# Random Forest Classifier

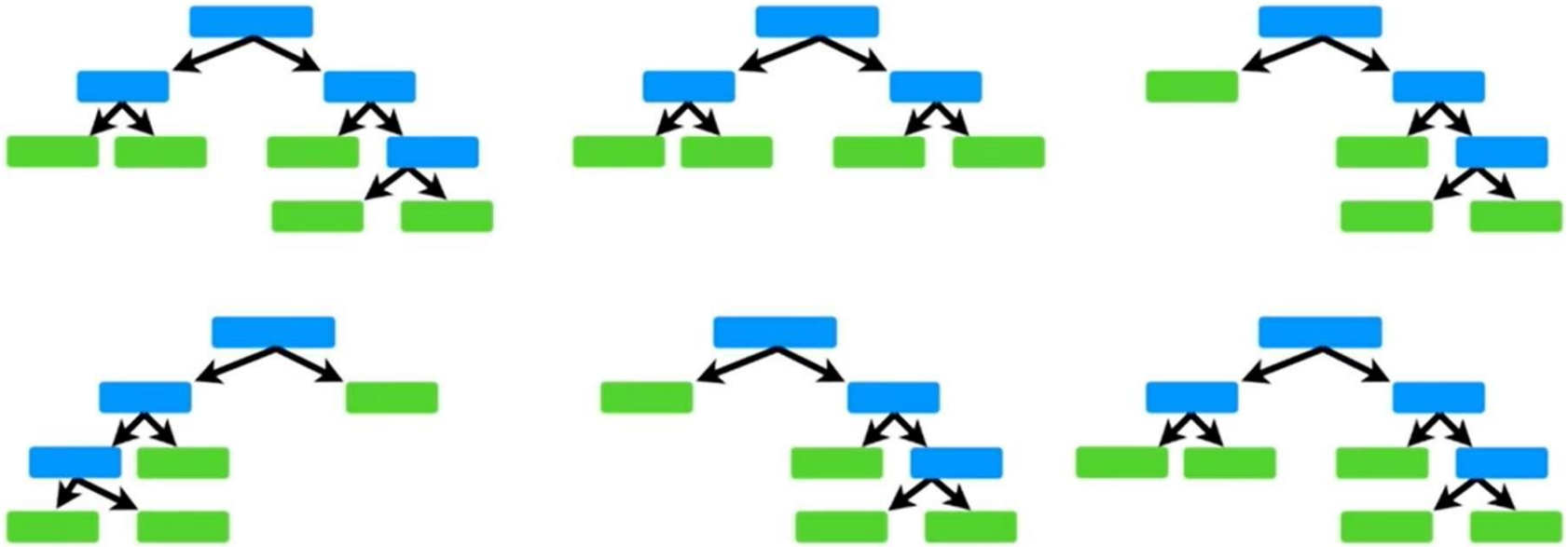
# Introduction

- Random Forest Classifiers are built from decision trees.
- Decision Trees are
  - Easy to build
  - Easy to use
  - Easy to interpret
- But in practice, they are not so good. According to *The Bible of Machine Learning* "Trees have one aspect that prevents them from being the ideal tool for predictive learning namely **inaccuracy".**
- All the variants of Decision Tree have following limitations.
  - They work great for the data used to create them, **but they are not flexible when it comes to classifying new examples.**
  - They are **computationally quite extensive** (especially when there are large number of training examples or training dimensions).

# Random Forests

- Random Forests combine the simplicity of decision trees with flexibility resulting in a vast improvement in accuracy.
- Random Forest are ***ensemble learning methods*** for classification and regression.
- ***Ensemble learning*** is the process by which multiple models, such as classifiers, are strategically generated and combined to solve a particular computational intelligence problem in order to improve the (classification, prediction, function approximation, etc.) performance of a model.
- Random forests operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.

# Random Forests (Contd.....)



# Random Forest Algorithm

Consider a training set with N examples and D- dimensions

1. For b=1 to B trees:
   a. Draw a bootstrap sample Z* of size N from the training examples.
   b. Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
      i. Select m variables at random from the D-variables (generally m<<D)
      ii. Pick the best variable/ split variable among the m according to any of the evaluation metric such as Information Gain , Gain Ratio, or Gini Index
      iii. Split the node into its daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$ i.e. outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.

