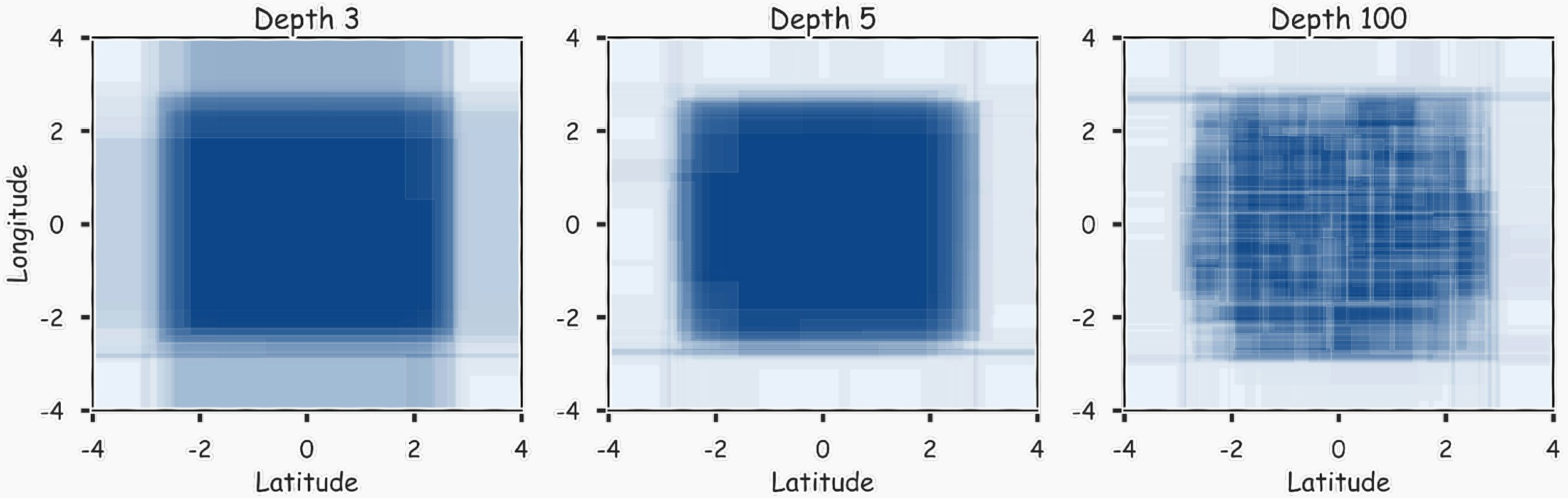
20 magic realisms



100 magic realisms

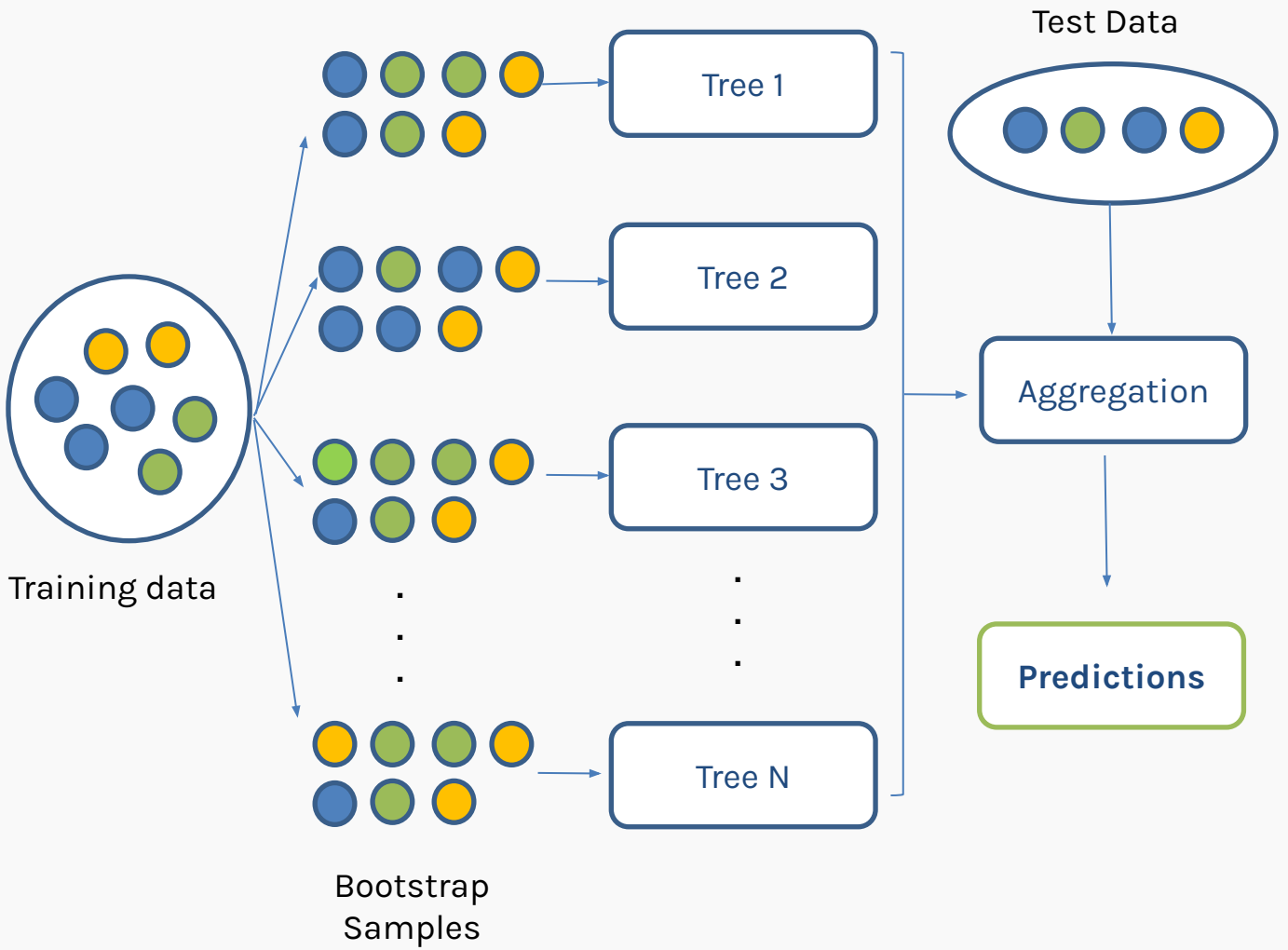


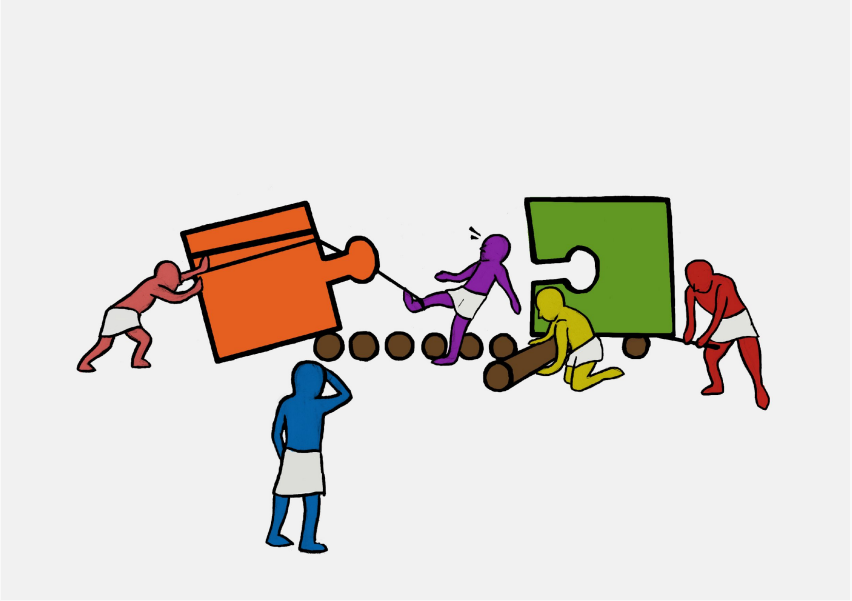
300 magic realisms



Bagging - Bootstrap + Aggregate

1. **Bootstrap**: we generate multiple samples of training data, via bootstrapping. We train a deeper decision tree on each sample of data.
2. **Aggregate:** for a given input, we output the averaged outputs of all the models for that input.

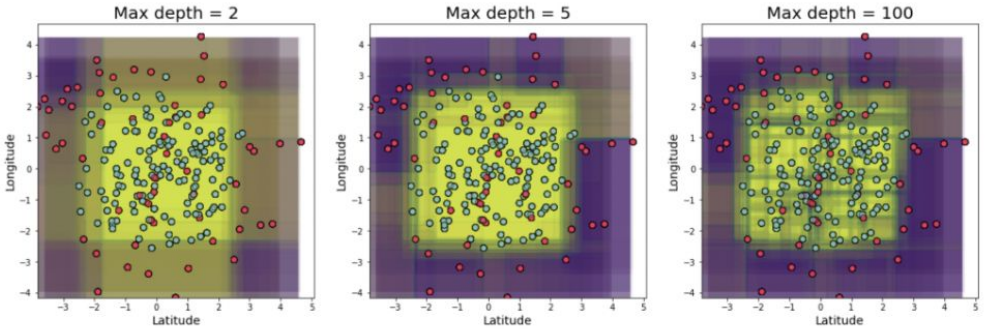




🏋️ Exercise: A.1 - Bagging Classification with Decision Boundary

The goal of this exercise is to use **Bagging** (Bootstrap Aggregated) to solve a classification problem and visualize the influence on Bagging on trees with varying depths.

Your final plot will resemble the one below.



Instructions:

- Read the dataset `agriland.csv`.
- Assign the predictor and response variables as `X` and `y`.
- Split the data into train and test sets with `test_split=0.2` and `random_state=44`.
- Fit a single `DecisionTreeClassifier()` and find the accuracy of your prediction.
- Complete the helper function `prediction_by_bagging()` to find the average predictions for a given number of bootstraps.
- Perform `Bagging` using the helper function, and compute the new accuracy.
- Plot the accuracy as a function of the number of bootstraps.
- Use the helper code to plot the decision boundaries for varying `max_depth`

Advantages of Bagging

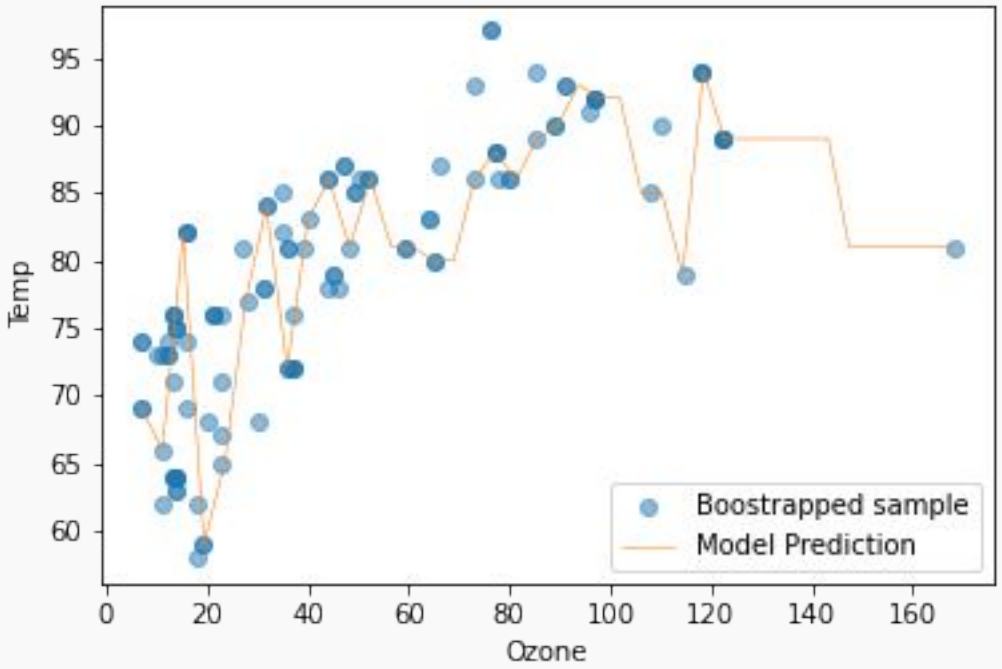
Bagging enjoys the benefits of:

1. High expressiveness - by using deeper trees each model is able to approximate complex functions and decision boundaries.

2. Low variance - averaging the prediction of all the models reduces the variance in the final prediction, assuming that we choose a sufficiently large number of trees.

