

MACHINE LEARNING

CHAP 5 RANDOM FOREST

Jean Tabet
j.tabet@ul.edu.lb

➔ RANDOM FOREST

Introduction

- “Trees have one aspect that prevents them from being the ideal tool for predictive learning, namely inaccuracy.”
- In other words, a single tree on its own is usually not accurate enough for robust predictive modelling.
- In decision trees, trees are very interpretable and flexible, but they tend to suffer from high variance (overfitting to training data) and high bias (when overly shallow).
- Ensemble methods like **Random Forests** (bagging many trees to reduce variance).
- Random Forest balances accuracy, robustness, and interpretability while overcoming the weaknesses of single decision trees

➔ RANDOM FOREST

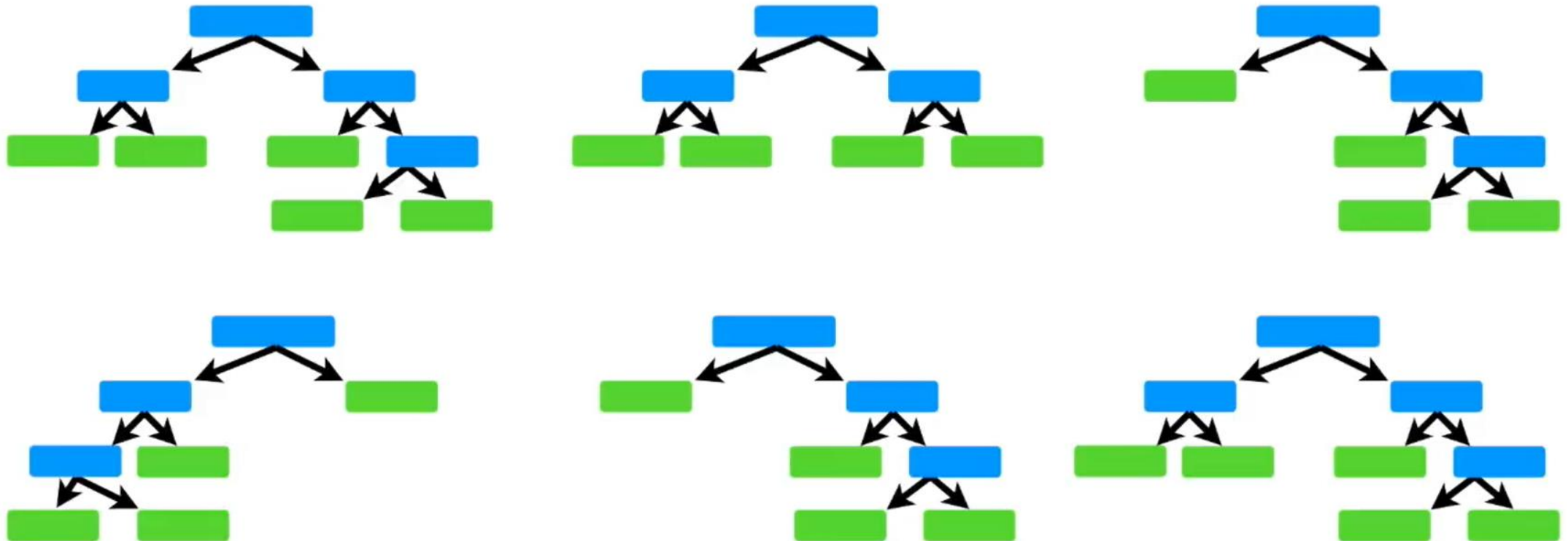
Introduction

- Healthcare & Medicine: Disease prediction (e.g., diabetes, cancer, heart disease)., identifying important biomarkers from genetic data.
- Finance & Banking: Credit scoring and loan default prediction, fraud detection in transactions.
- E-commerce & Marketing: Customer segmentation and churn prediction, product recommendation systems.
- Environment & Agriculture: predicting crop yield and soil quality, classifying land cover from satellite images.
- Cybersecurity: Intrusion detection systems (IDS), spam and phishing email filtering.
- Industry & Manufacturing: Predictive maintenance of machines, quality control and defect detection.

→ RANDOM FOREST

Accuracy

The good news is that **Random Forests** combine the simplicity of decision trees with flexibility resulting in a vast improvement in accuracy.