This kind of ensemble where the output from various bootstrapped sample is computed by voting for majority or averaging is called **bagging (or bootstrap aggregating).**

# Bootstrapped Dataset- Step 1(a) Algorithm

- A bootstrapped sample of same size of the training data, is created by randomly selecting samples from the original dataset such that we are allowed to pick the same sample more than once.
- For instance, in the figure shown, the bootstrapped dataset has been created from the original dataset by randomly selecting examples 1 and 2 once, and example 4 twice. Though example 3 has not been chosen.

Original Dataset

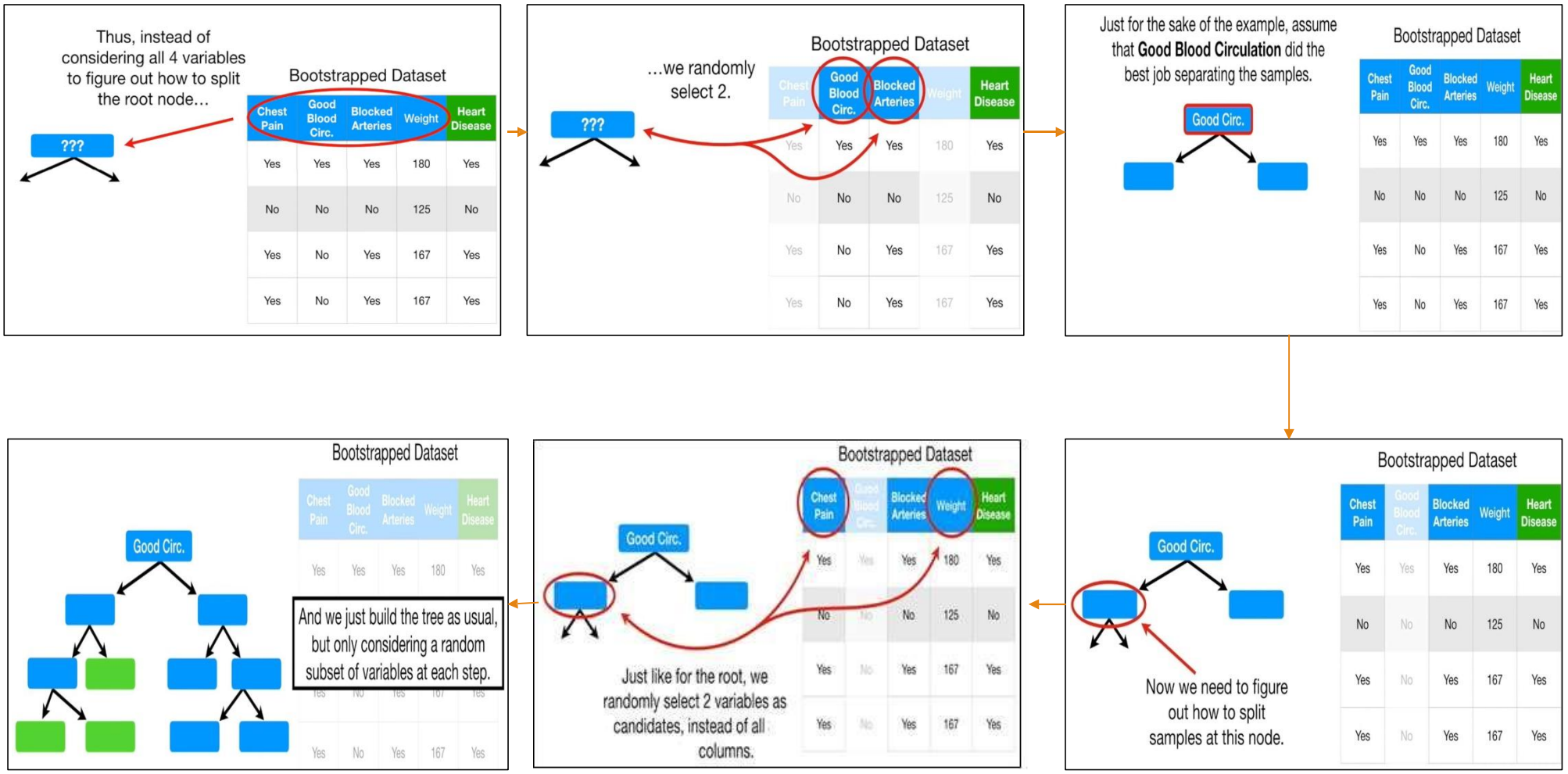
| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | No | Yes | 167 | Yes |