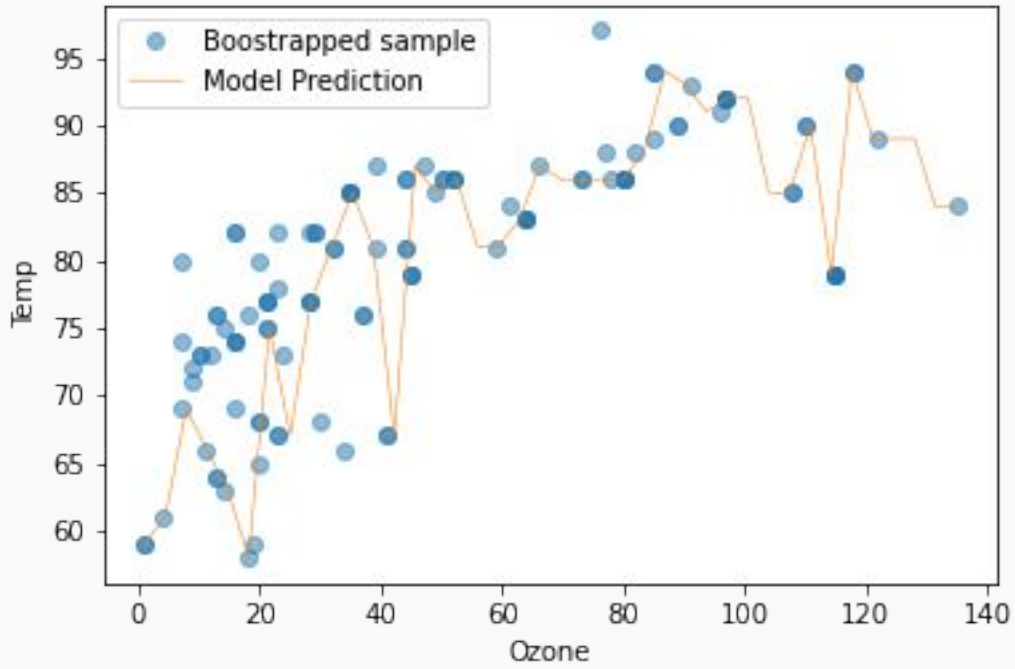
Regression in Bagging

Prediction from Decision Tree #1 with Bootstrapped Sample #1



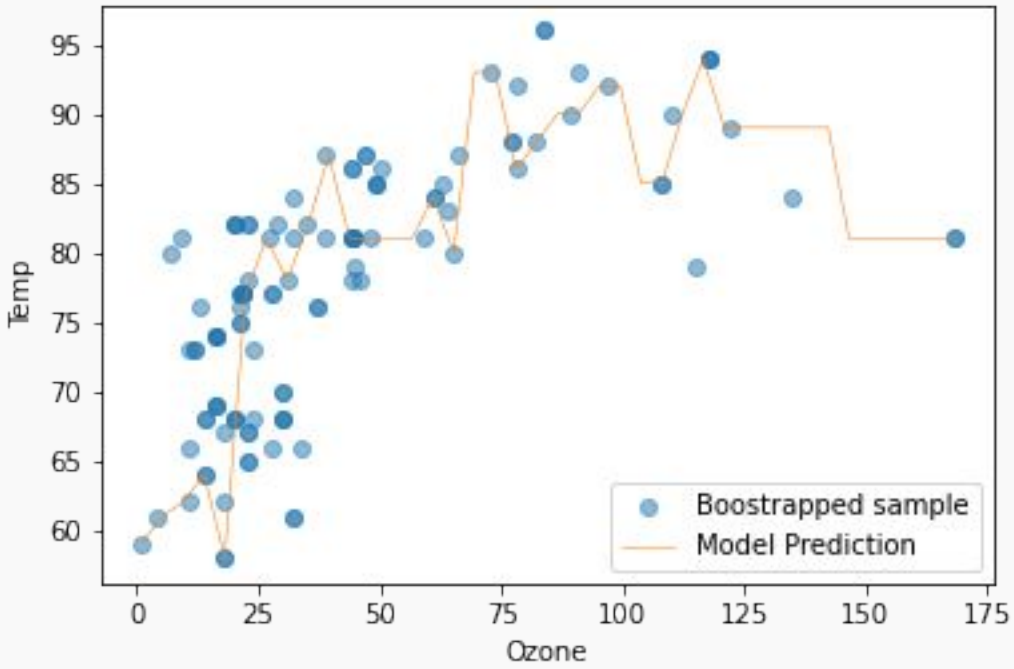
Regression in Bagging

Prediction from Decision Tree #2 with Bootstrapped Sample #2



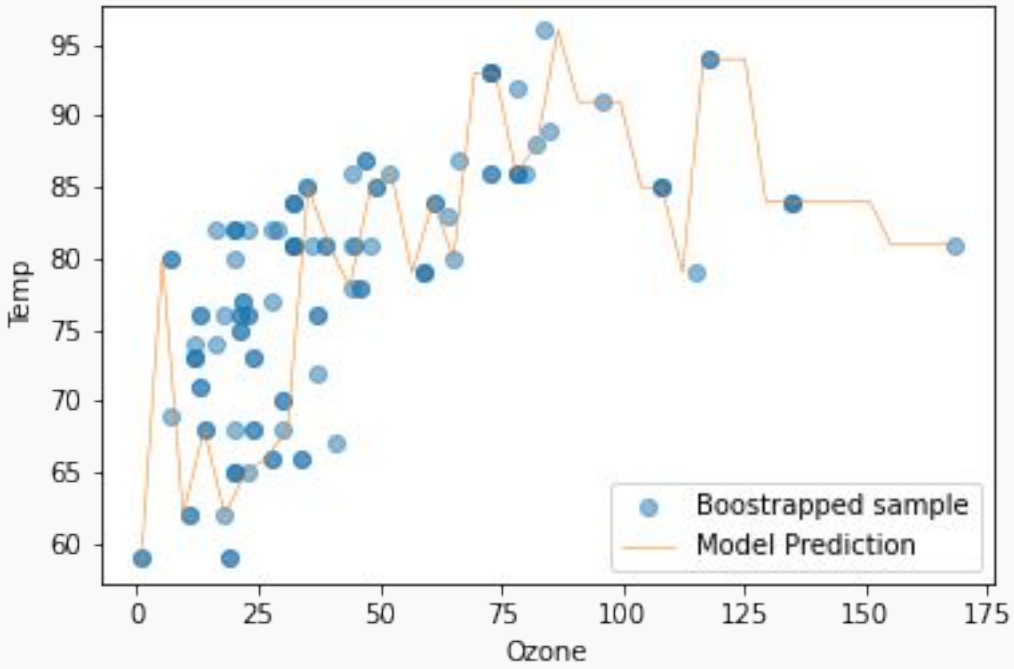
Regression in Bagging

Prediction from Decision Tree #3 with Bootstrapped Sample #3



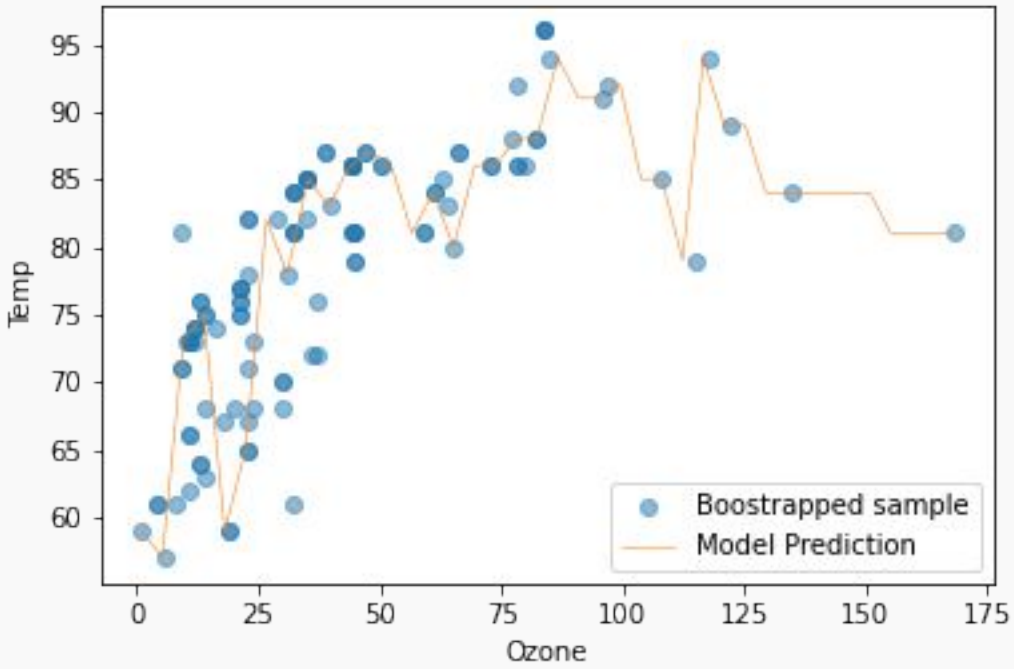
Regression in Bagging

Prediction from Decision Tree #4 with Bootstrapped Sample #4



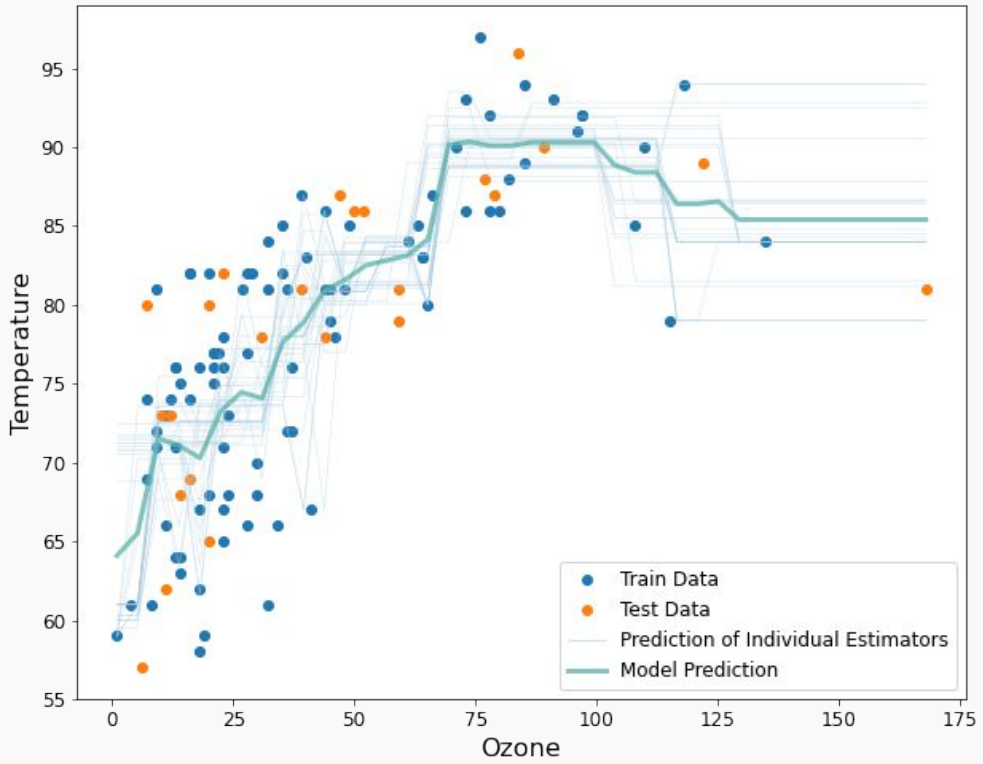
Regression in Bagging

Prediction from Decision Tree #5 with Bootstrapped Sample #5



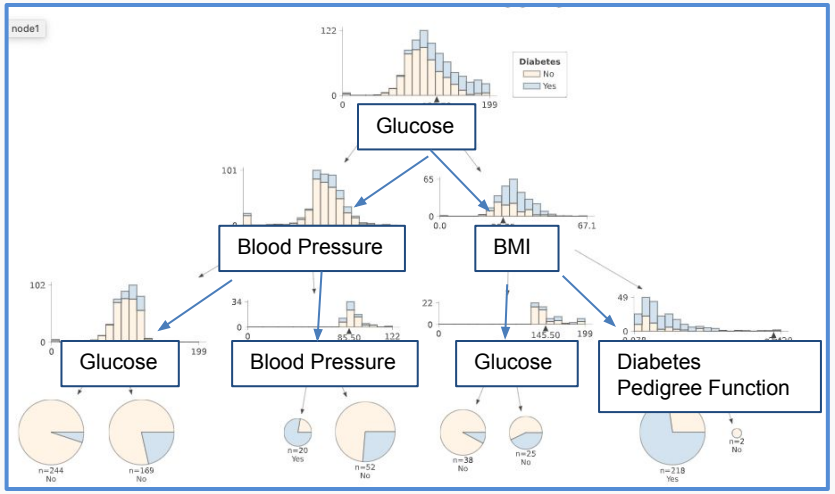
Regression in Bagging

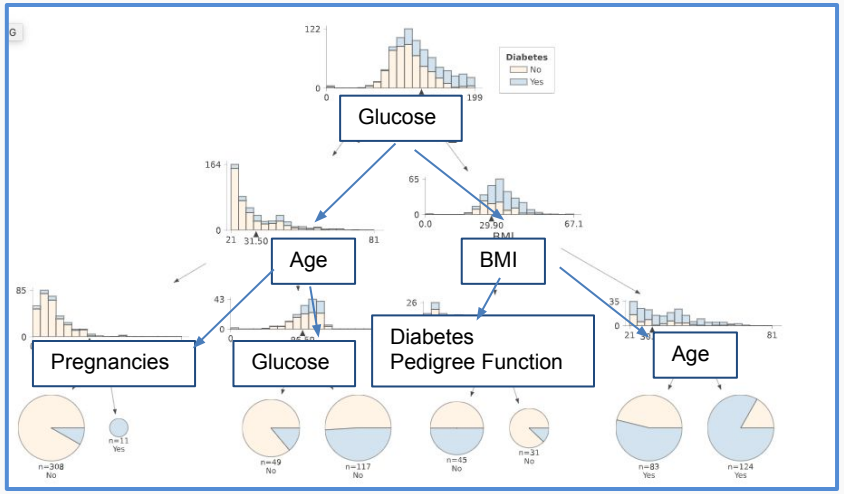
The prediction by the Bagging Regressor model is average of all the individual predictions of the trees or estimators.

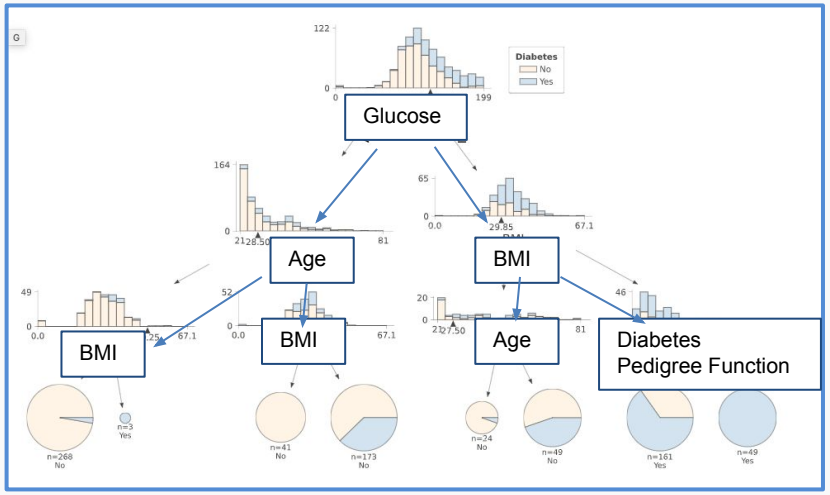


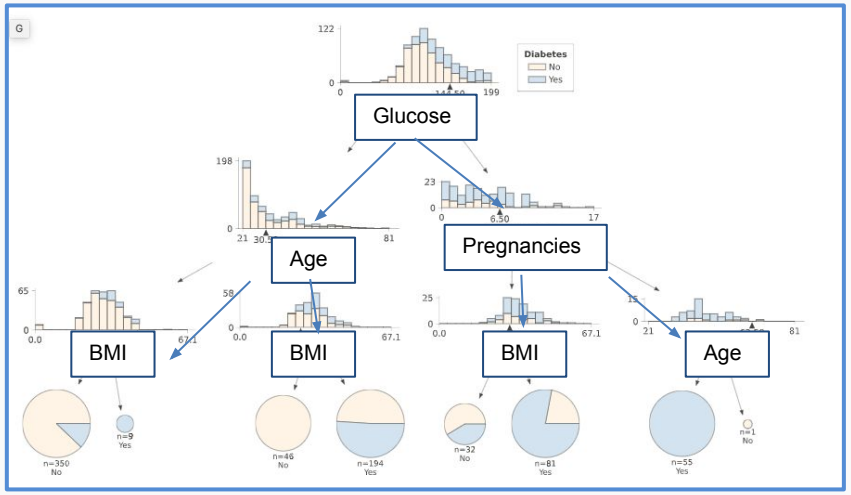
Resulting Prediction by the Bagging Regressor

Classification in Bagging









For each bootstrap, we build a decision tree. The results is a combination (majority) of the predictions from all trees.

Created by: Dr. Rahul Dave

Drawbacks of Bagging

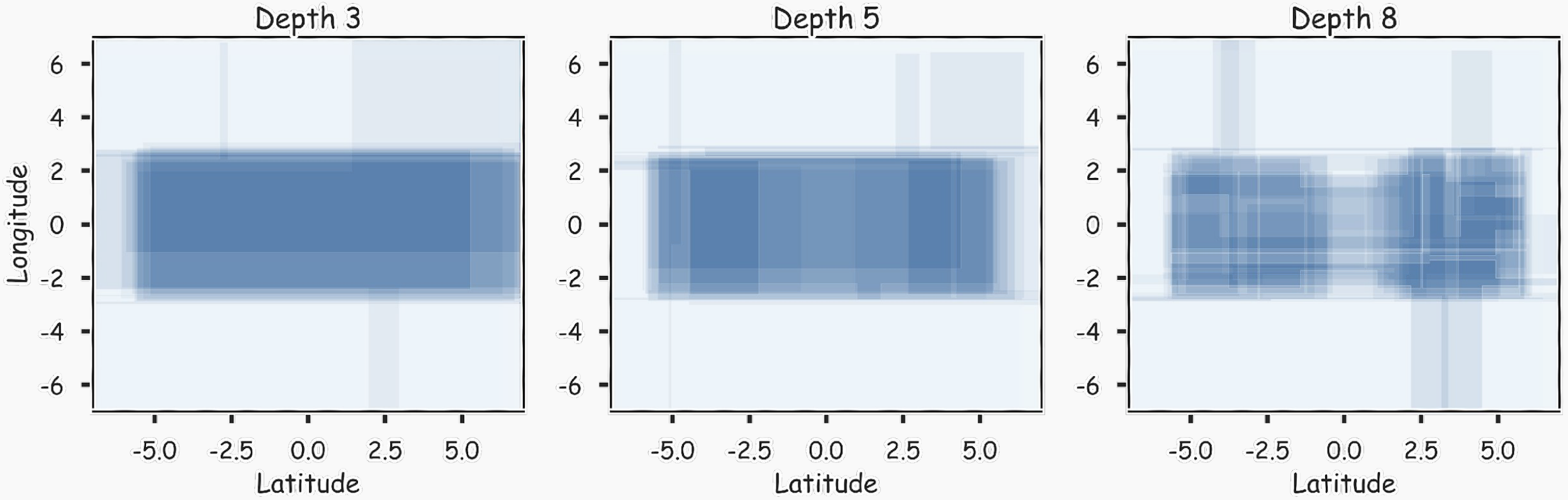
Question: Do you see any problems?

- If trees are too shallow it can still underfit.
- Still some overfitting if the trees are too large.
- Interpretability -

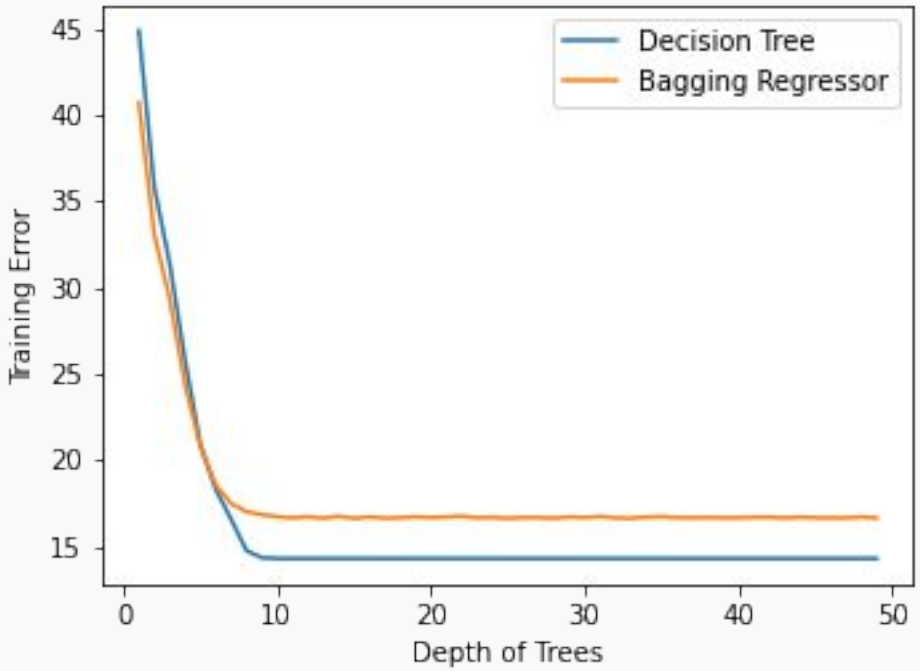
  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

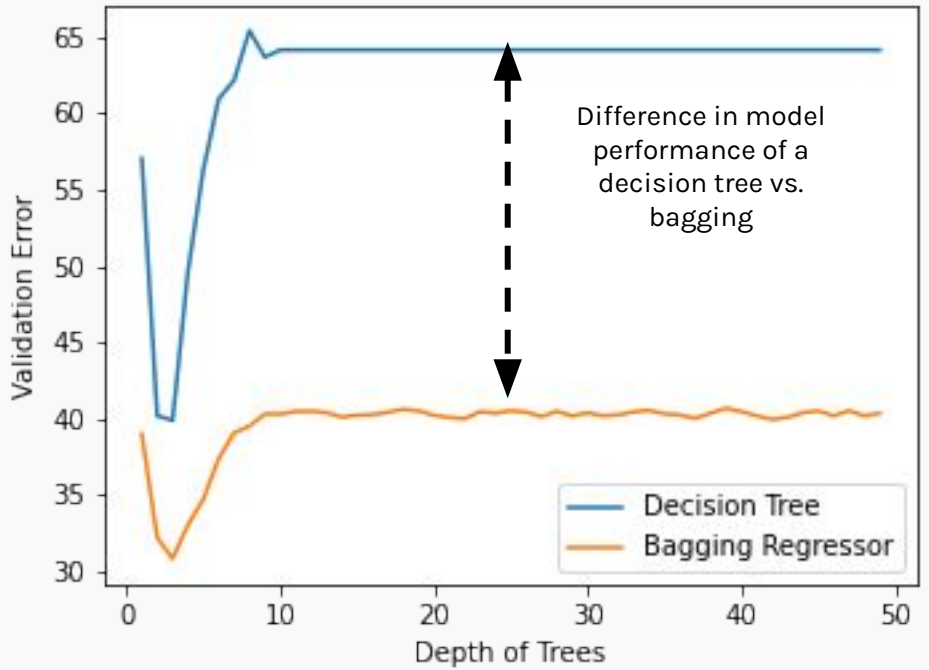
Underfitting & Overfitting

Here we fit 100 trees using bootstrapped samples. Even with multiple estimators, the shallow tree will not be able to capture the real pattern.



Underfitting and Overfitting in Decision Tree vs. Bagging

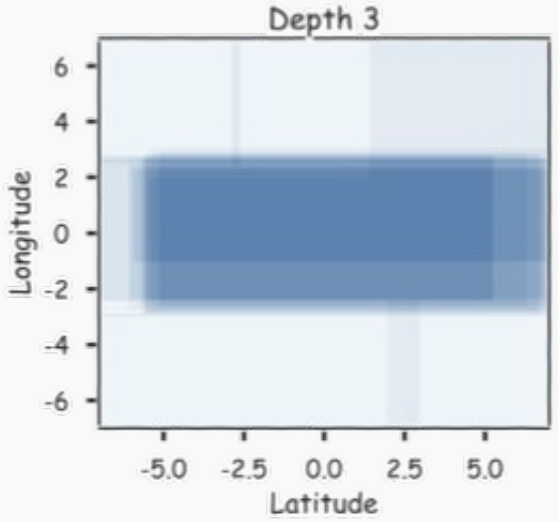




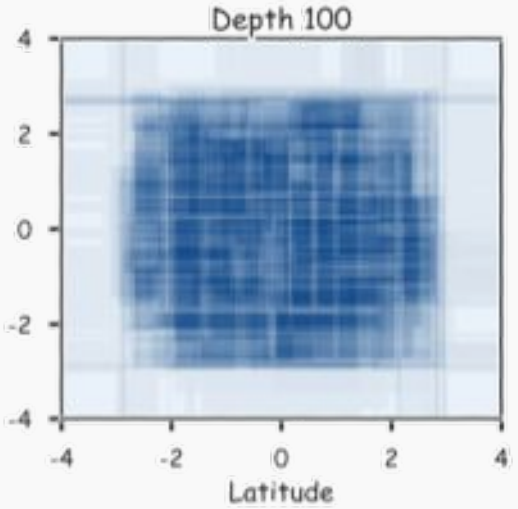
- The graph shows us that the **more depth we add to Decision Trees, the quickly the model overfits.**
- However, for Bagging, we see that there is still some form of **better model performance maintained after increase of depth of trees** although eventually it does **lead to no improvement of performance after a certain point as well.**

Cases of underfitting and overfitting in Bagging

Underfitting

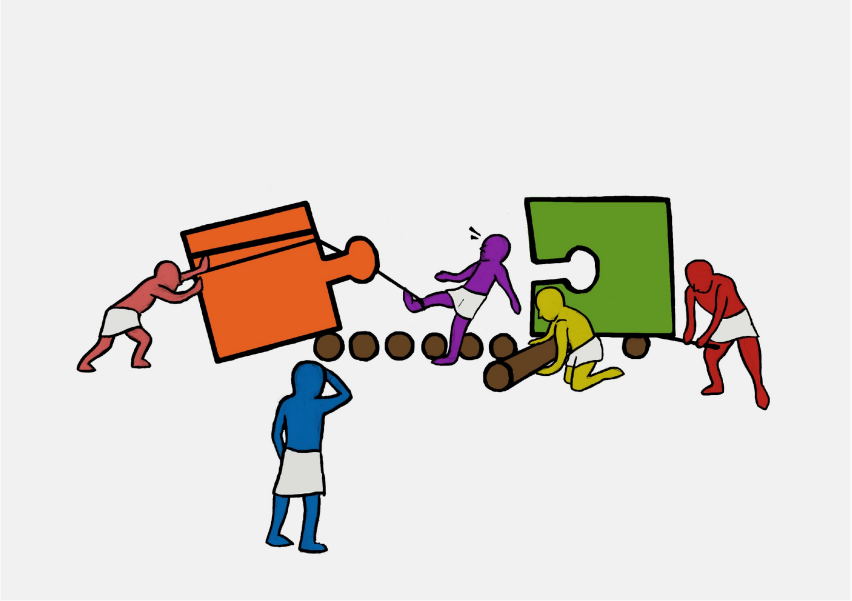


Overfitting



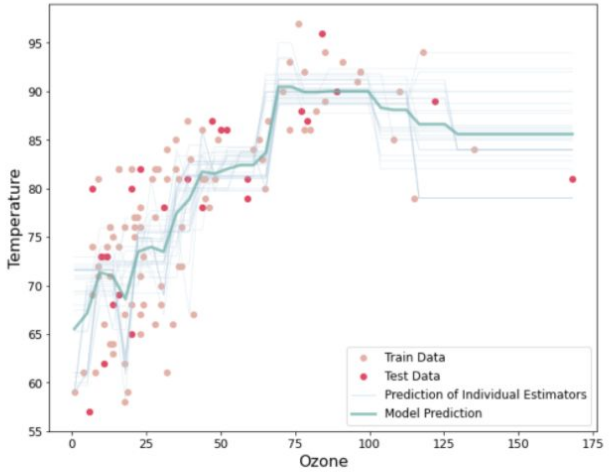
How to solve it?