➔ RANDOM FOREST

Step 1:

Creation of Bootstrapped dataset that is the same size of the original dataset by selecting randomly samples (data rows) from the original dataset.

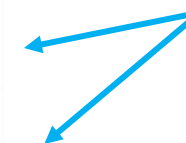
Original Dataset

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

The important detail is that we're allowed to pick the same sample more than once.



➔ RANDOM FOREST

Step 2:

We create a decision tree using the bootstrapped dataset, but use a random subset of variables at each step.

Thus, instead of considering all 4 variables to figure out how to split the root node...

...we randomly select 2.



In this case, we randomly selected **Good Blood Circulation** and **Blocked Arteries** as candidates for 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

➔ RANDOM FOREST

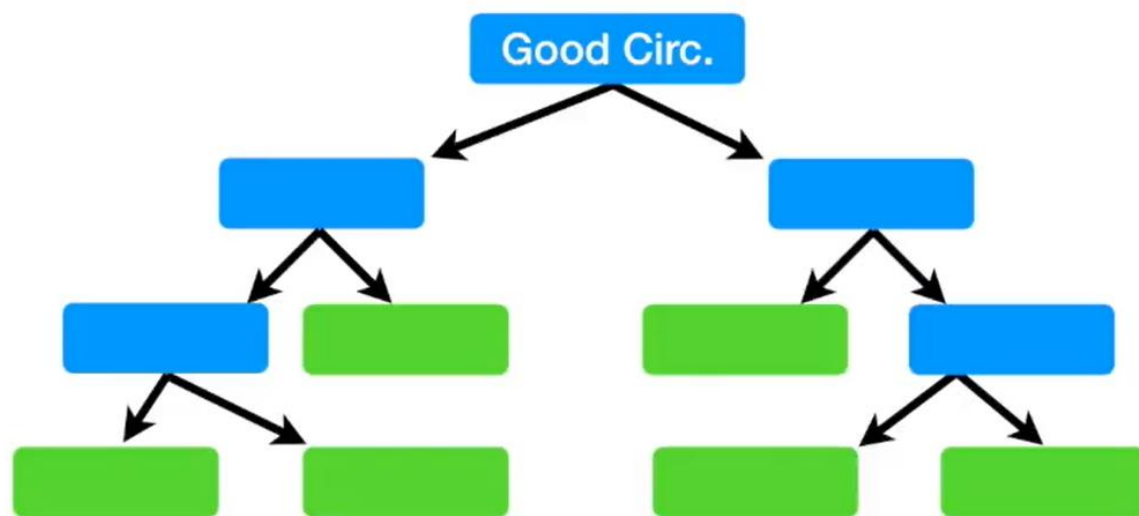
Step 2:

We create a decision tree using the bootstrapped dataset, but use a random subset of variables at each step.

And we just build the tree as usual, but only considering a random subset of variables at each step.