Bootstrapped Dataset

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| Yes | Yes | Yes | 180 | Yes |
| No | No | No | 125 | No |
| Yes | No | Yes | 167 | Yes |
| Yes | No | Yes | 167 | Yes |

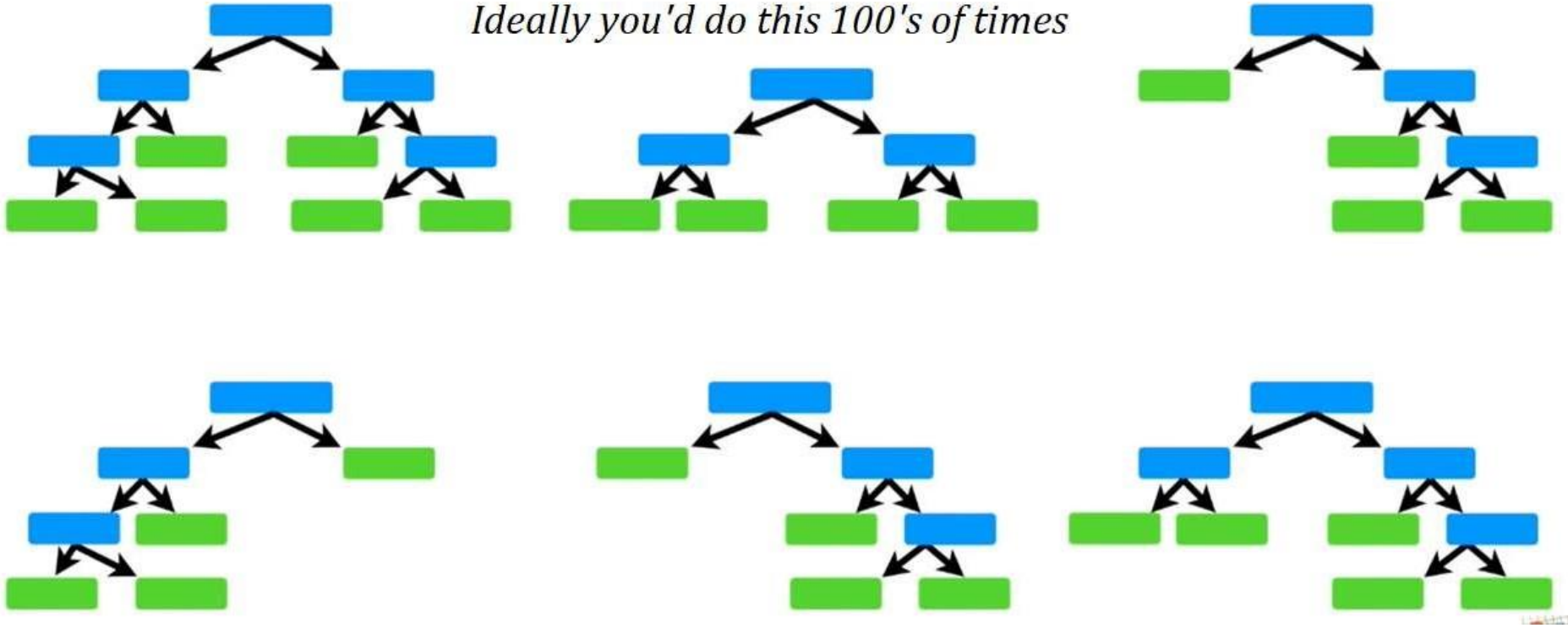
# Generate Random Forest Tree using bootstrapped dataset -Step 2(b) Algorithm

- The decision tree is created from the bootstrapped dataset, but only use a random subset of variables (or columns) at each step.
- For example in the figure shown (in next slide), we considered 2 variables at each step.
- Though there is a method to determine optimal number of variables at step (discussed later in the slides).