Cross Validation



🏋️ Exercise: A.2 - Regression with Bagging

The aim of this exercise is to understand regression using Bagging.



Instructions:

- Read the dataset `airquality.csv` as a Pandas dataframe.
- Take a quick look at the dataset.
- Split the data into train and test sets.
- Specify the number of bootstraps as 30 and a maximum depth of 3.
- Define a Bagging Regression model that uses Decision Tree as its base estimator.
- Fit the model on the train data.
- Use the helper code to predict using the mean model and individual estimators. The plot will look similar to the one given above.
- Predict on the test data using the first estimator and the mean model.
- Compute and display the test MSEs.

Bagging
Part B – OOB Error and Variable Importance

Pavlos Protopapas

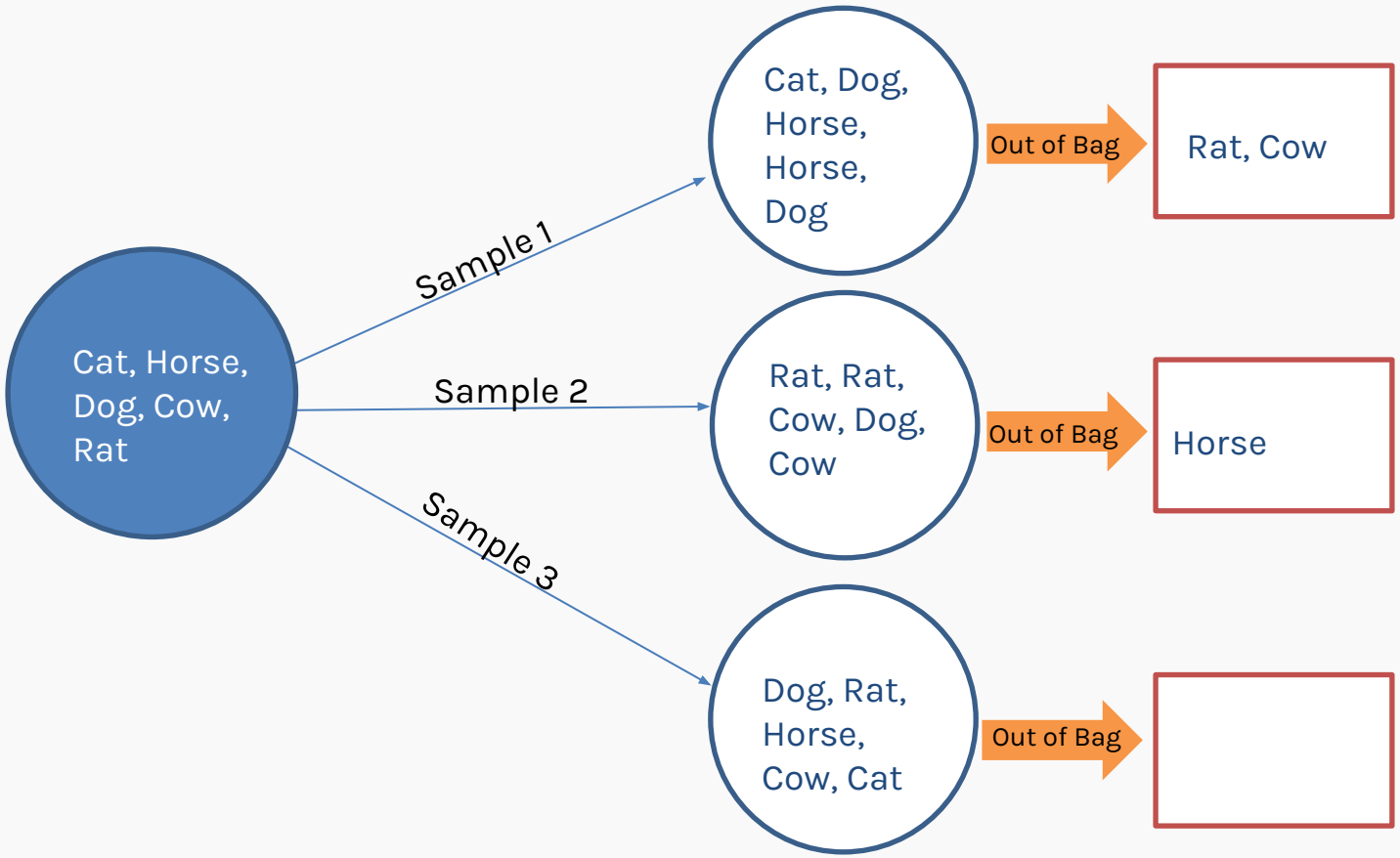
Outline

- Memory Bytes - Decision Trees
- Bagging
- Out of Bag Error (OOB)
- Variable Importance

Outline

- Memory Bytes - Decision Trees
- Bagging
- **Out of Bag Error (OOB)**
- Variable Importance

What is OOB?



Out-of-bag (OOB) error/Out-of-bag estimate is a method of determining the prediction error that allows the trees to be fit and validated whilst being trained.

Why?

- To measure generalizability.
- Replaces the need for a separate measurement of performance for a validation-set performance.

Out-of-Bag Error

With ensemble methods, we get a new metric for assessing the predictive performance of the model, the ***out-of-bag error***.

Given a training set and an ensemble of models, each trained on a bootstrap sample, we compute the ***out-of-bag error*** of the averaged model by

1. For each point x_i in the training set, we average the predicted output $\hat{y}_i$ for this point over the ***B*** trees whose bootstrap training set excludes this point.
2. We compute the error or squared error of this averaged prediction. Call this the **point-wise out-of-bag error.**
3. We average the point-wise out-of-bag error over the full training set ***N***.

Out of Bag Error (OOB)

Original Data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
.	.
.	.
X_n	y_n

Bootstrap Sample 1

X	Y
X_4	y_4
X_9	y_9
X_{11}	y_{11}
X_{21}	y_{21}
X_{35}	y_{35}
.	.
.	.
.	.
X_k	y_k

Decision Tree 1

Used and unused data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
.	.
.	.
X_n	y_n

Out of Bag Error (OOB)

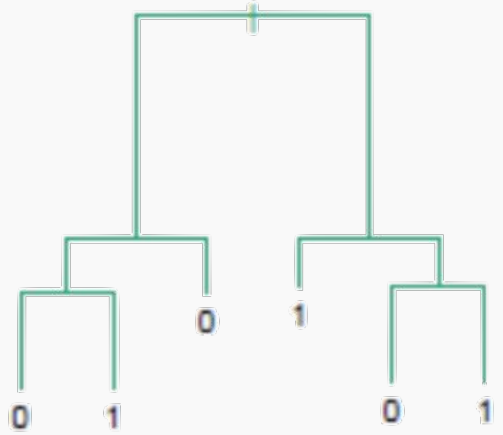
Original Data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
.	.
.	.
X_n	y_n

Bootstrap Sample 2

X	Y
X_5	y_5
X_7	y_7
X_{13}	y_{13}
X_{27}	y_{27}
X_{32}	y_{32}
.	.
.	.
.	.
X_k	y_k

Decision Tree 2



Used and unused data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
.	.
.	.
X_n	y_n

Out of Bag Error (OOB)

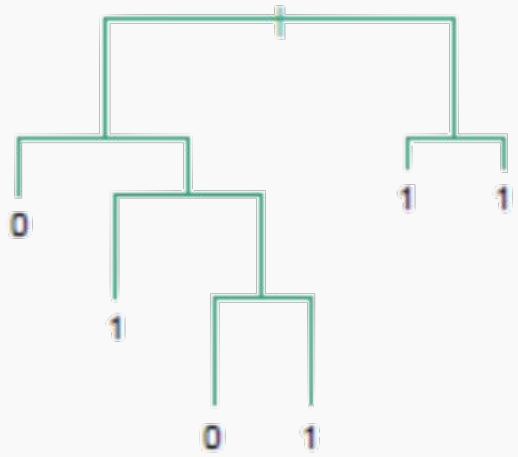
Original Data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
.	.
.	.
X_n	y_n

Bootstrap Sample 3

X	Y
X_9	y_9
X_{38}	y_{38}
X_{11}	y_{11}
X_{45}	y_{45}
X_{34}	y_{34}
.	.
.	.
.	.
X_k	y_k

Decision Tree 3



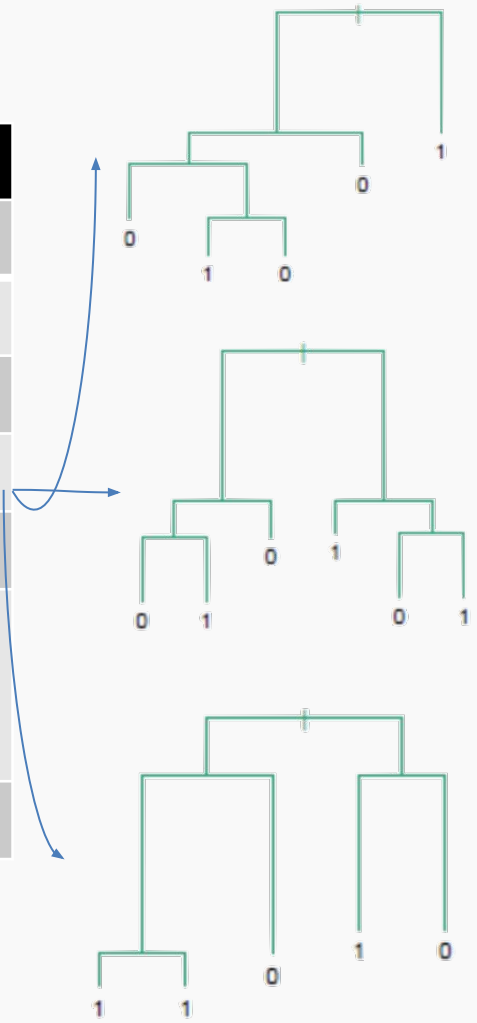
Used and unused data

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
.	.
.	.
X_n	y_n

Point-wise out-of-bag error

B Trees that did not see $\{X_i, y_i\}$

X	Y
X_1	y_1
X_2	y_2
X_3	y_3
X_4	y_4
X_5	y_5
.	.
X_i	y_i
.	.
X_n	y_n



Classification

Regression

Majority over the B trees
B trees are the tree that
have not seen I point

$$\hat{y}_{i,pw} = majority(\hat{y}_i) \qquad e_i = \mathbb{I}(\hat{y}_{i,pw} \neq y_i)$$

$$\hat{y}_{i,pw} = \frac{1}{B}\sum_{j \in B} \hat{y}_{i,j} \qquad e_i = (y_i - \hat{y}_{i,pw})^2$$

- B – No. of trees that the model hasn't seen
- pw – point wise error (the error for point-i from B trees)
- $\hat{y}_{i,pw}$ – output of the tree that the model hasn't seen

OOB Error

We average the point-wise out-of-bag error over the full training set.

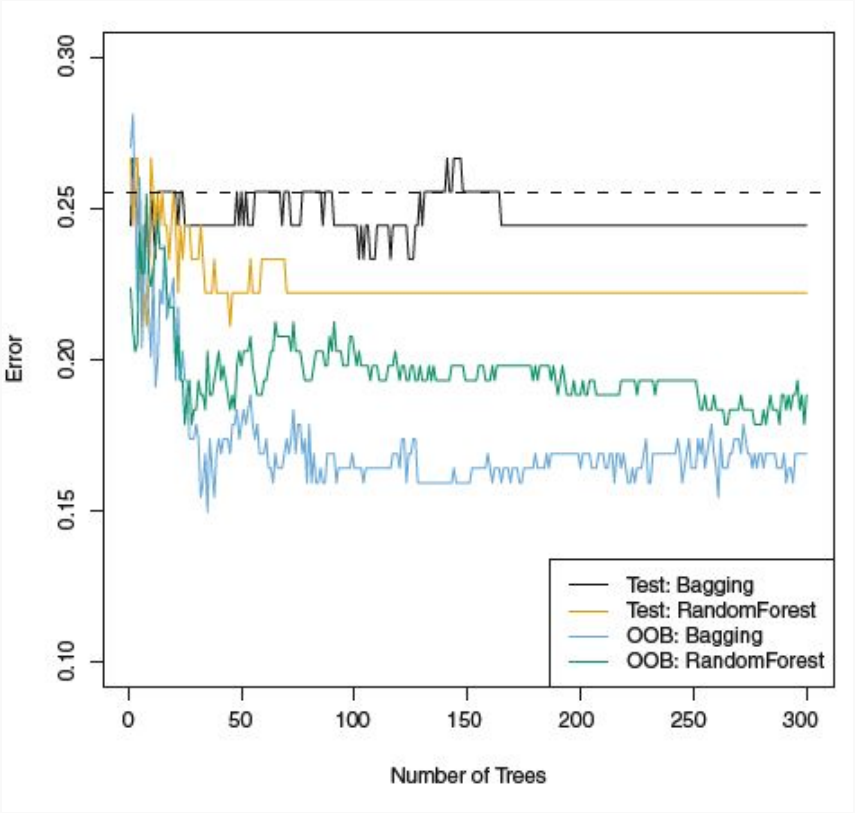
Classification

$$Error_{OOB} = \frac{1}{N}\sum_i^N e_i = \frac{1}{N}\sum_i^N \left(\hat{y}_{i,pw} \neq y_i\right)$$

Regression

$$Error_{OOB} = \frac{1}{N}\sum_i^N e_i = \frac{1}{N}\sum_i^N \left(y_i - \hat{y}_{i,pw}\right)^2$$

Why OOB Error?



Source: An Introduction to Statistical Learning with Applications in R, Springer

- While using the cross-validation technique, every validation set has already been seen or used in training by a few decision trees and hence there is a leakage of data, therefore more variance.
- But, OOB Error prevents leakage and gives a better model with low variance, so we use OOB error.
- There is also lesser computational cost for OOB error as compared to CV for bagging.

Bagging

Question: Do you see any problems?

- If trees are too shallow it can still underfit.
- Still some overfitting if the trees are too large.
- Interpretability:

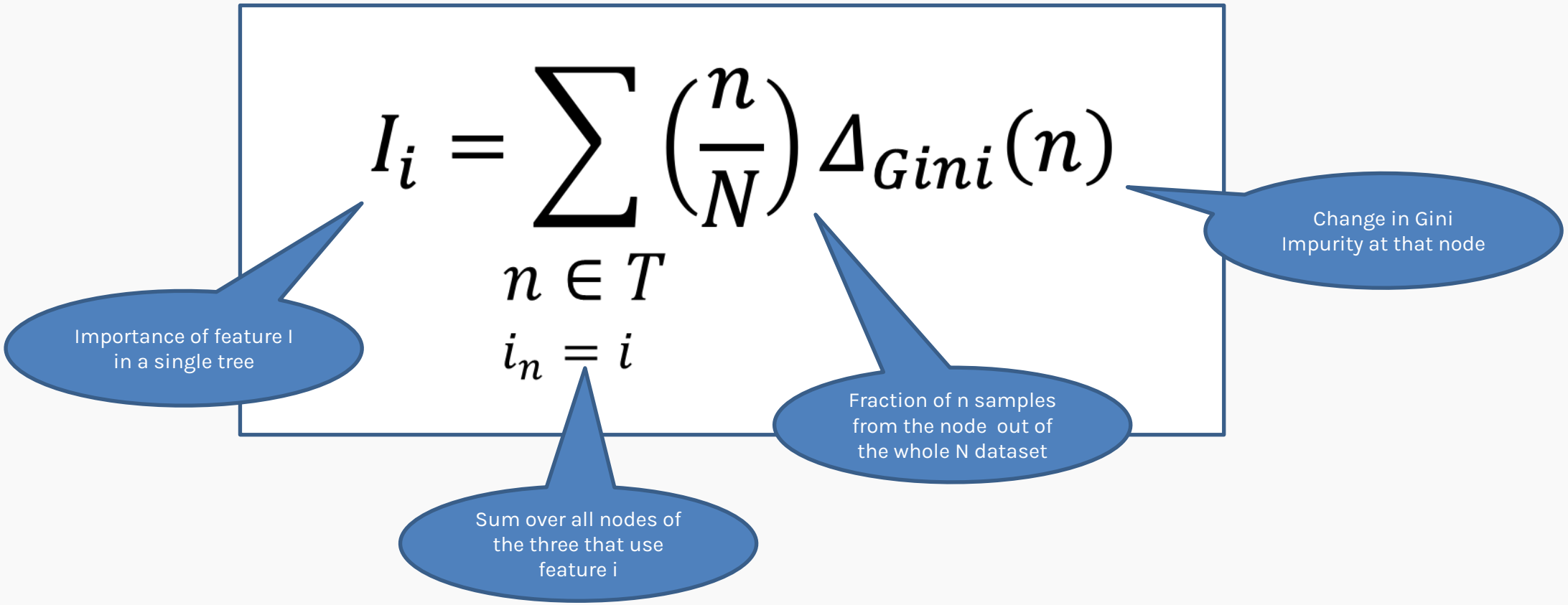
  The **major drawback** of bagging (and other ***ensemble methods*** that we will study) is that the averaged model is no longer easily interpretable - i.e. one can no longer trace the 'logic' of an output through a series of decisions based on predictor values!

Outline

- Review of Decision Trees
- Bagging
- Out of Bag Error (OOB)
- **Variable Importance**

Variable Importance for Bagging

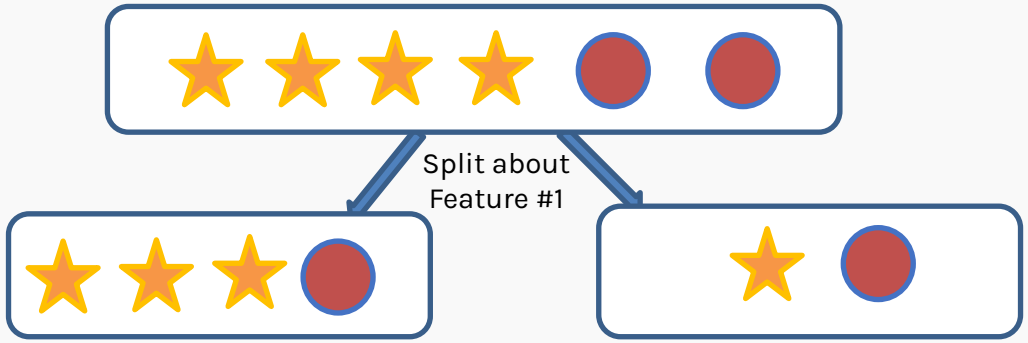
Decision trees make splits that maximize the decrease in impurity. By calculating the feature importance of the variables/features over singular trees and then averaging across all trees, we arrive at the variable importance in a Bagging Model.



Variable Importance for Bagging

Decision trees make splits that maximize the decrease in impurity. By calculating the feature importance of the variables/features over singular trees and then averaging across all trees, we arrive at the variable importance in a Bagging Model.