Bootstrapped Dataset

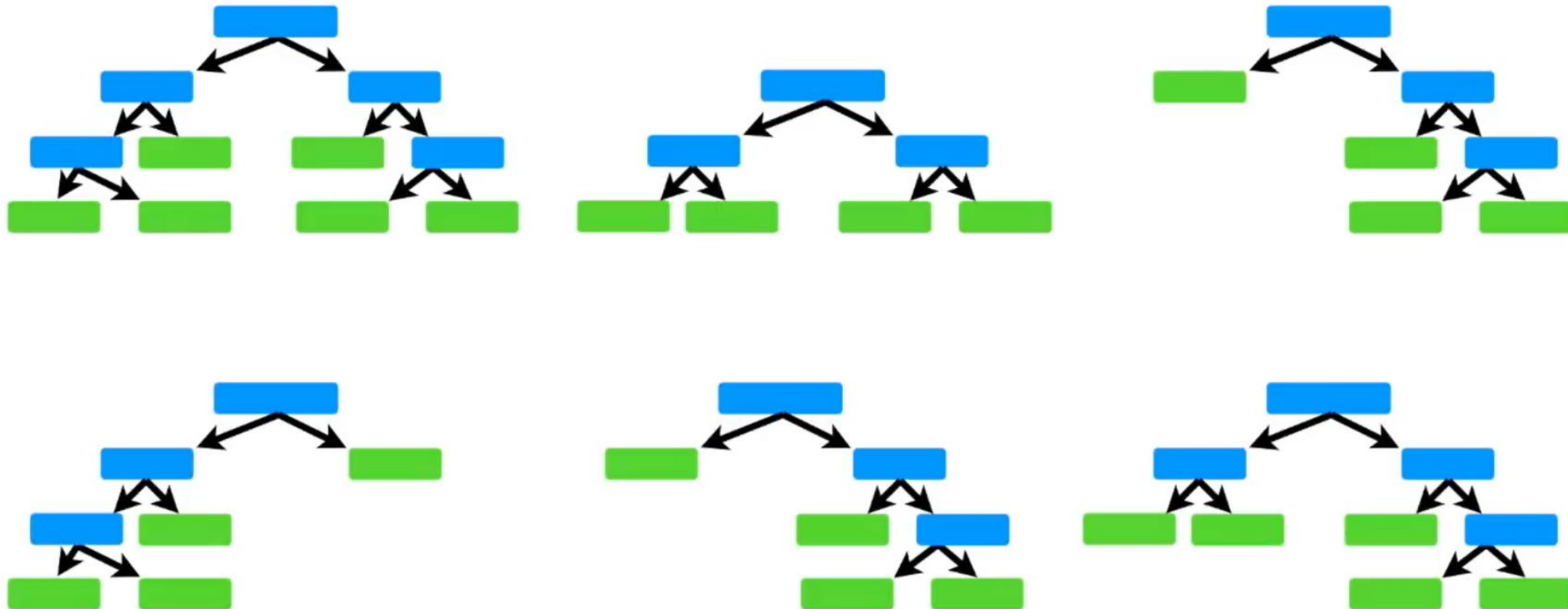
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes



➔ RANDOM FOREST

Redo step 1 & step 2

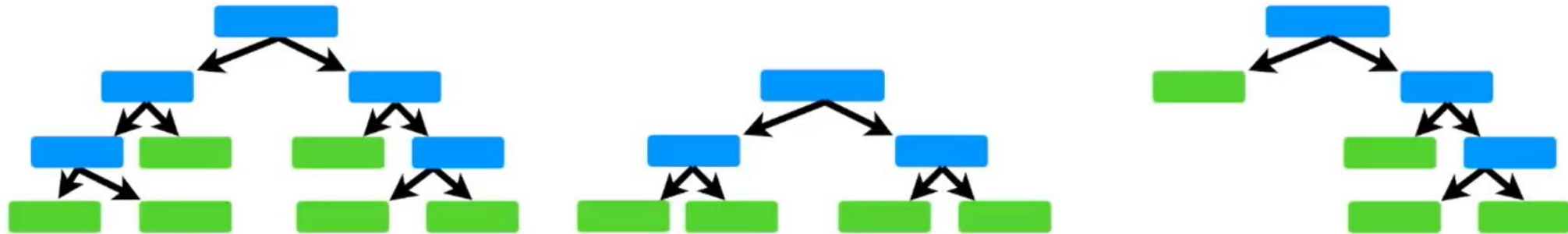
Now go back to Step 1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.



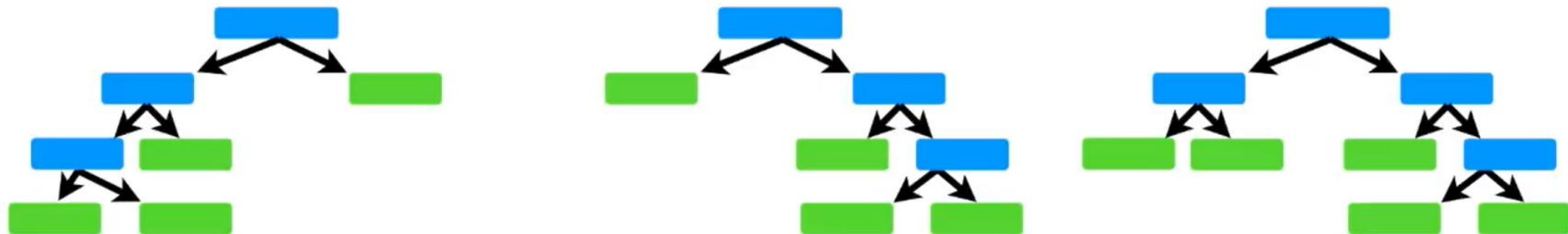
➔ RANDOM FOREST

Result: Large variety of trees

Using a bootstrapped sample and considering only a subset of the variables at each step results in a wide variety of trees.