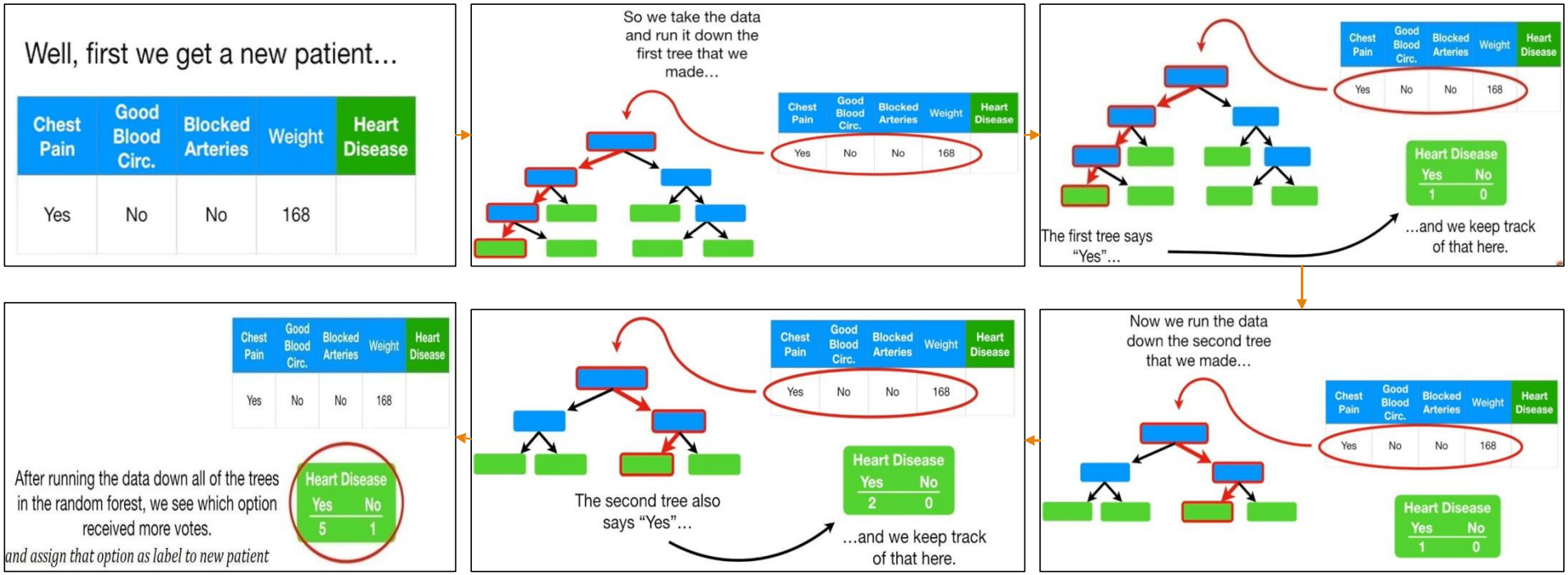

Thus, instead of considering all 4 variables to figure out how to split the root node…
???
Bootstrapped Dataset
Chest Pain
Good Blood Circ.
Blocked Arteries
Weight
Heart Disease
Yes Yes Yes 180 Yes
No No No 125 No
Yes No Yes 167 Yes
Yes No Yes 167 Yes
…we randomly select 2.
???
Bootstrapped Dataset
Just for the sake of the example, assume that Good Blood Circulation did the best job separating the samples.
Good Circ.
Bootstrapped Dataset
Now we need to figure out how to split samples at this node.
Good Circ.
Bootstrapped Dataset
Just like for the root, we randomly select 2 variables as candidates, instead of all columns.
Good Circ.
Bootstrapped Dataset
And we just build the tree as usual, but only considering a random subset of variables at each step.
Good Circ.