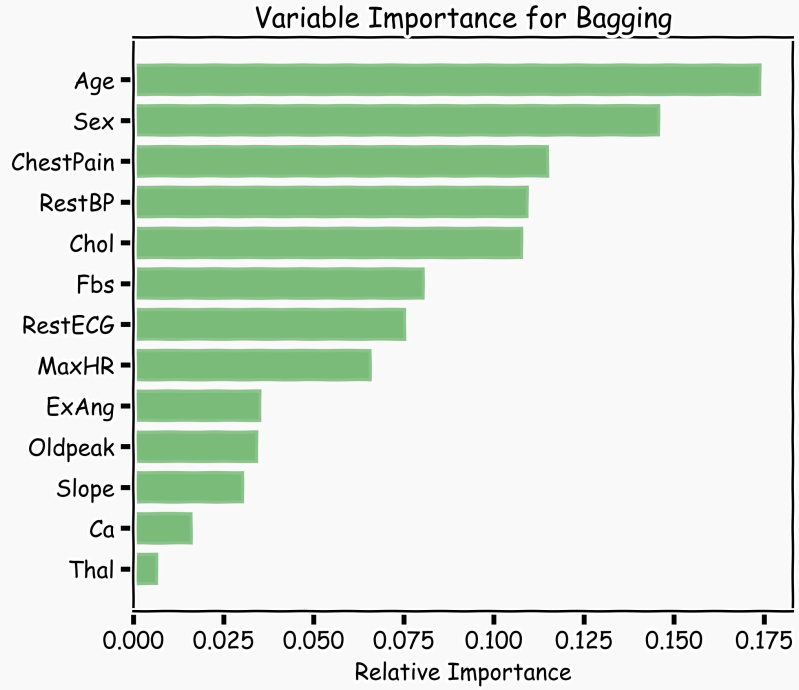
Example:



Impurity Decrease for Feature #1 = Fraction of samples in the Node * (G1 – G2)

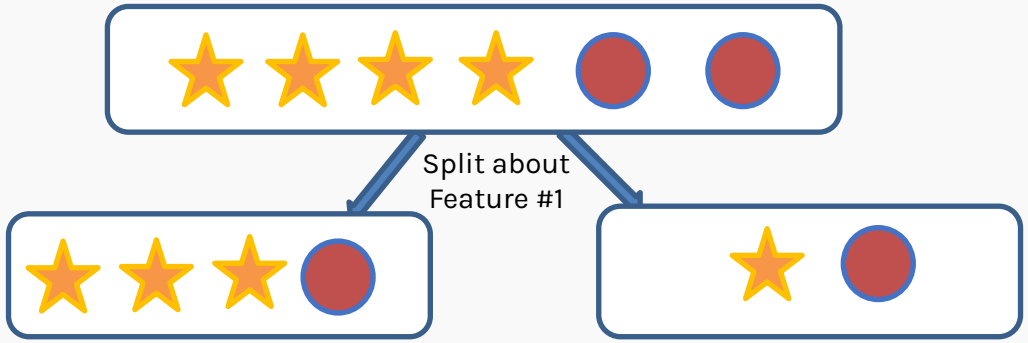
Probability of using that node for a sample data point

Average this over all of the trees



Variable Importance for Bagging

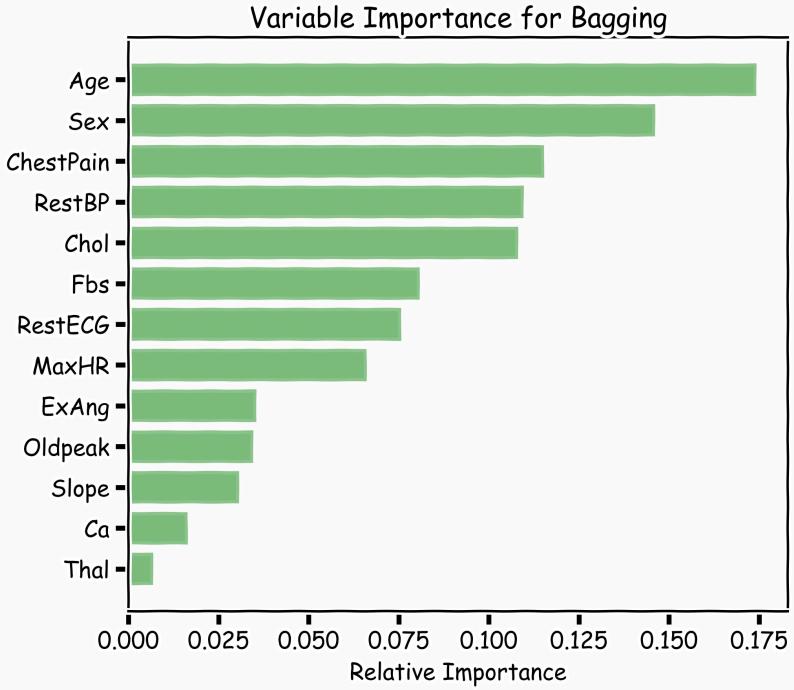
Decision trees make splits that maximize the decrease in impurity. By calculating the mean decrease in impurity for each feature across all trees, we arrive at the variable importance.



Impurity Decrease for Feature #1 = Fraction of samples in the Node * (G1 – G2)

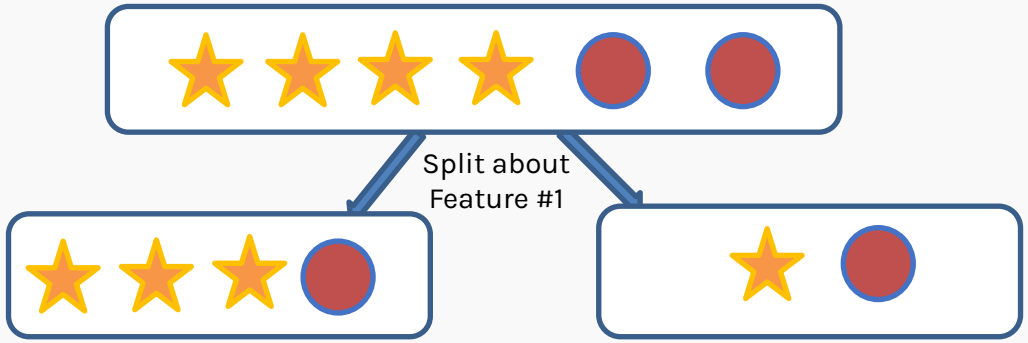
G1 is the Gini index of the Parent

Average this over all of the trees



Variable Importance for Bagging

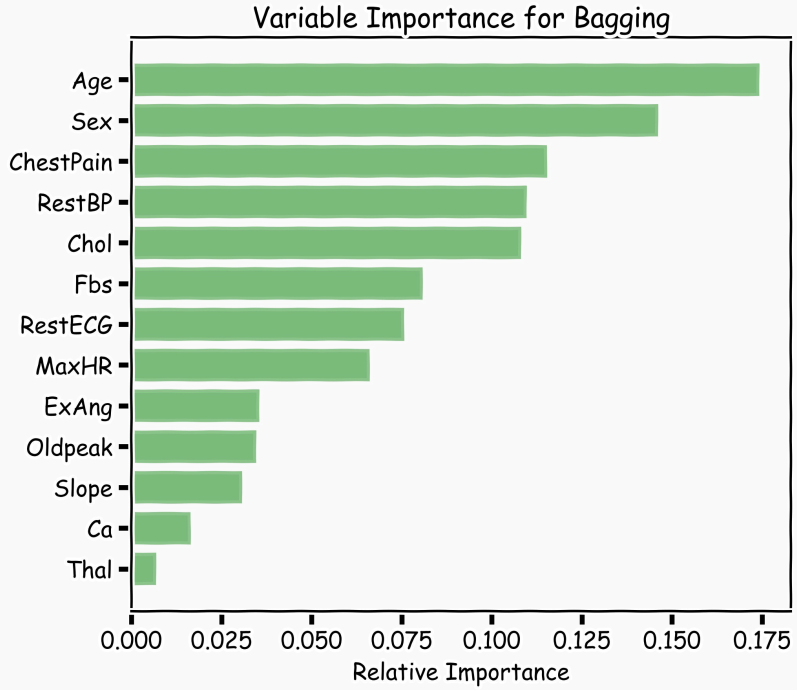
Decision trees make splits that maximize the decrease in impurity. By calculating the mean decrease in impurity for each feature across all trees, we arrive at the variable importance.



Impurity Decrease for Feature #1 = Fraction of samples in the Node * (G1 – G2)

G2 is the weighted average of the Gini index of the children

Average this over all of the trees



! We will see a better way to calculate feature importance through permuting over the features rather than averaging across trees.

Improving on Bagging

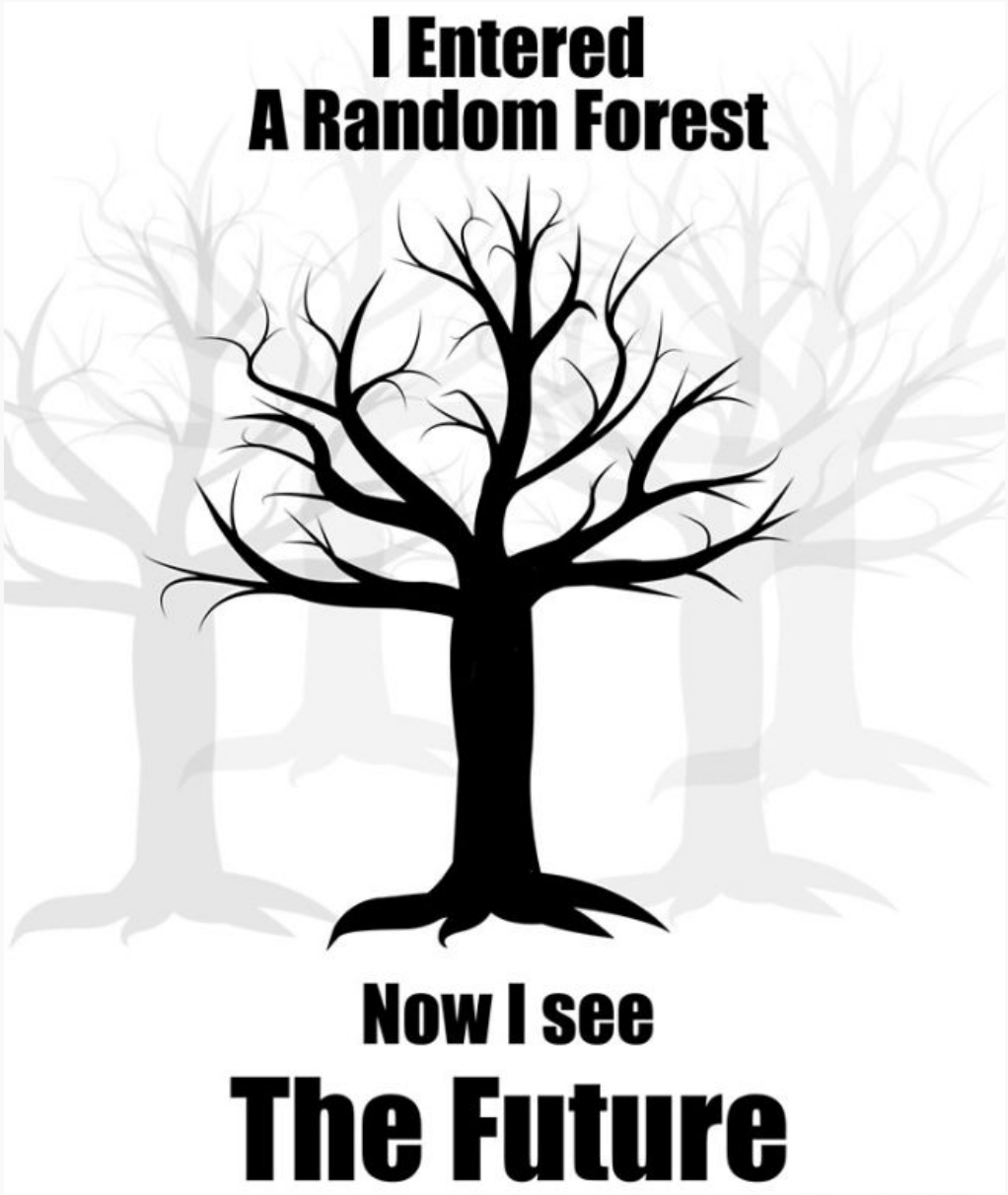
- In practice, the ensembles of trees in Bagging tend to be **highly correlated.**
- Suppose we have an **extremely strong predictor**, x_j, in the training set amongst **moderate predictors**. Then the greedy learning algorithm ensures that most of the models in the ensemble will choose to split on x_j in early iterations.
- However, we assumed that each tree in the ensemble is **independently and identically distributed**, with the expected output of the averaged model the same as the expected output of any one of the trees.

… Cliffhanger!

Random Forest

Part A – Random Forest and Variable Importance

Pavlos Protopapas


I Entered
A Random Forest
Now I see
The Future

Outline

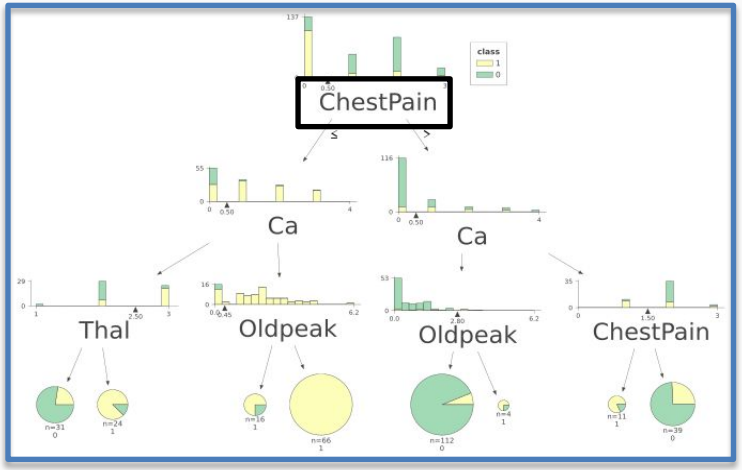
- Motivation
- Random Forests
- Variable Importance for RF
- Missing data
- Imbalanced data
- Tree building algorithms

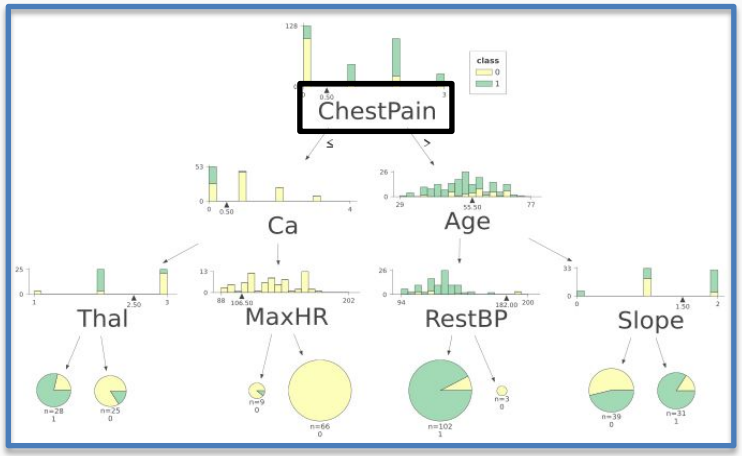
Outline

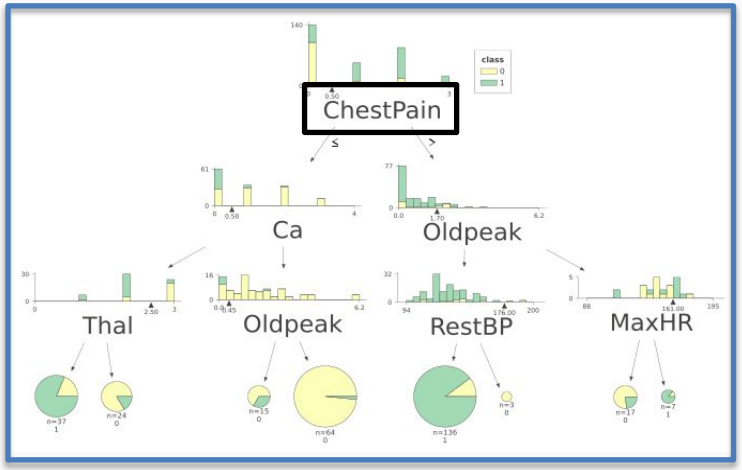
- **Motivation**
- Random Forests
- Variable Importance for RF
- Missing data
- Imbalanced data
- Tree building algorithms

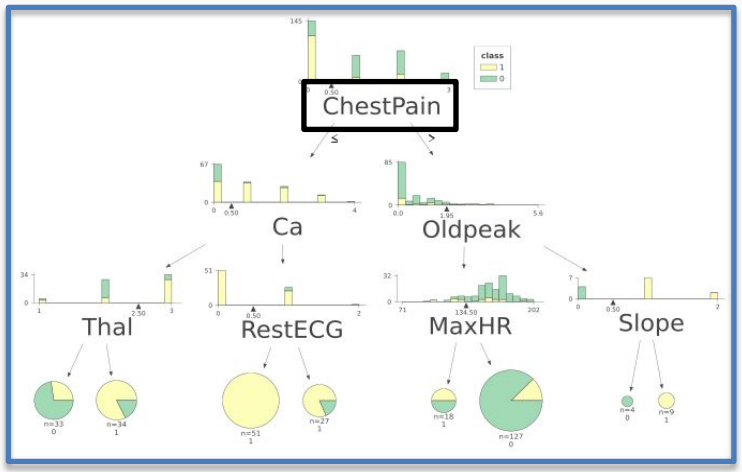
Motivation

Consider the following decision trees in a bagging model that predicts if a person has heart disease:



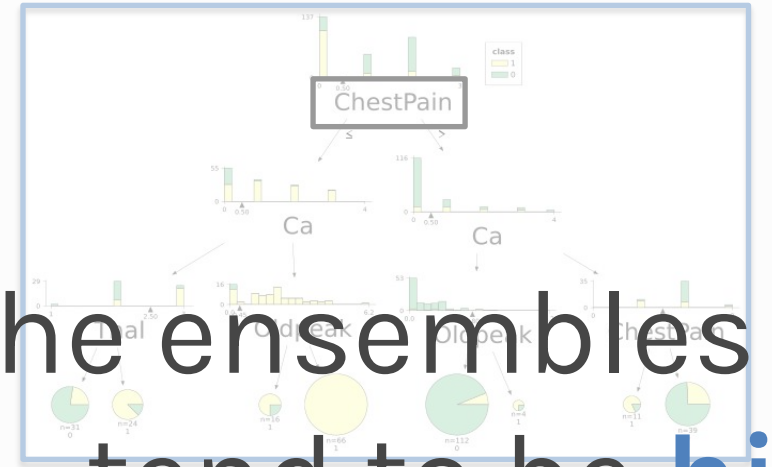


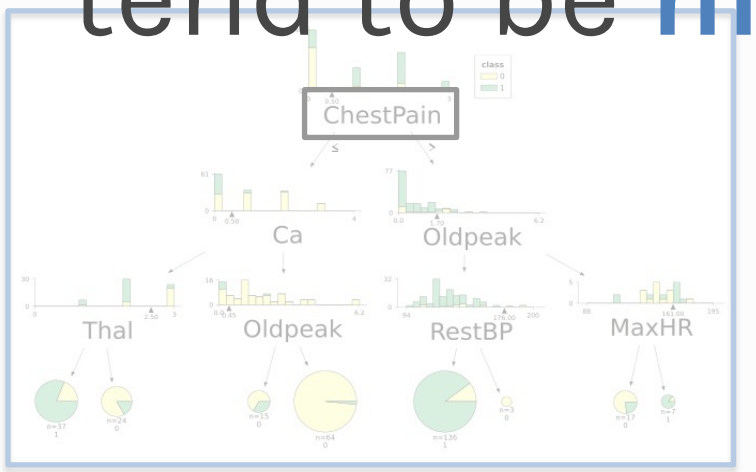


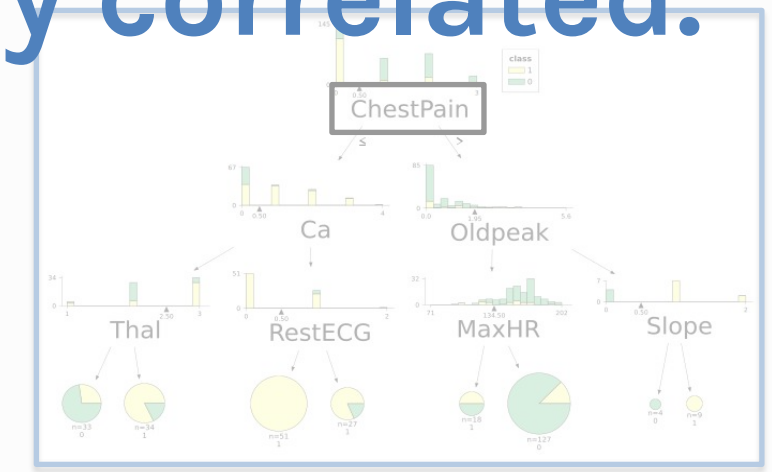


Motivation

Consider the following decision trees in a bagging model that predicts if a person has heart disease:







The ensembles of trees in bagging tend to be **highly correlated.**

Outline

- Motivation
- **Random Forests**
- Variable Importance for RF
- Missing data
- Imbalanced data
- Tree building algorithms

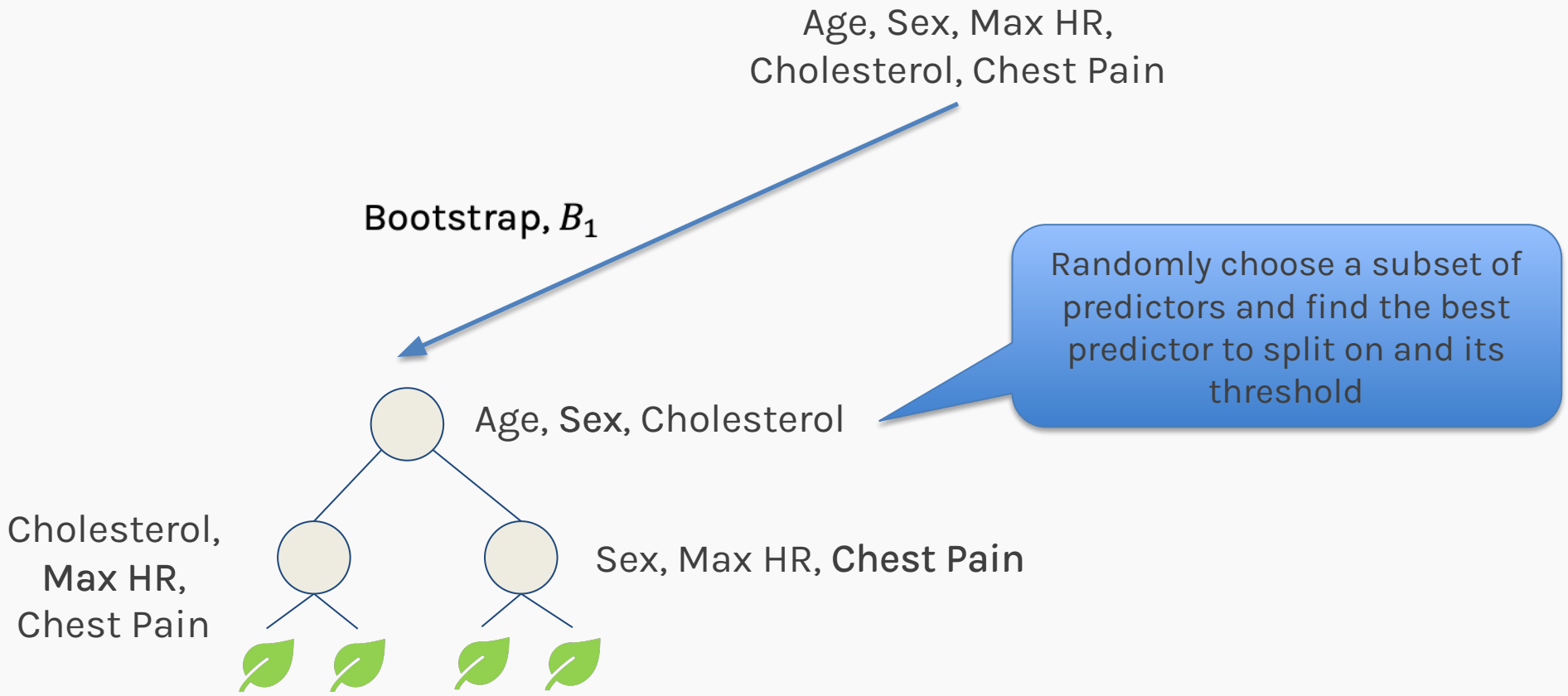
Random Forests

Random Forest is a modified form of bagging that creates ensembles of **independent** decision trees.

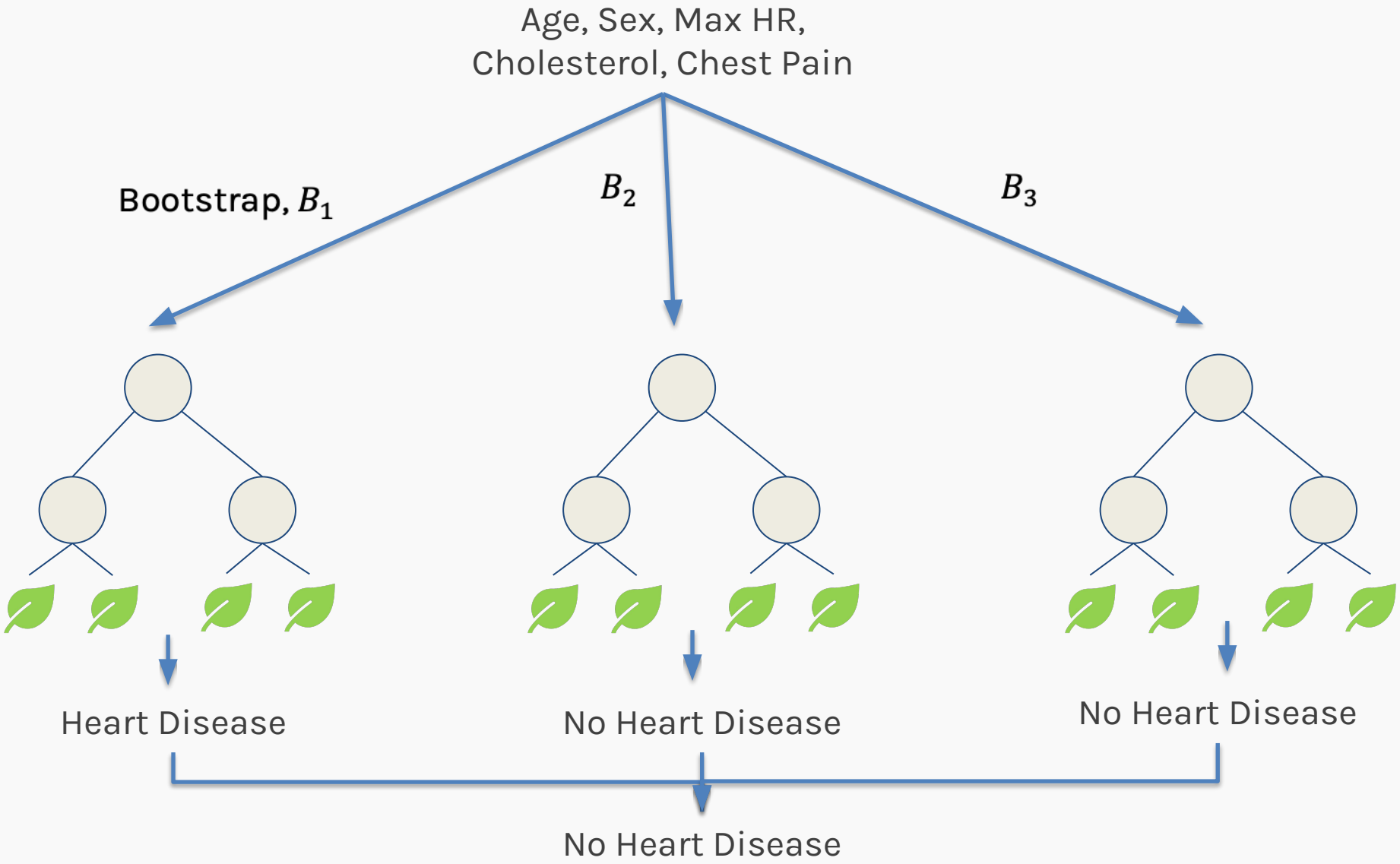
Question: How do we de-correlate the trees?

Random Forests

Consider a dataset that contains the following predictors:



Random Forests



Random Forests

In summary:

1. Train each tree on a separate **bootstrap sample** of the full training set with J predictors (same as in bagging).

2. For each tree, at each split, we **randomly** select a sub-set of J' predictors from the full set of predictors.

3. From amongst the J' **predictors**, we select the optimal predictor and the optimal corresponding threshold for the split.

Tuning Random Forests

Random forest models have multiple **hyper-parameters** to tune:

1. The **number of predictors** to randomly select at each split.
2. The **total number of trees** in the ensemble.
3. The **stopping criteria** - maximum depth, minimum leaf node size, etc.

Tuning Random Forests

There are standard (default) values for each of random forest hyper-parameters recommended by long time practitioners, but generally these parameters should be tuned through **OOB** (making them data and problem dependent).

e.g. Number of predictors to randomly select at each split:

- $\sqrt{N_j}$ for classification
- $\frac{N}{3}$ for regression

For the number of trees, we use out-of-bag errors. With OOB, training and validation can be done in a single sequence - we cease training once the out-of-bag error stabilizes

Outline

- Motivation
- Random Forests
- **Variable Importance for RF**
- Missing data
- Imbalanced data
- Tree building algorithms

Variable Importance for RF

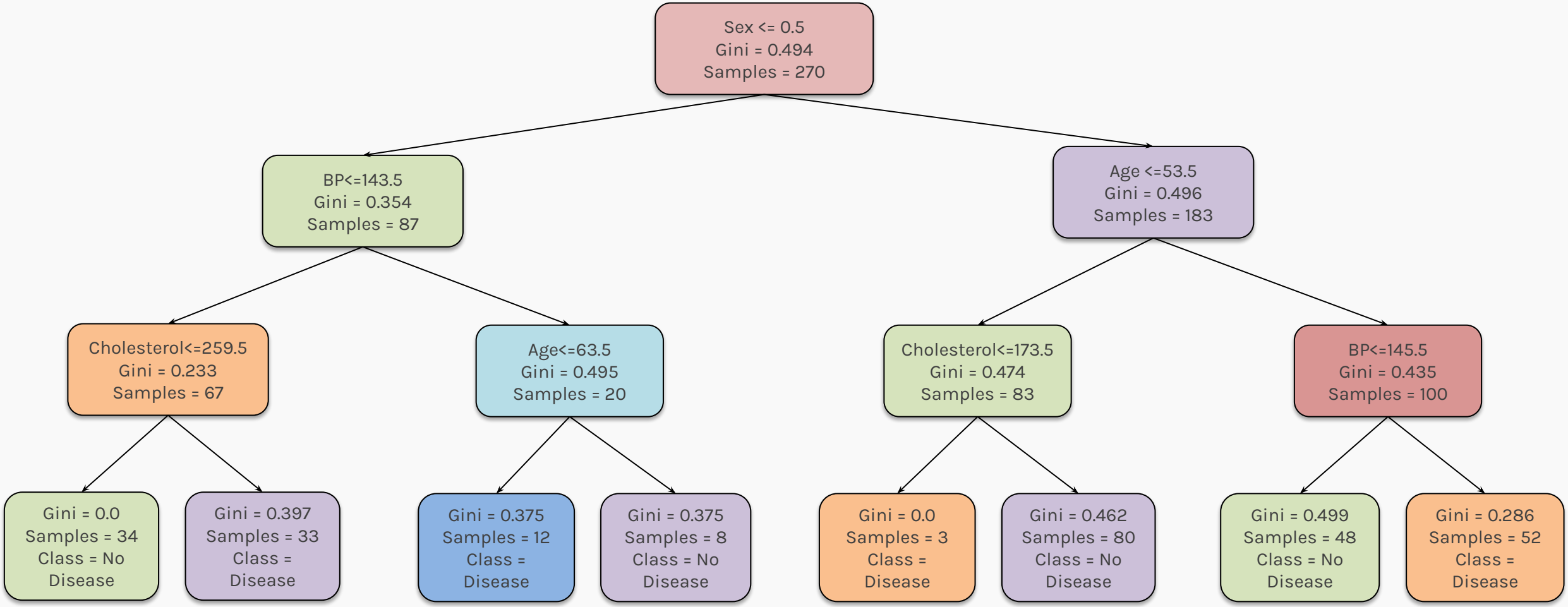
1. Mean Decrease in Impurity.

2. Permutation importance.

3. One step further (SHAP values, LIME) - We will see these methods in later lectures.

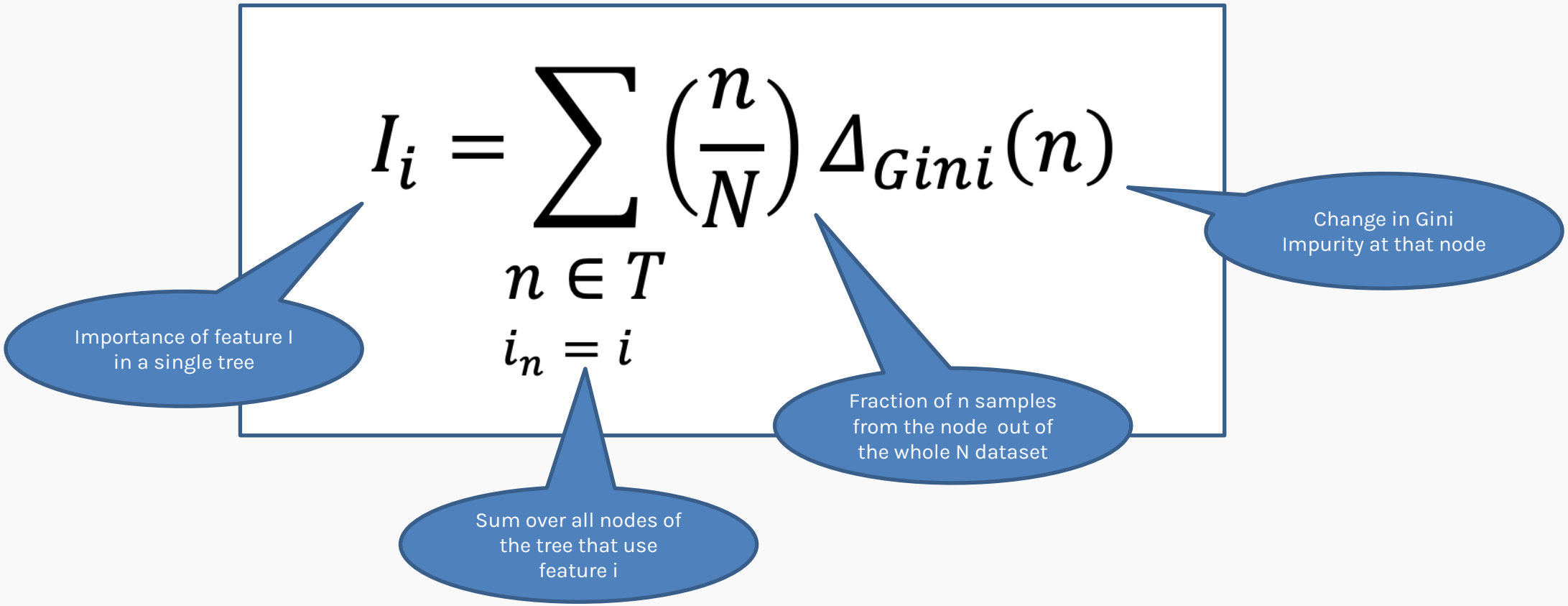
Recap: Mean Decrease in Impurity (MDI)

Consider the following decision tree:



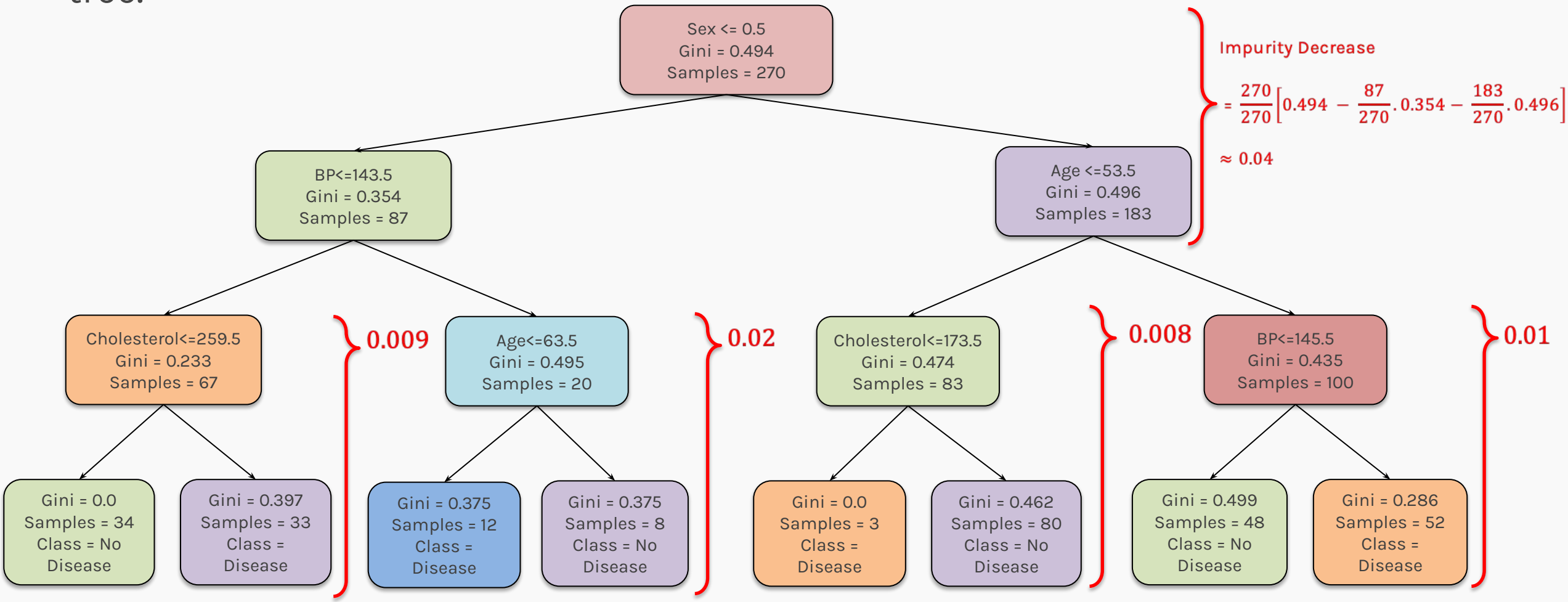
Recap: Mean Decrease in Impurity (MDI)

Step 1: Calculate the mean decrease in impurity for each node in the decision tree.



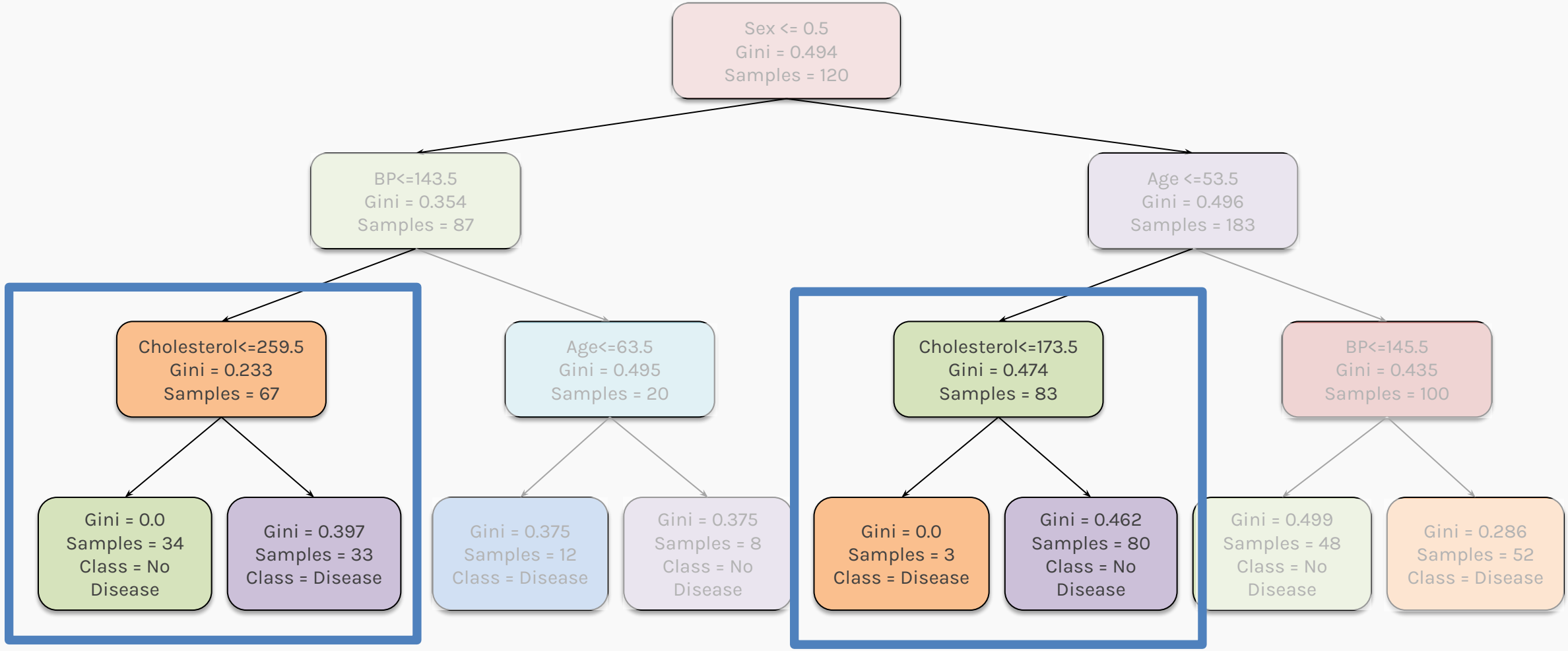
Recap: Mean Decrease in Impurity (MDI)

Step 1: Calculate the mean decrease in impurity for each node in the decision tree.