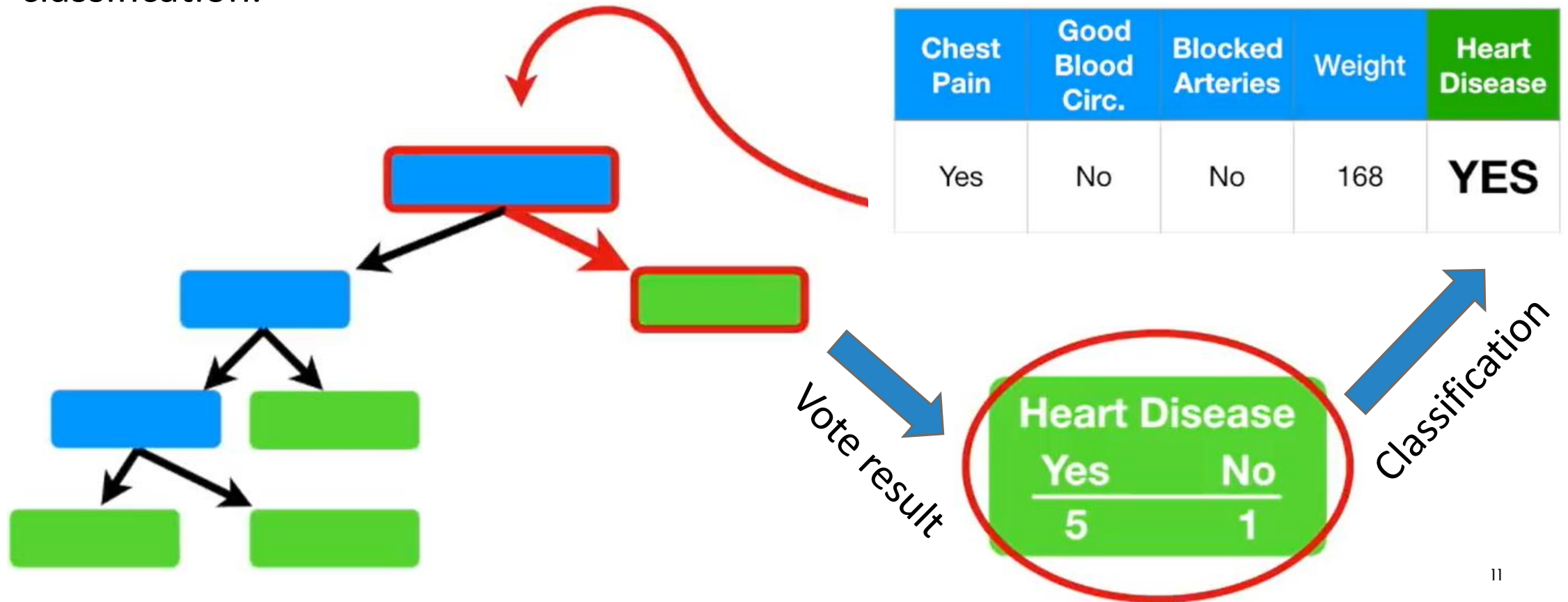
The variety is what makes random forests more effective than individual decision trees.



→ RANDOM FOREST

Classification of a new observation

- We run the data down each tree and we count the votes of all trees to make the classification.