**Now go back to Step 1 and repeat:** Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.

*Ideally you'd do this 100's of times*



# Ensemble of Trees- Step 2



# Estimate the accuracy of Random Forest

- While creating a bootstrapped dataset, many entries are not included in any of the bootstrapped dataset.
- Typically, about 1/3$^{rd}$ of the original dataset does not end up in the bootstrapped dataset.
- The entries which are not included in any of the bootstrapped datasets are called as "**Out-of-Bag Dataset**". The accuracy of the random forest is computed on the Out-of-Bag dataset.
- The proportion of Out-of-Bag samples that are incorrectly classified is the "**out-of-Bag Error**" which needs to be minimized.

# Tuning Parameters for Random Forest

1. How to determine optimal numbers of features to consider at each split point?
   - Consider different values of optimal number of features.
   - Compute Out-of-Bag error from all the Out-of-Bag samples for each value of optimal number of features
   - Choose the value of optimal number of features for which Out-of-Bag error is minimum.
2. How to determine the number of trees the algorithm builds?
   - In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.
   - This parameter is also computed using the same steps as discussed in point 1.

# Tuning Parameters for Random Forest (Contd…)

3. Which features to select at each step?

- Usually, the features are selected at random.
- But, we can also compute correlation among the randomly selected features at each step and reject the features which are highly correlated.
- We must also check correlation between features and the output variable as We need features that have at least some predictive power.

# Advantages of Random Forests

1. Random Forests Classifier has improved accuracy over Decision Tree Classifier
   - Random forests often achieve higher accuracy than a single decision tree
   - This is due to the reason that it does not consider each training examples to form a random forests of trees. So, it does not exactly generalize the training data and does not overfit.
   - Moreover, it employs some sort of cross-validation on Out-of-Bag datasets while tuning parameters.
   - Hence, **they are flexible when it comes to classifying new examples.**
2. Random Forest Classifiers are computationally less expensive.
   - Random Forest Classifiers consider a subset of features at each point. So this reduces a lot of computations especially when the feature space is large.
   - Though it needs to generate a large number of trees which makes the algorithm a bit slow but it improves accuracy drastically.

# Advantages of Random Forests (Contd…)

3. Feature importance: Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way.
   - The first step in measuring the feature importance in a data set is to fit a random forest to the data.
   - During the fitting process the out-of-bag error for each data point is recorded and averaged over the forest.
   - To measure the importance of the j-th feature after training, the values of the j-th feature are shuffled among the training data and the out-of-bag error is again computed on this perturbed data set.
   - The importance score for the j-th feature is computed by averaging the difference in out-of-bag error before and after the permutation over all trees.
   - The score is normalized by the standard deviation of these differences for all features.

# Advantages of Random Forests (Contd…)

4. Random Forests Classifiers can also be used to handle the missing values in the dataset.
   - Whenever, there is some missing values in training data, random forest applies following general technique," It makes an initial guess and then gradually refines the guess until it is hopefully good.
   - Initial bad guess can be the mode for categorical feature and mean/median for a numerical feature.
   - The guesses are refined by finding samples which are similar to one with missing data.

# Advantages of Random Forests (Contd…)

- In order to determine similarity following steps are followed.
  i. Build a random forest.
  ii. Run all the data down all the trees.
  iii. Note the samples that end up at same leaf nodes. Keep track of similar samples using a Proximity Matrix.
  iv. Then divide each entry in proximity matrix with total number of trees.
  v. Use proximity values to make better guesses about the missing data i.e. calculated weighted frequency of each value using proximity values as weight.
  vi. Repeat steps 5 and 6 until the missing values converge.

Original Dataset

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | No | ??? | ??? | No |

However, for patient #4, we've got some missing data.

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | **No** | ??? | No |

*Since No is the most commonly occurring value (it occurs 2 out of 3 times)*

← So "**No**" is our initial guess.

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | No | **180** | No |

← In this case, the median value is 180

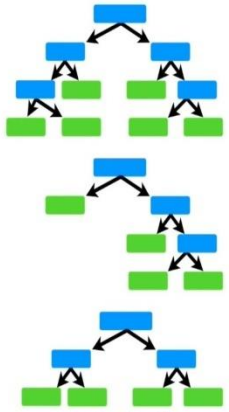
Filled-in Missing Values

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | **No** | **180** | No |

Now we want to refine these guesses.

We do this by first determining which samples are similar to the one with missing data.

Step 1: Build a random forest...