Recap: Mean Decrease in Impurity (MDI)

Let us try to compute the importance of the feature **cholesterol**:



Recap: Mean Decrease in Impurity (MDI)

Step 2: Calculate the importance of the feature by summing up the impurity decrease calculated in step 1 for each node in which that feature occurs.

$$\text{Feature Importance}_{cholesterol} = \sum_{n \in \text{nodes split on cholesterol}} Impurity\ Decrease_n$$

$$= 0.009 + 0.008 = 0.19$$

Recap: Mean Decrease in Impurity (MDI)

Step 3: Normalize this value between 0 and 1 by dividing by the sum of all feature importance values.

This ensures the sum of all feature importances in a decision tree adds up to 1.

$$\text{Norm Feature Importance}_{cholesterol} = \frac{\text{Feature Importance}_{cholesterol}}{\sum_{j \in all\ features} \text{Feature Importance}_j}$$

Recap: Mean Decrease in Impurity (MDI)

Step 4: At the Random Forest level, average over all the trees.

$$\text{Norm RF Feature Importance}_{cholesterol} = \frac{\sum_{t \in all\ trees} \text{Norm Feature Importance}}{Total\ number\ of\ trees}$$

Summary: Mean Decrease in Impurity (MDI)

For each feature j in the dataset:

Step 1: Calculate the mean decrease in impurity for each node in the decision tree.

Step 2: Calculate the importance of the feature by summing up the impurity decrease calculated in step 1 for each node in which that feature occurs.

$$\text{Feature Importance}_j = \sum_{n \in \text{nodes split on } j} Impurity\ Decrease_n$$

Step 3: Normalize this value between 0 and 1 by dividing by the sum of all feature importance values.

$$\text{Norm Feature Importance}_j = \frac{\text{Feature Importance}_j}{\sum_{i \in all\ features} \text{Feature Importance}_i}$$

Step 4: At the Random Forest level, average over all the trees.

$$\text{Norm RF Feature Importance}_j = \frac{\sum_{t \in all\ trees} \text{Norm Feature Importance}_j}{Total\ number\ of\ trees}$$

Permutation Importance

Consider the following dataset:

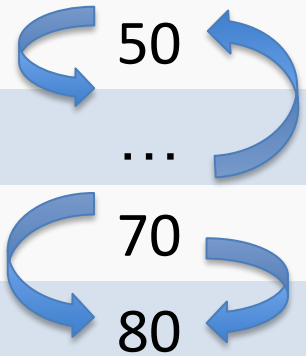
Height (cm)	Weight (kg)	…	Fitness Level (1 – 5)
150	65	…	2
140	50	…	3
…	…	…	…
170	70	…	4
160	80	…	1

Step 1: Record the validation/OOB accuracy of RF model: **Accuracy = 88.6**

Permutation Importance

Step 1: Record the validation/OOB accuracy of RF model: **Accuracy = 88.6**

Height (cm)	Weight (kg)	…	Fitness Level (1 – 5)
150	65	…	2
140	50	…	3
…	…	…	…
170	70	…	4
160	80	…	1



Step 2: Randomly permute the data for column j in the validation/OOB set.

Permutation Importance

Step 1: Record the validation/OOB accuracy of RF model: **Accuracy = 88.6**

Height (cm)	Weight (kg)	…	Fitness Level (1 – 5)
150	70	…	2
140	65	…	3
…	…	…	…
170	80	…	4
160	50	…	1

Step 2: Randomly permute the data for column j in the validation/OOB set.

Permutation Importance

Step 1: Record the validation/OOB accuracy of RF model: **Accuracy = 88.6**

Height (cm)	Weight (kg)	…	Fitness Level (1 – 5)
150	70	…	2
140	65	…	3
…	…	…	…
170	80	…	4
160	50	…	1

Step 2: Randomly permute the data for column *j* in the validation/OOB set. Record the validation/OOB accuracy of the RF model on the modified dataset: **Permuted Accuracy = 87.3**

Permutation Importance

Step 1: Record the validation/OOB accuracy of RF model: **Accuracy = 88.6**

Step 2: Randomly permute the data for column j in the validation/OOB set. Record the validation/OOB accuracy of the RF model on the modified dataset: **Permuted Accuracy = 87.3**

Step 3: Repeat the previous step K number of times and average all the accuracies.

Assume we permute the feature 3 times, we get:

Permuted Accuracy$_1$ = 87.3

Permuted Accuracy$_2$ = 86.6 → **Avg Permuted Accuracy = $\frac{87.3+86.6+86.1}{3}$ = 86.6**

Permuted Accuracy$_3$ = 86.1

Permutation Importance

Step 1: Record the validation/OOB accuracy of RF model: **Accuracy = 88.6**

Step 2: Randomly permute the data for column j in the validation/OOB set. Record the validation/OOB accuracy of the RF model on the modified dataset: **Permuted Accuracy = 87.3**

Step 3: Repeat the previous step K number of times and average all the accuracies: **Avg Permuted Accuracy = 86.6**

Step 4: Calculate the difference between average permuted and unpermuted accuracy to get the importance of the feature in the random forest: **Difference = 88.6 - 86.6 = 2**

Summary: Permutation Importance

For each feature j in the dataset:

Step 1: Record the validation/OOB accuracy of RF model: s .

For each repetition k in 1 …. K:

Step 2: Randomly permute the data for column j in the validation/OOB set. Record the validation/OOB accuracy $s_{k,j}$ of the RF model on the modified dataset.

Step 3: Compute the average of all the permuted datasets accuracies.

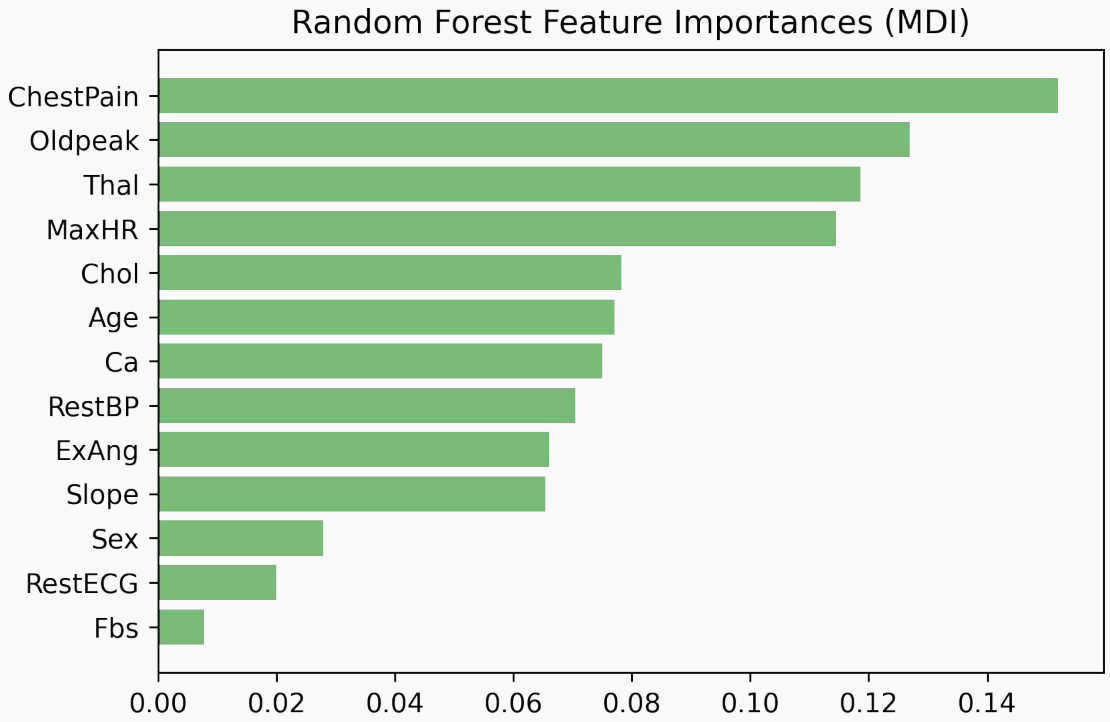
$$s_j = \frac{1}{K}\sum_{k=1}^{K} s_{k,j}$$

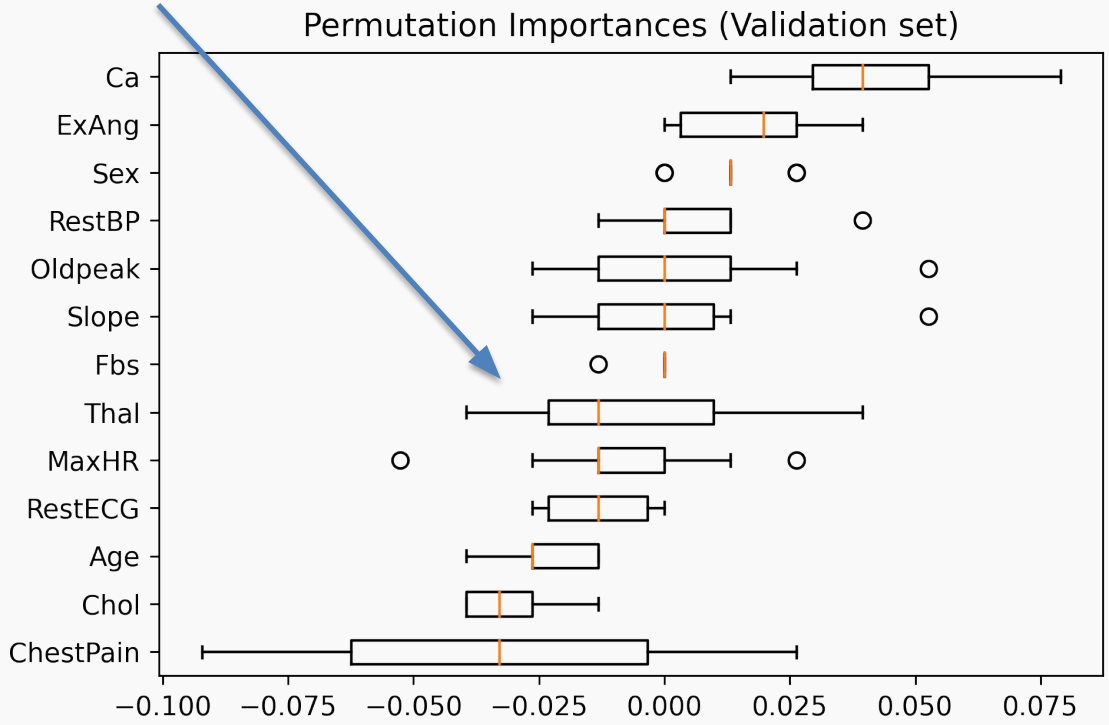
Step 4: Calculate the difference between average permuted and unpermuted accuracy to get the importance of the feature in the random forest.

$$\text{RF Feature Importance}_j = s - s_j$$

MDI vs Permutation Importance

The experiment is repeated k times.

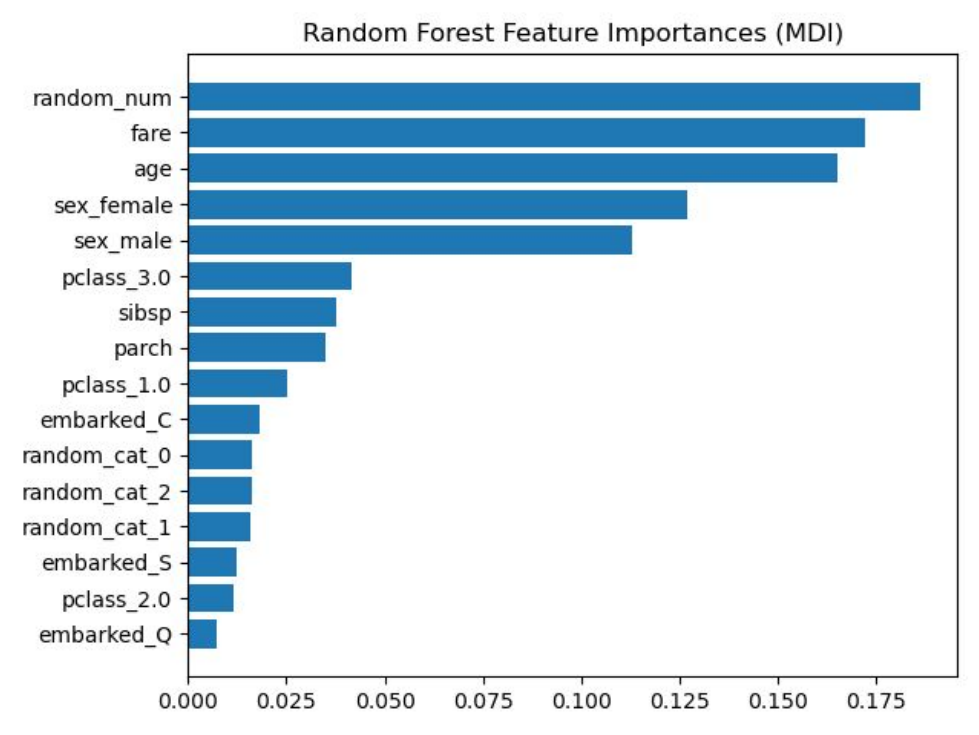




Features like *ChestPain*, *OldPeak*, *Thal* are ranked most important in MDI importance plot but they are ranked low in permutation importance plot.

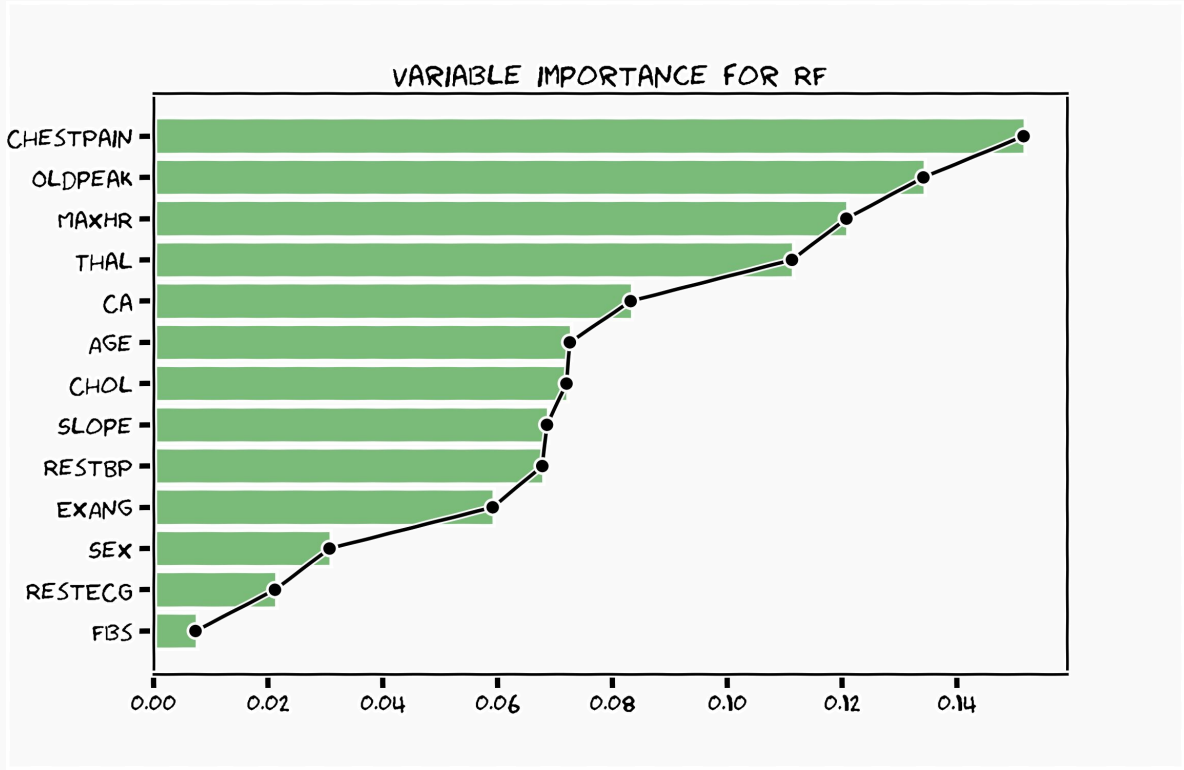
MDI vs Permutation Importance

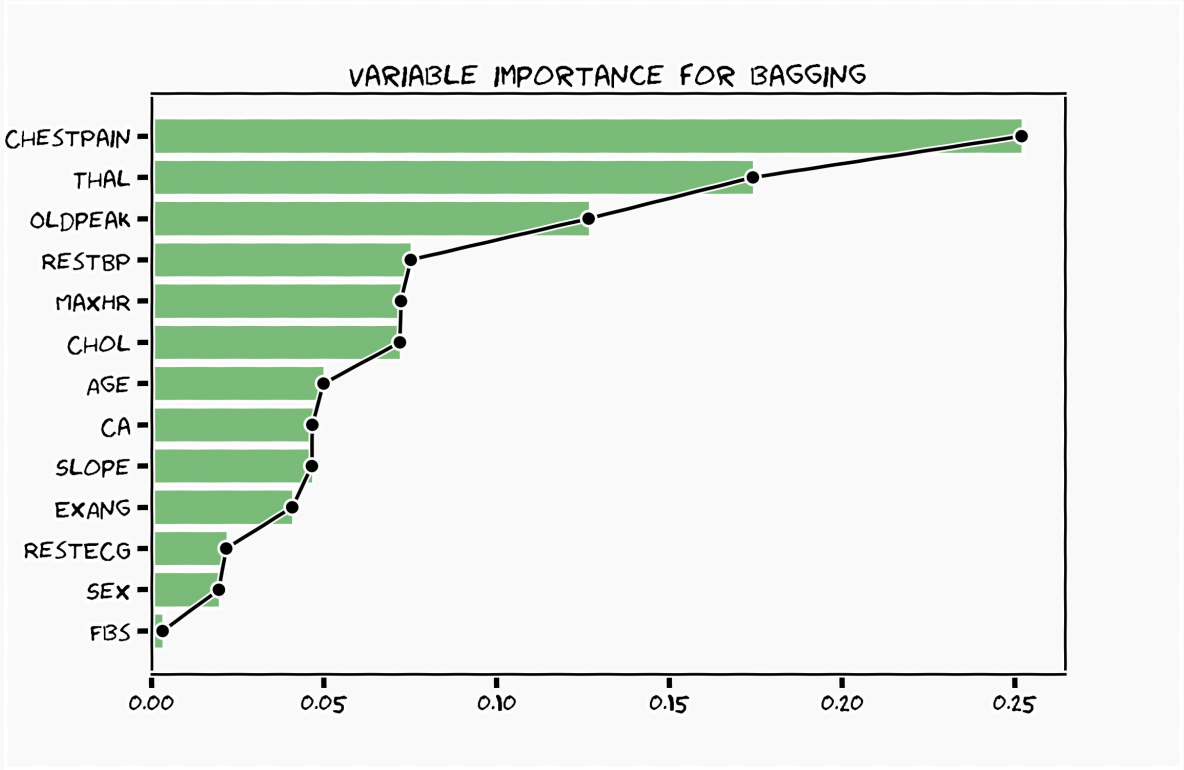
- The biggest advantage of the MDI is a speed of computation. All needed values are computed during the Random Forest training.
- The drawbacks of the method is its tendency to prefer (select as important) numerical features and categorical features with high cardinality.