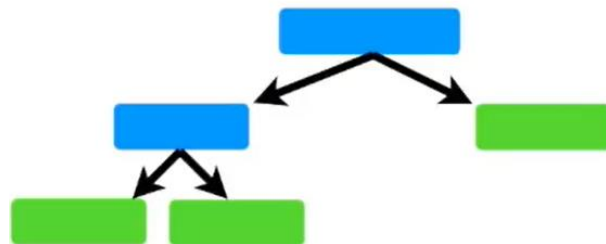
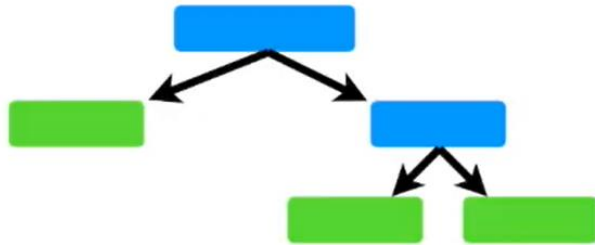
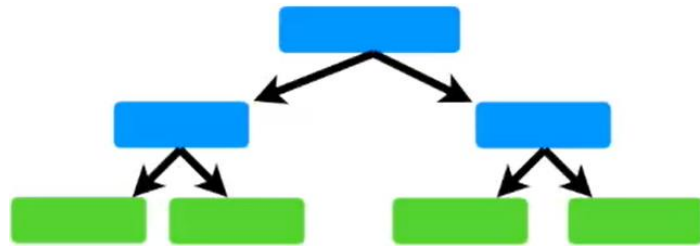
➔ RANDOM FOREST

Out-of-bag dataset:

- When we created the bootstrapped dataset, we allowed duplicate entries.
- As a result, around $1/3$ of the original data does not end up in bootstrapped dataset.
- All entries that are not included in the bootstrapped dataset are called Out-Of-Bag Dataset.

→ RANDOM FOREST

Classification of Out-of-bag sample:

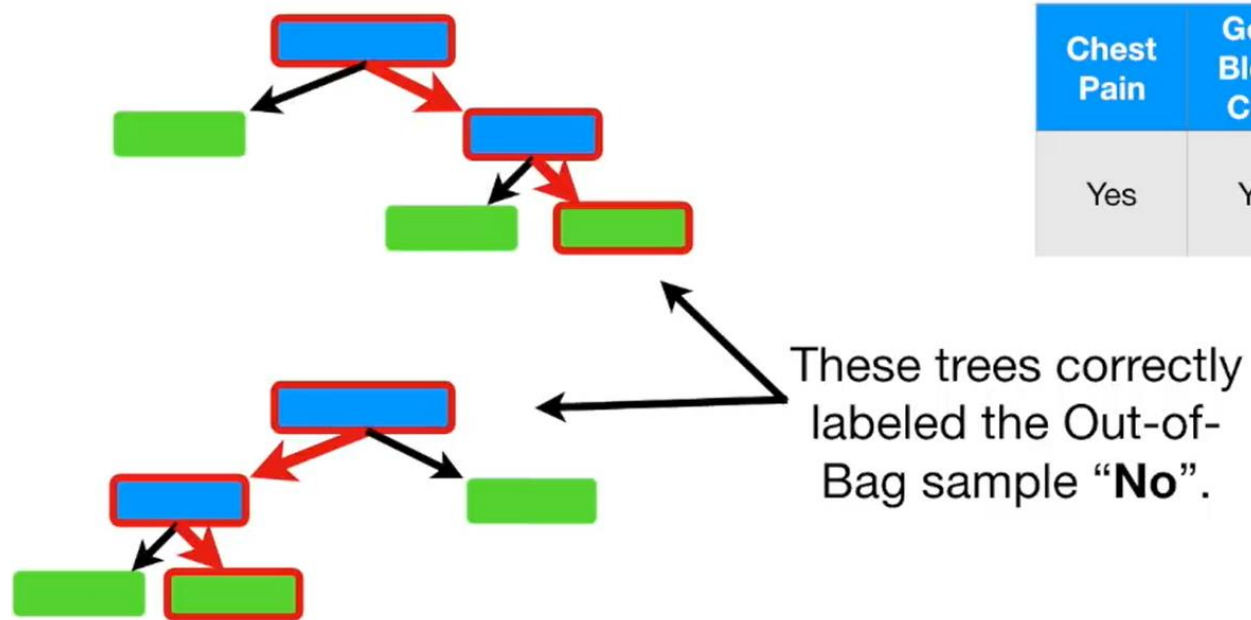
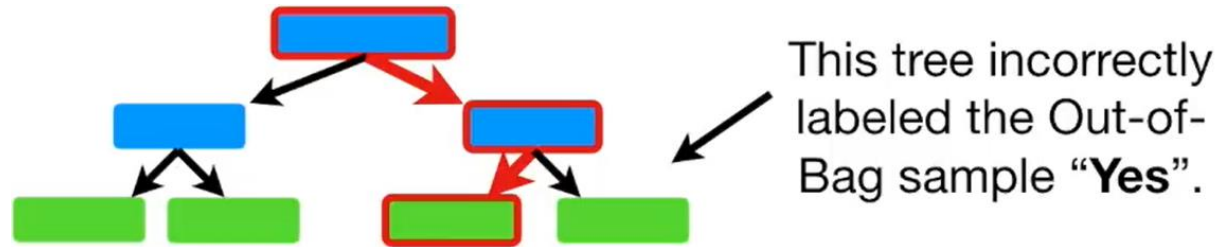