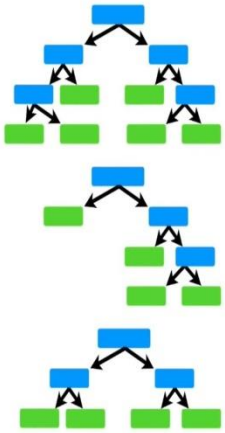


etc. etc. etc.

Filled-in Missing Values

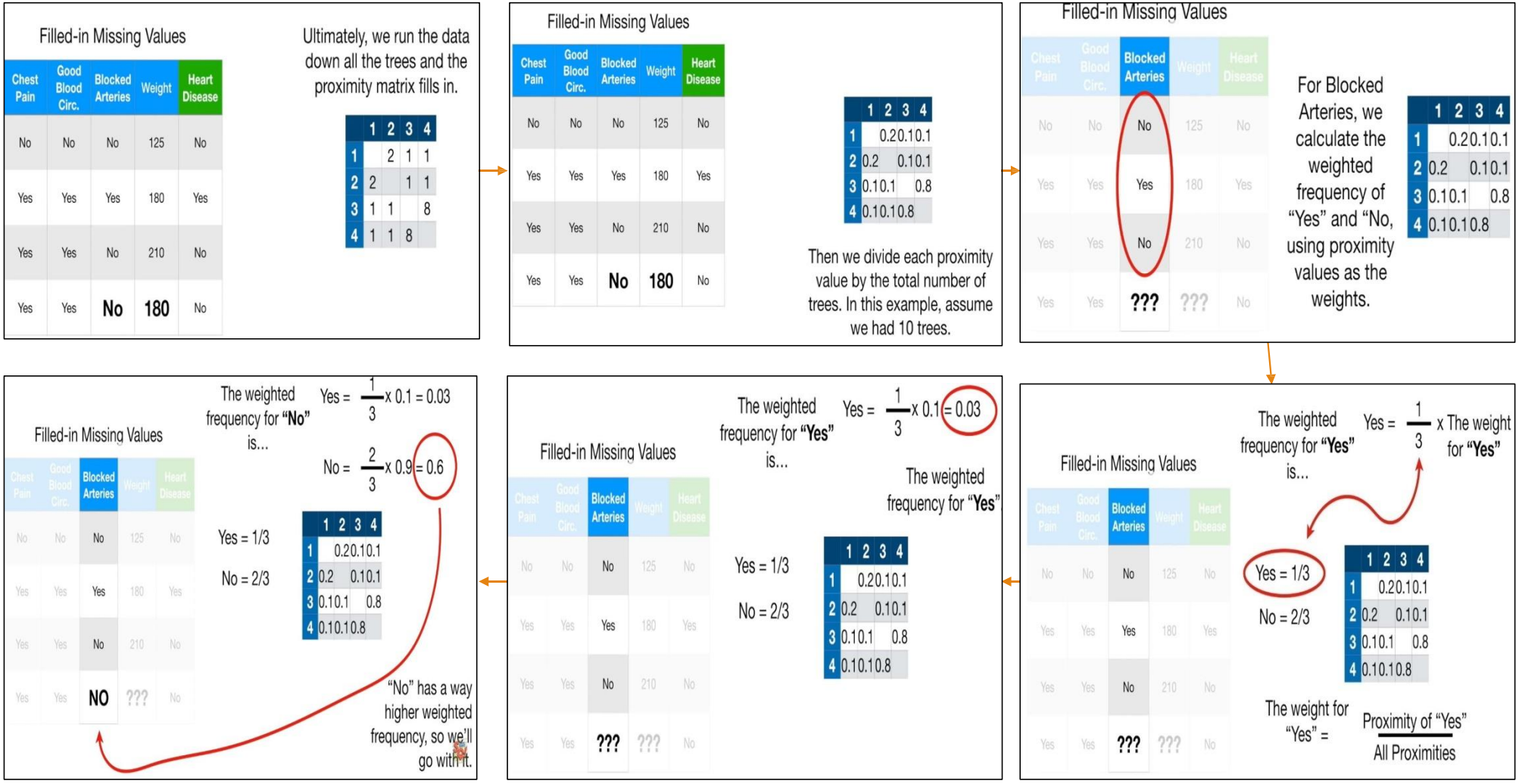
| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | **No** | **180** | No |

Step 2: Run all of the data down all of the trees.