Variable Importance for bagging vs RF

Variable importance for RF is smoother than that for bagging due to randomness introduced by selecting a subset of predictors to choose from.



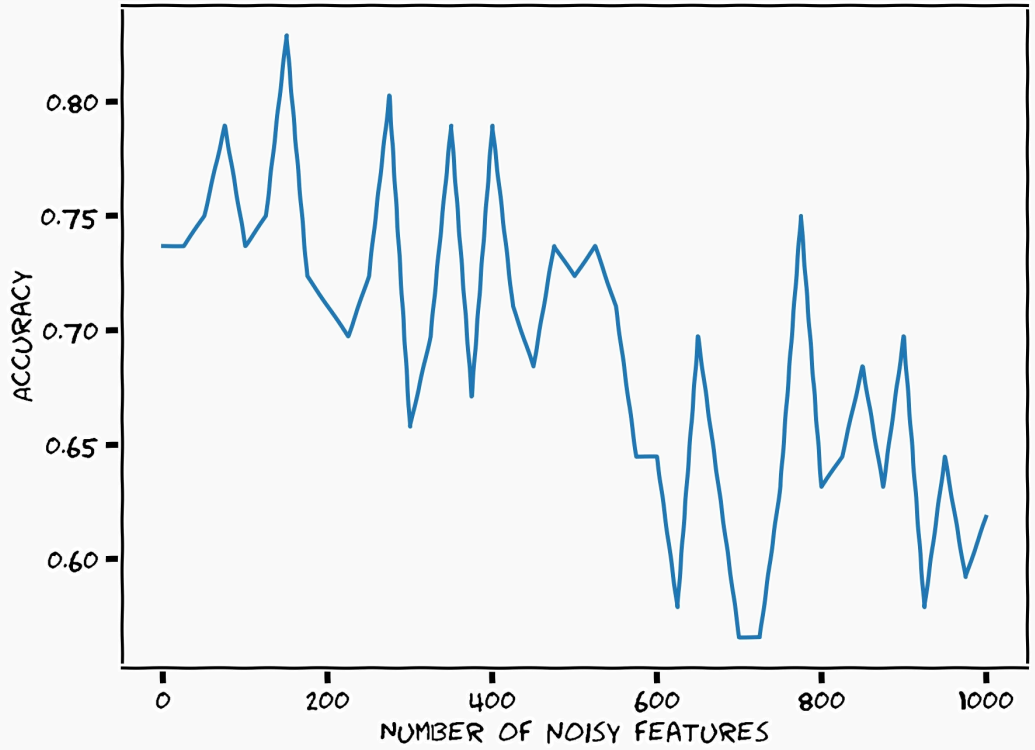


Final Thoughts on Random Forests

When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly.

Question: Why?

In each split, the chances of selecting a relevant predictor will be low and hence most trees in the ensemble will be weak models.



Final Thoughts on Random Forests

Increasing the number of trees in the ensemble generally does not increase the risk of overfitting.

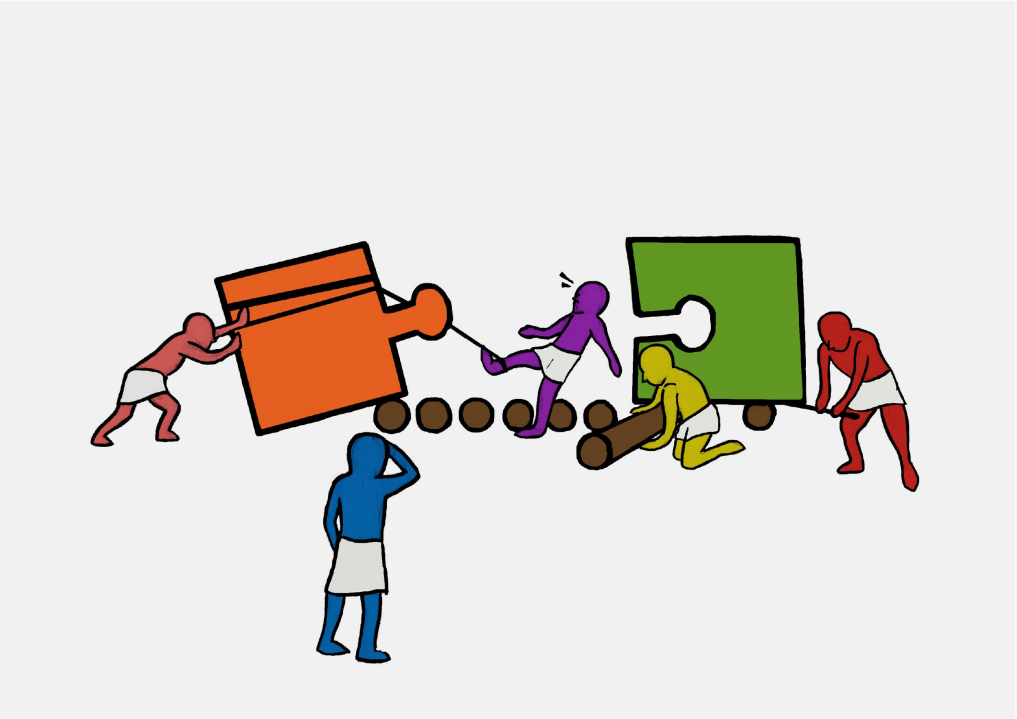
By decomposing the generalization error in terms of bias and variance, we see that increasing the number of trees produces a model that is at least as robust as a single tree.

Final Thoughts on Random Forests

Random Forest Classifier (and bagging) can return probabilities.

Question: How?

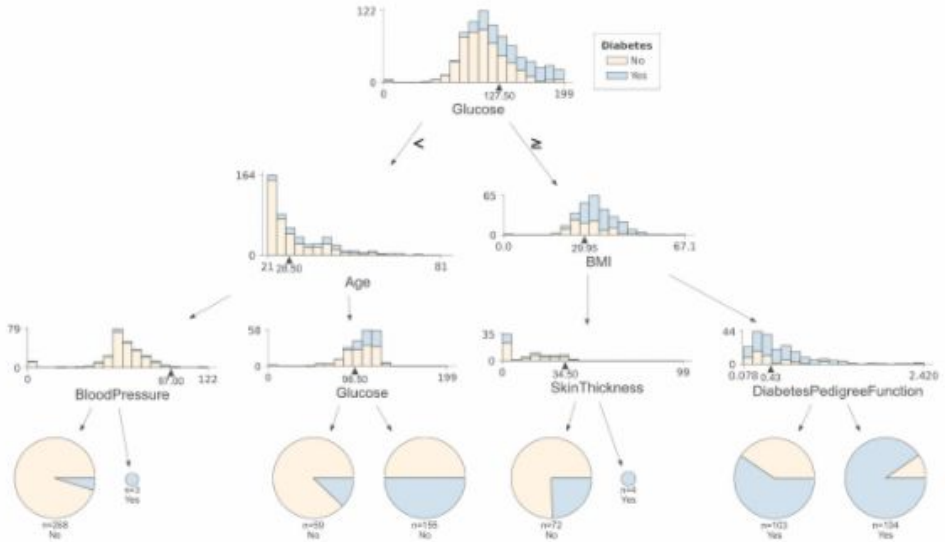
The predicted class probabilities of an input sample is computed as the mean predicted class probabilities of the trees in the forest.



🏋 Exercise: Bagging vs Random Forest (Tree correlation)

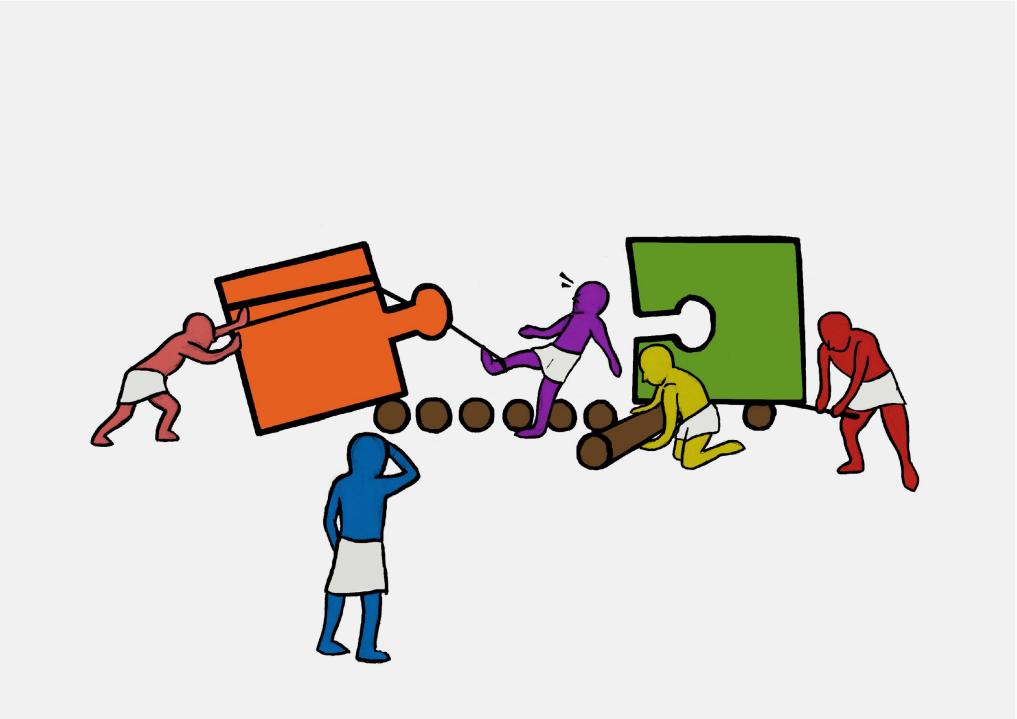
How does Random Forest improve on Bagging?

The goal of this exercise is to investigate the correlation between randomly selected trees from *Bagging* and *Random Forest*.



Instructions:

- Read the dataset `diabetes.csv` as a pandas dataframe, and take a quick look at the data.
- Split the data into *train* and *validation* sets.



Exercise: Hyperparameter tuning

Tuning the hyperparameters

Random Forests perform very well out-of-the-box, with the pre-set hyperparameters in `sklearn`. Some of the tunable parameters are:

- The number of trees in the forest: `n_estimators`, int, default=100
- The complexity of each tree: stop when a leaf has <= `min_samples_leaf` samples
- The sampling scheme: number of features to consider at any given split: `max_features` {"auto", "sqrt", "log2"}, int or float, default="auto".

Instructions:

- Read the datafile `diabetes.csv` as a Pandas data frame.
- Assign the predictor and response variable as mentioned in the scaffold.
- Split the data into train and validation sets.
- Define a vanilla Random Forest and fit the model on the entire data.
- For various hyper parameters of the model, define different Random Forest models and train on the data.
- Compare the results with each model.

Random Forest

Part B – Missing Data and Tree Building Algorithms

Pavlos Protopapas

Outline

- Motivation
- Random Forests
- Variable Importance for RF
- **Missing data**
- Imbalanced data
- Tree building algorithms

Missing Data: Surrogate splits

- When an observation is missing a value for predictor X, it cannot get past a node that splits based on this predictor. What can we do instead?

SURROGATE SPLITS

Surrogate splits - example

- Imagine situation with multiple features, two of them being phone_bill (continuous) and marital_status (categorical).
- Node 1 splits based on phone_bill.

	Left child	Right child
Phone_bill > 100	550R, 99G	50R, 301G

Surrogate splits - example

- Imagine situation with multiple features, two of them being phone_bill (continuous) and marital_status (categorical).
- Node 1 splits based on phone_bill.
- **Surrogate** search might find that marital_status = 1 generates a similar distribution of observations in left and right node.
- This condition is then chosen as top surrogate split (for prediction).

	Left child	Right child
Phone_bill > 100	550R, 99G	50R, 301G
Marital_status = 1	510R, 128G	51R, 311G

Missing Data: Surrogate splits

- When an observation is missing a value for predictor X, it cannot get past a node that splits based on this predictor. What can we do instead?
- We need surrogate splits: Mimic of actual split in a node but using another predictor. It is used in replacement of the original split in case a datapoint has missing data.
- To build them, we search for a feature-threshold pair that most closely matches the original split.
- Association measure is used to select surrogate splits. Depends on the probabilities of sending cases to a particular node + how the new split is separating observations of each class. Read more about it here.

Surrogate splits - example

- In our example, primary splitter = `phone_bill`
- We might find that surrogate splits include `marital_status`, `commute_time`, `age`, `city_of_residence`.
  - `commute_time` associated with more time on the phone
  - Older individuals might be more likely to call vs text
  - City variable hard to interpret because we don't know identity of cities
- Surrogates can help us understand primary splitter.
- Surrogate splits perform better when there is multicollinearity (just like imputation!)

Surrogate splits

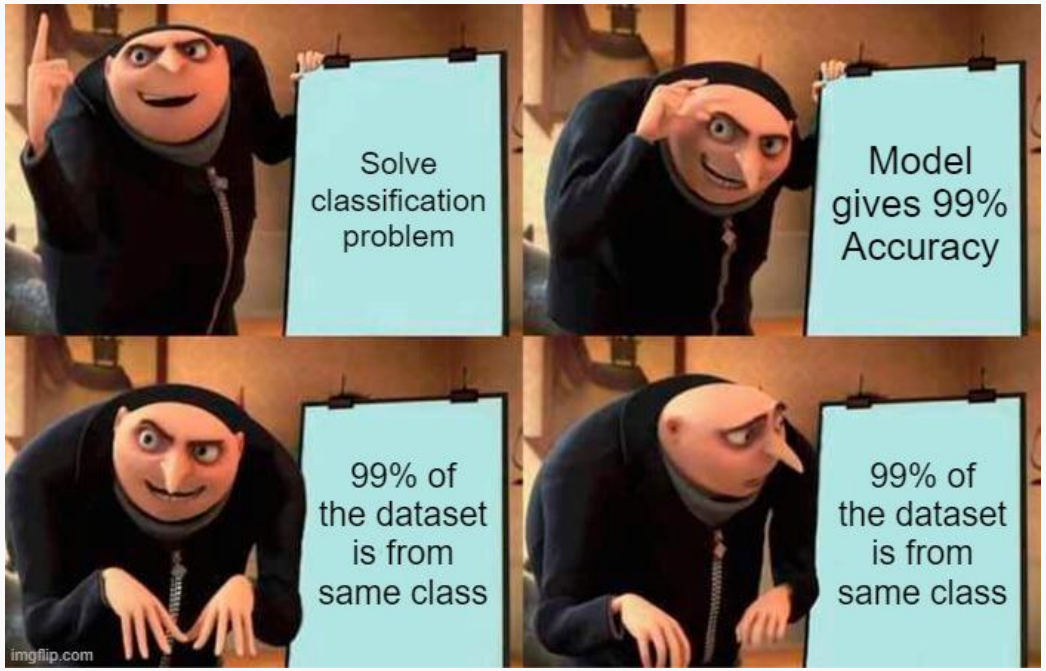
- The main function is to split when the primary splitter is missing, which *may* never happen in the training data, but being ready for future test data increases robustness.
- No guarantee that useful surrogates can be found.
- Sklearn's decision trees attempt to find **five** surrogates per split.
- Number of surrogates often varies from split to split.

Outline

- Motivation
- Random Forests
- Variable Importance for RF
- Missing data
- **Imbalanced data**
- Tree building algorithms

Imbalanced classes

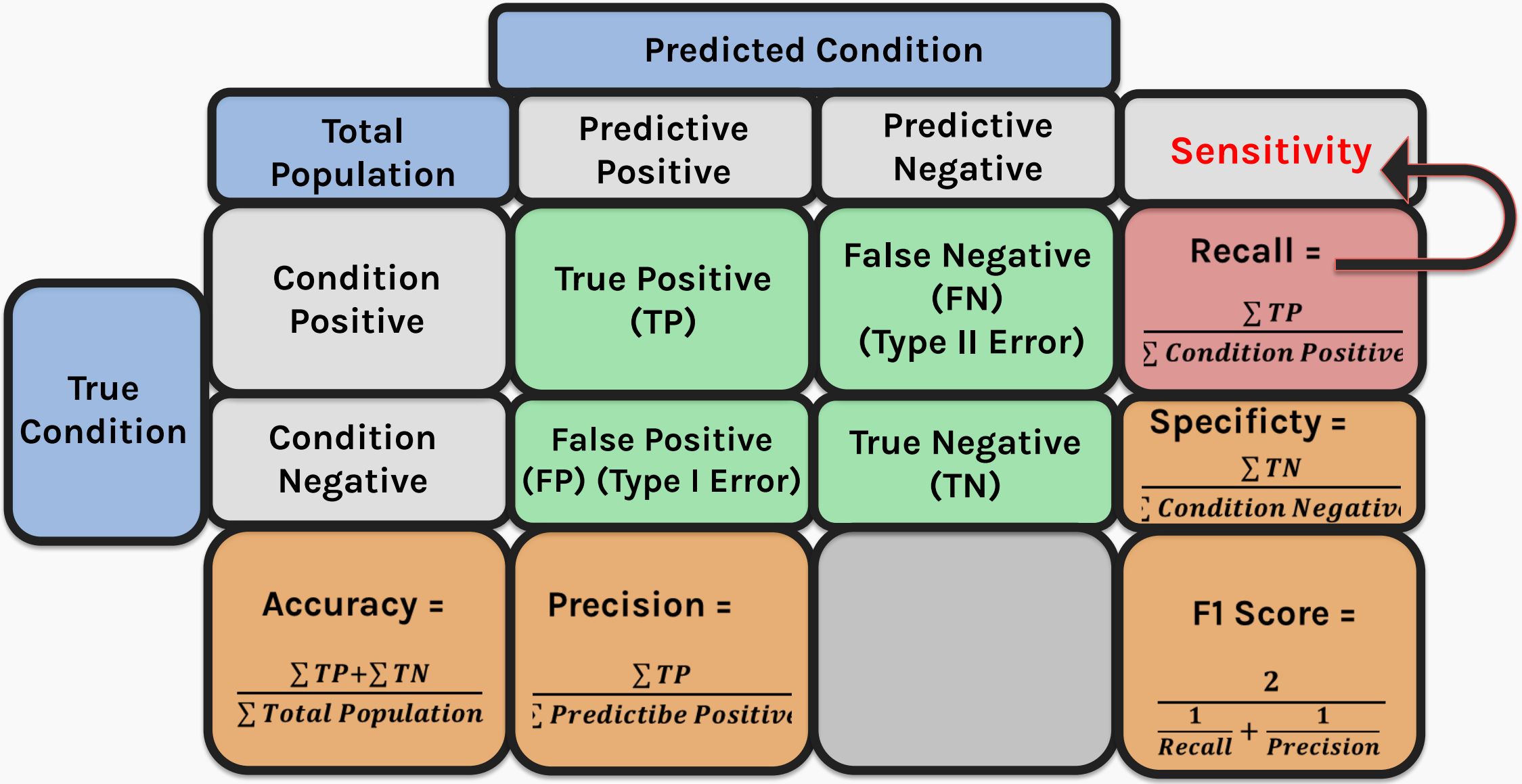
Training a RF (or any machine learning model) on an imbalanced dataset can introduce unique challenges to the learning problem.



Imbalanced classes

Using AUC score (or F1 score) is a better measure of the performance of a model when the classes are not balanced.