Then we run this Out-Of-Bag sample through all of the other trees that were built without it...

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

→ RANDOM FOREST

Classification of Out-of-bag sample:



Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

→ RANDOM FOREST

Classification of Out-of-bag sample:

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Since the label with the most votes wins, it is the label that we assign this Out-of-Bag sample.

In this case, the Out-of-Bag sample is correctly labeled by the Random Forest.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No

➔ RANDOM FOREST

Out-of-bag error:

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

etc... etc... etc...

Estimate the accuracy of a Random Forest.

Ultimately, we can measure how accurate our random forest is by the proportion of Out-Of-Bag samples that were correctly classified by the Random Forest.

The proportion of Out-Of-Bag samples that were *incorrectly* classified is the “**Out-Of-Bag Error**”

➔ RANDOM FOREST

Changing the number of variables:

Remember when we built our first tree and we only used 2 variables to make a decision

Now we can compare the Out-Of-Bag error for a random forest built using only 2 variables per step...



...to random forest built using 3 variables per step...



...and we test a bunch of different settings and choose the most accurate random forest.

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

➔ RANDOM FOREST

Changing the number of variables:

- 1) Build a Random Forest
 - 2) Estimate the accuracy of a Random Forest.
- ...change the number of variables used per step...
- 

Do this for a bunch of times and then choose the one that is most accurate.

Typically, we start by using the square of the number of variables and then try a few settings above and below that value.

➔ RANDOM FOREST

Key hyperparameters to tune:

- **n_estimators (Number of Trees)**
 - Defines how many decision trees are built.
 - More trees generally improve performance but increase computation.
 - Rule of thumb: start with 100–500 and increase until accuracy plateaus.
- **max_depth (Maximum Depth of Trees)**
 - Controls how deep each tree can grow.
 - Shallow trees → underfitting; very deep trees → overfitting.
 - Typical tuning range: 5 to 50.
- **min_samples_split**
 - Minimum number of samples needed to split an internal node.
 - Higher values reduce overfitting by forcing trees to be less complex.
- **Min_samples_leaf**
 - Minimum number of samples required in a leaf node.
 - Ensures that leaves contain enough samples, smoothing predictions.

➔ RANDOM FOREST

Key hyperparameters to tune:

- **max_features**
 - Number of features to consider at each split.
 - Options:
 - "sqrt" (default for classification)
 - "log2"
 - Or an integer/float specifying absolute number/percentage.
 - Balances bias–variance tradeoff.
- **bootstrap**
 - Whether bootstrap samples are used to build trees.
 - Default is True; can be set to False for Extremely Randomized Trees.
- **class_weight** (for imbalanced classification)
 - Adjusts weights of classes to deal with imbalance.
 - Options: "balanced", "balanced_subsample", or custom weights

➔ RANDOM FOREST

Key hyperparameters to tune:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 300, 500],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2']
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, n_jobs=-1, scoring='accuracy')
grid.fit(X_train, y_train)
print(grid.best_params_)
```

➔ RANDOM FOREST

Key hyperparameters to tune:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_dist = {
    'n_estimators': randint(100, 1000),
    'max_depth': randint(5, 50),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'max_features': ['sqrt', 'log2']
}

rand_search = RandomizedSearchCV(RandomForestClassifier(), param_distributions=param_dist,
                                n_iter=50, cv=5, n_jobs=-1, scoring='accuracy')
rand_search.fit(X_train, y_train)
print(rand_search.best_params_)
```