Filled-in Missing Values

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | **No** | **180** | No |


Filled-in Missing Values
Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease
No | No | No | 125 | No
Yes | Yes | Yes | 180 | Yes
Yes | Yes | No | 210 | No
Yes | Yes | No | 180 | No
Ultimately, we run the data down all the trees and the proximity matrix fills in.
1 2 3 4
1 2 1 1
2 2 1 1
3 1 1 8
4 1 1 8
Filled-in Missing Values
Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease
No | No | No | 125 | No
Yes | Yes | Yes | 180 | Yes
Yes | Yes | No | 210 | No
Yes | Yes | No | 180 | No
1 2 3 4
1 0.2 0.1 0.1
2 0.2 0.1 0.1
3 0.1 0.1 0.8
4 0.1 0.1 0.8
Then we divide each proximity value by the total number of trees. In this example, assume we had 10 trees.
Filled-in Missing Values
Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease
No | No | No | 125 | No
Yes | Yes | Yes | 180 | Yes
Yes | Yes | No | 210 | No
Yes | Yes | ??? | ??? | No
For Blocked Arteries, we calculate the weighted frequency of "Yes" and "No, using proximity values as the weights.
1 2 3 4
1 0.2 0.1 0.1
2 0.2 0.1 0.1
3 0.1 0.1 0.8
4 0.1 0.1 0.8
Filled-in Missing Values
Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease
No | No | No | 125 | No
Yes | Yes | Yes | 180 | Yes
Yes | Yes | No | 210 | No
Yes | Yes | ??? | ??? | No
The weighted frequency for "Yes" is... Yes = 1/3 x The weight for "Yes"
Yes = 1/3
No = 2/3
1 2 3 4
1 0.2 0.1 0.1
2 0.2 0.1 0.1
3 0.1 0.1 0.8
4 0.1 0.1 0.8
The weight for "Yes" = Proximity of "Yes" / All Proximities
Filled-in Missing Values
Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease
No | No | No | 125 | No
Yes | Yes | Yes | 180 | Yes
Yes | Yes | No | 210 | No
Yes | Yes | ??? | ??? | No
The weighted frequency for "Yes" is... Yes = 1/3 x 0.1 = 0.03
Yes = 1/3
No = 2/3
1 2 3 4
1 0.2 0.1 0.1
2 0.2 0.1 0.1
3 0.1 0.1 0.8
4 0.1 0.1 0.8
The weighted frequency for "Yes"
Filled-in Missing Values
Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease
No | No | No | 125 | No
Yes | Yes | Yes | 180 | Yes
Yes | Yes | No | 210 | No
Yes | Yes | NO | ??? | No
The weighted frequency for "No" is... Yes = 1/3 x 0.1 = 0.03
No = 2/3 x 0.9 = 0.6
Yes = 1/3
No = 2/3
1 2 3 4
1 0.2 0.1 0.1
2 0.2 0.1 0.1
3 0.1 0.1 0.8
4 0.1 0.1 0.8
"No" has a way higher weighted frequency, so we'll go with it.

Weighted average = (125 x 0.1) + (180 x 0.1) + (210 x 0.8)

= 198.5

Filled-in Missing Values

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | NO | ??? | No |

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | | 0.2 | 0.1 | 0.1 |
| 2 | 0.2 | | 0.1 | 0.1 |
| 3 | 0.1 | 0.1 | | 0.8 |
| 4 | 0.1 | 0.1 | 0.8 | |

Filled-in Missing Values

| Chest Pain | Good Blood Circ. | Blocked Arteries | Weight | Heart Disease |
|---|---|---|---|---|
| No | No | No | 125 | No |
| Yes | Yes | Yes | 180 | Yes |
| Yes | Yes | No | 210 | No |
| Yes | Yes | **NO** | **198.5** | No |

Now that we've revised our guesses a little bit, we do the whole thing over again…

We build a random forest, run the data through the trees, recalculate the proximities and recalculate the missing values.

We do this 6 or 7 times until the missing values converge (i.e. no longer change each time we recalculate).