Metrics (review)



F1-score (review)

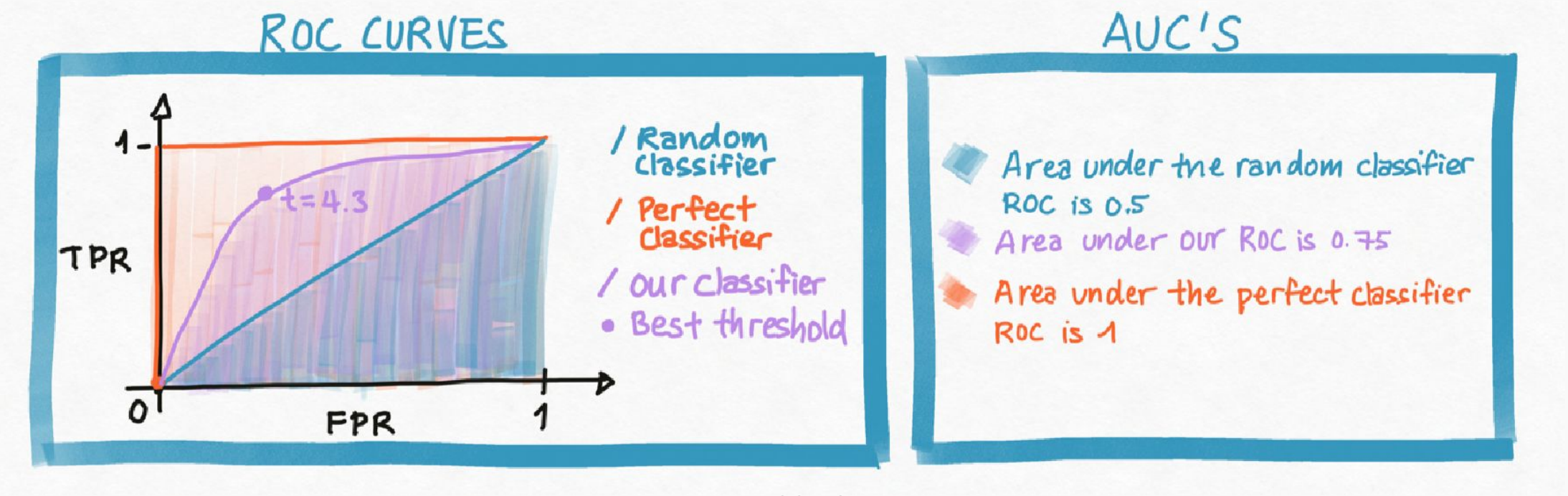
Accuracy is a great measure but only **when you have symmetric datasets** (false negatives & false positives counts are close), also, **false negatives & false positives have similar costs.**

If the cost of false positives and false negatives are different then F1 is your savior. **F1 is best if you have an uneven class distribution**

Area Under the ROC curve (review)

The ROC curve allows us to find the classification threshold that gives the best FPR and TPR trade-off.

We summarize the ROC by computing the Area Under the ROC curve. For the perfect classifier the AUC is 1, the random classifier has AUC of 0.5. We want our classifier to have AUC as close to 1 as possible.



Dealing with Imbalanced classes

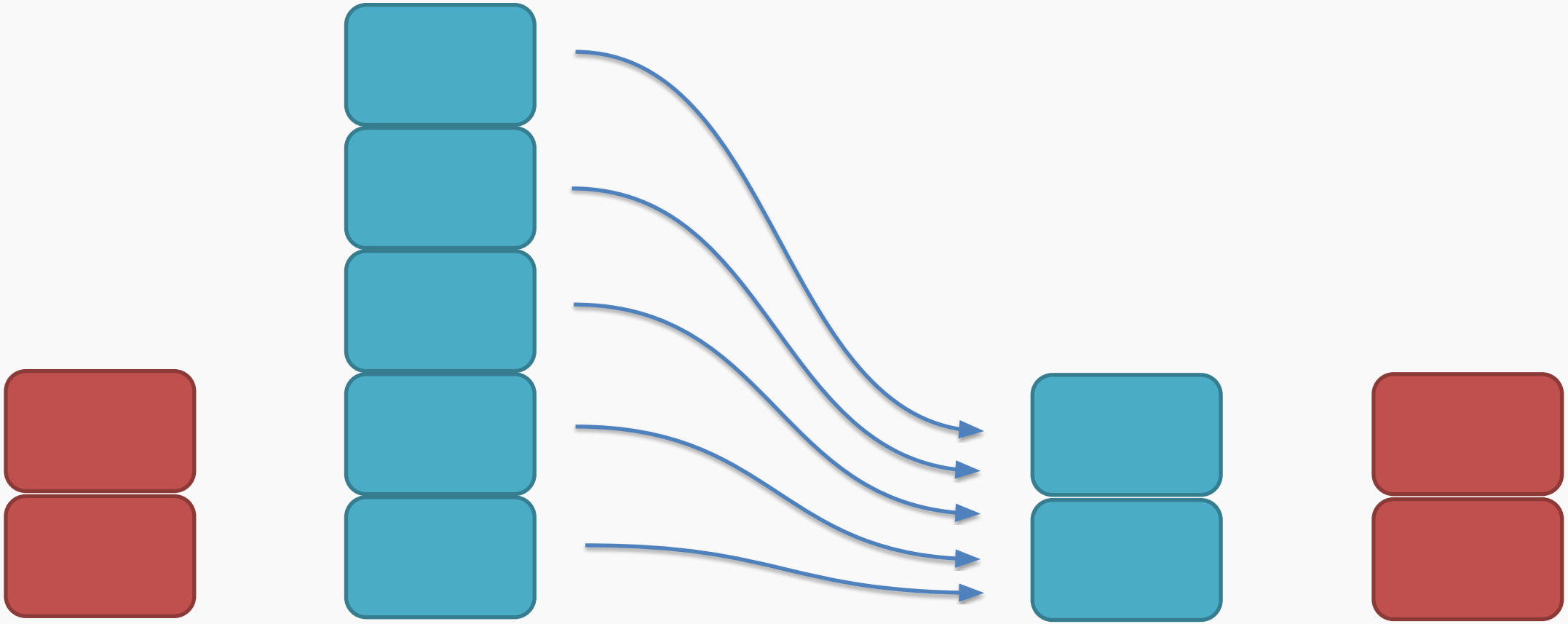
1. Undersampling
    i. Random Sampling
    ii. Near Miss

2. Oversampling
    i. Random Sampling
    ii. SMOTE

3. Class weighting

Dealing with Imbalanced classes

1. Undersampling



Dealing with Imbalanced classes

1. Undersampling

   We reduce the number of samples in majority class to match the number of samples in minority class.

   This can be done in two ways:

   i. Random Sampling:

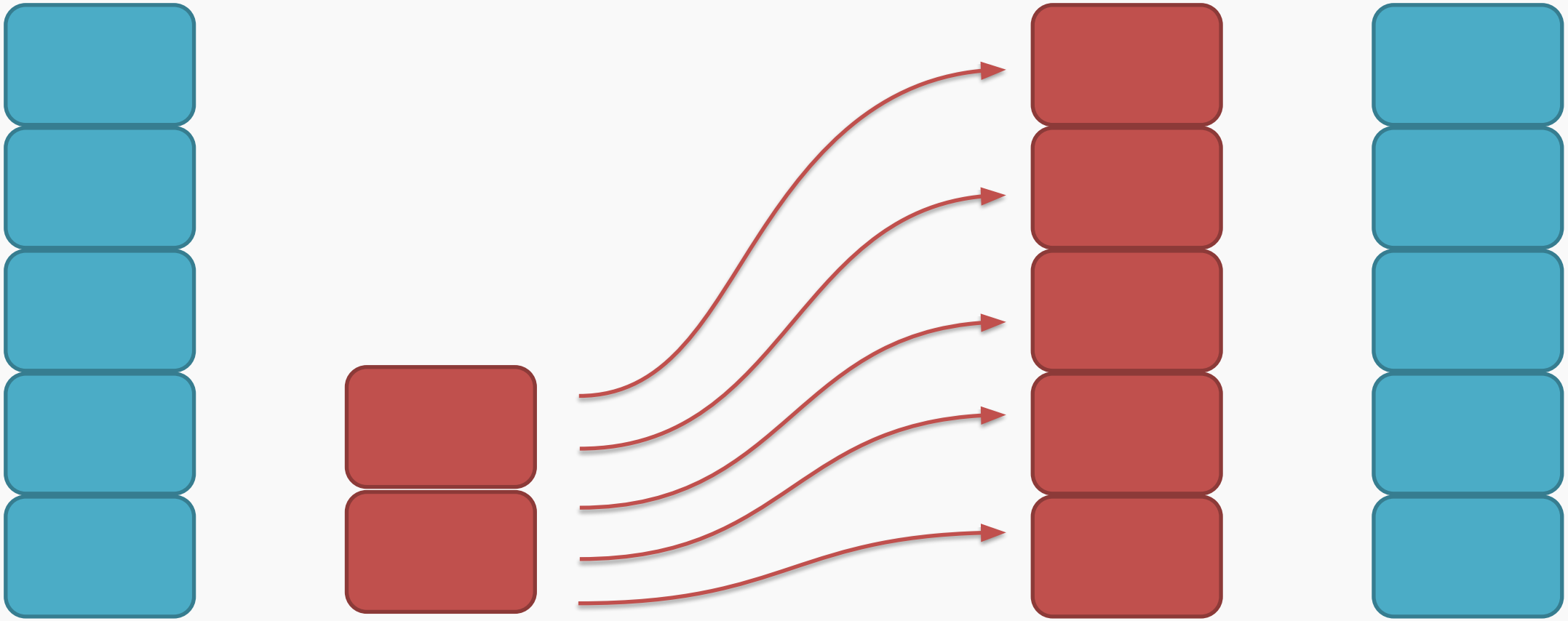
      Randomly sample from majority class with or without replacement.

   ii. Near Miss:

      Select data points by using simple heuristics like finding samples from which the average distance to some data points of minority class is smallest. Read more about it here.

Dealing with Imbalanced classes

2. Oversampling



Dealing with Imbalanced classes

2. Oversampling

We fight imbalanced data by generating new samples for minority class.

This can be done in two ways:

i. Random Sampling:

Randomly sample from minority class with **replacement**.

ii. SMOTE:

SMOTE is an improved alternative for oversampling.

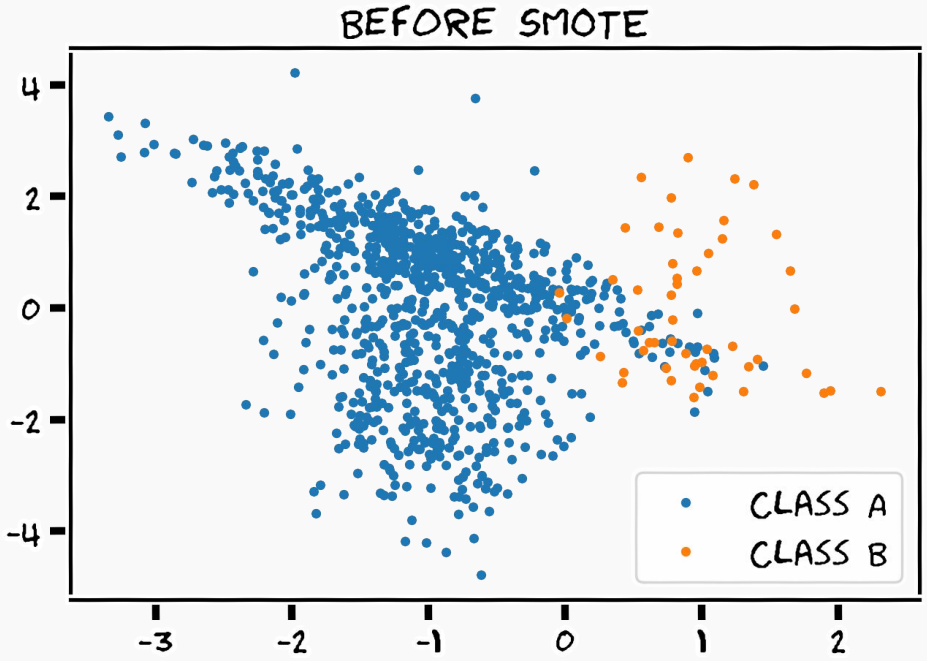
SMOTE (Synthetic Minority Oversampling Technique):

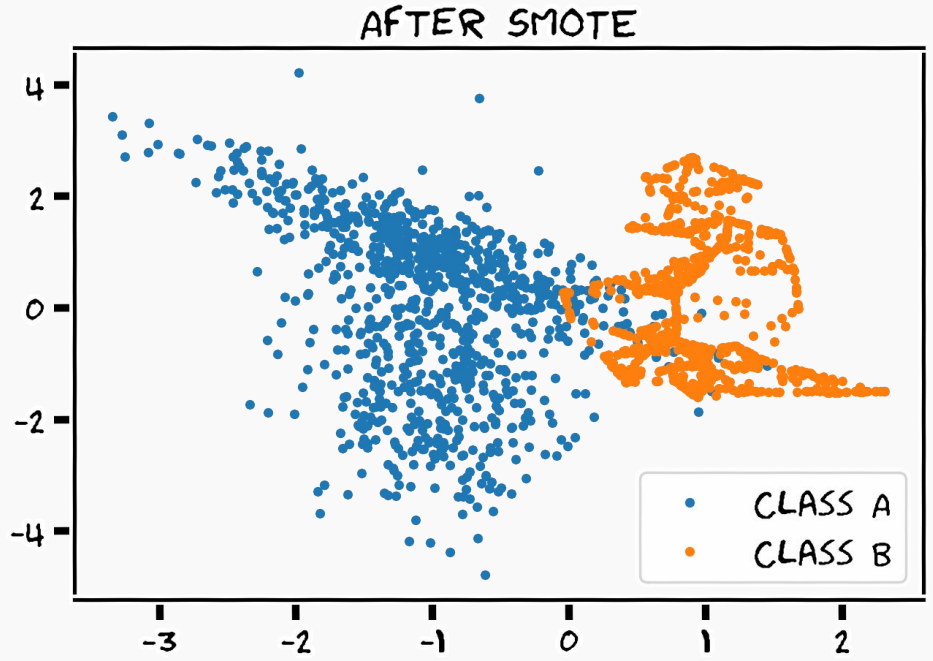
ii. SMOTE:

SMOTE is an improved alternative for oversampling.

SMOTE works by finding points that are closer in feature space.

Drawing a line between these points and generating new data points along this line.





Dealing with Imbalanced classes

3. Class weighting

A simple way to address the class imbalance is to provide a weight for each class which places more emphasis on the minority classes.

In sklearn we can provide the class weight as a dictionary or use

```
class_weight = balanced
```

Then it automatically adjust weights inversely proportional to class frequencies in the input data as:

$$W_k = \frac{N}{K \times N_K}$$

Where N is the total number of samples, N_k is the number of samples in class K and K is the total number of classes.

Imbalanced classes in sklearn

Sklearn we can provide the class weight as a dictionary or use

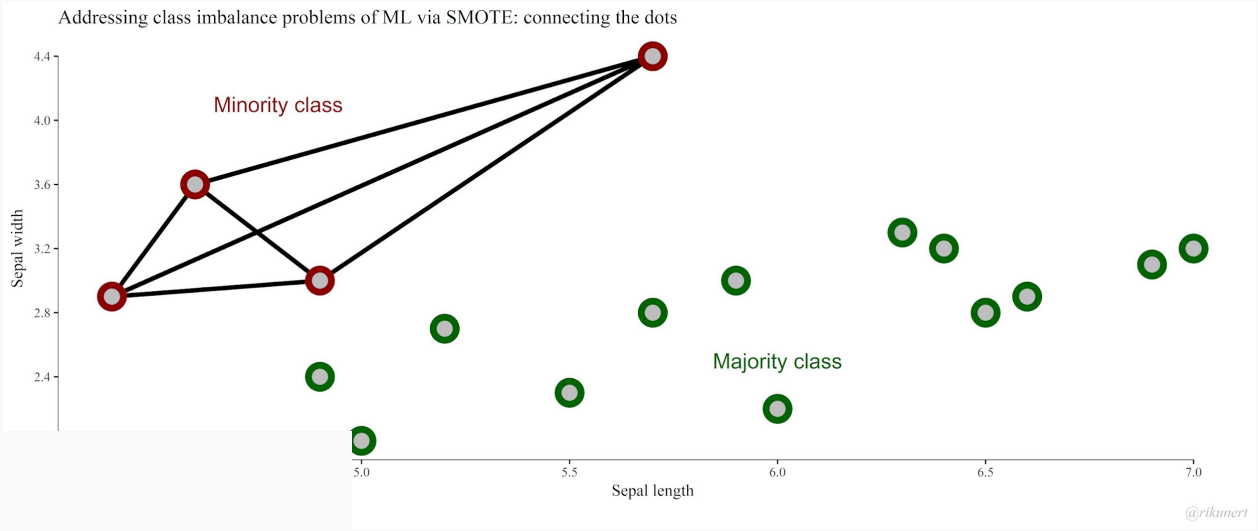
class_weight = balanced

Then it automatically adjust weights inversely proportional to class frequencies in the input data as

$$N$$

One step further

SMOTE: A library that does upsampling by “interpolation”. See exercise.



Outline

- Motivation
- Random Forests
- Variable Importance for RF
- Missing data
- Imbalanced dataset
- **Tree building algorithms**

Tree-building algorithms

Any tree-building algorithm needs to consider many things:

- what metric to decide splitting?
- where to look for splits?
- how to handle categorical predictors?
- how to handle missing features?
- how to handle imbalanced classes?

Different libraries may use different algorithms for building trees.

Tree-building algorithms

ID3: Iterative Dichotomiser 3. Developed in the 80s by Ross Quinlan.

- Uses the top-down induction approach:
  - Works with the **Information Gain** (IG) metric.
  - At each step, algorithm chooses feature to split on and calculates IG for each possible split along that feature.
  - Greedy algorithm.

Tree-building algorithms

C4.5: Successor of ID3. Main improvements over ID3:

- Works with both continuous and discrete features, while ID3 only works with discrete values.
- Handles missing values by using **fractional cases** (penalizes splits that have multiple missing values during training, fractionally assigns the datapoint to all possible outcomes).
- Reduces overfitting by **pruning**, a bottom-up tree reduction technique.
- Accepts weighting of input data.
- Works with multiclass response variables.

Tree-building algorithms

CART: Most popular tree-builder. Introduced by Breiman et al. in 1984. Usually used with Gini purity metric.

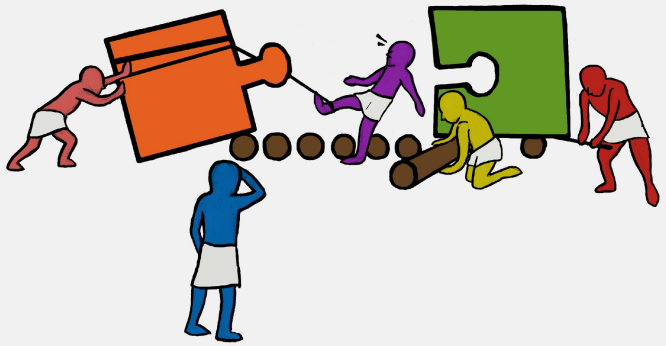
- Main characteristic: **builds binary trees.**
- Can work with discrete, continuous and categorical values in features.
- Handles missing values by using **surrogate splits**.
- Uses **cost-complexity pruning**.

Note: "scikit-learn uses an optimised version of the CART algorithm; however, scikit-learn implementation does not support categorical variables for now.“

Many more algorithms…

Feature	C4.5	CART	CHAID	CRUISE	GUIDE	QUEST
Unbiased Splits				√	√	√
Split Type	*u*	*u,l*	*u*	*u,l*	*u,l*	*u,l*
Branches/Split	≥2	2	≥2	≥2	2	2
Interaction Tests				√	√	
Pruning	√	√		√	√	√
User-specified Costs		√	√	√	√	√
User-specified Priors		√		√	√	√
Variable Ranking		√			√	
Node Models	*c*	*c*	*c*	*c,d*	*c,k,n*	*c*
Bagging & Ensembles					√	
Missing Values	*w*	*s*	*b*	*i,s*	*m*	*i*

b, missing value branch; *c*, constant model; *d*, discriminant model; *i*, missing value imputation; *k*, kernel density model; *l*, linear splits; *m*, missing value category; *n*, nearest neighbor model; *u*, univariate splits; *s*, surrogate splits; *w*, probability weights



Exercise: Random Forest with Class Imbalance

The goal of this exercise is to investigate the performance of Random Forest on a dataset with class imbalance, and then use corrections strategies to improve performance.

Your final comparison may look like the table below (but not with these exact values):

STRATEGY	F1 SCORE	AUC SCORE
No imbalance correction	0.44	0.68
Class weighting	0.47	0.67
Upsampling	0.57	0.71
Downsampling	0.63	0.72