Classification Model Evaluation

Evaluating Classifier Accuracy

We start by dividing the data into **Training** and **Testing** sets…

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
…				…

Training Data

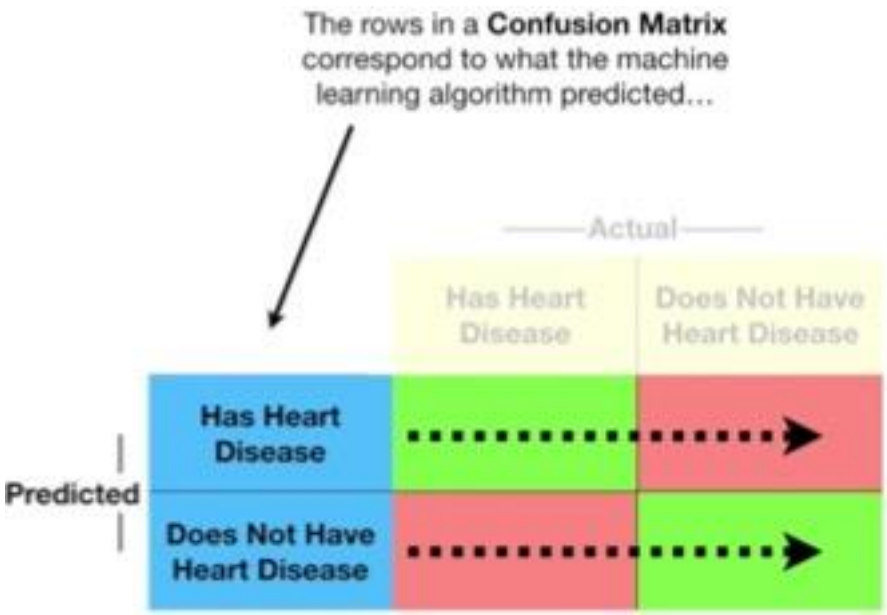
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	210	No
…	…	…	…	…

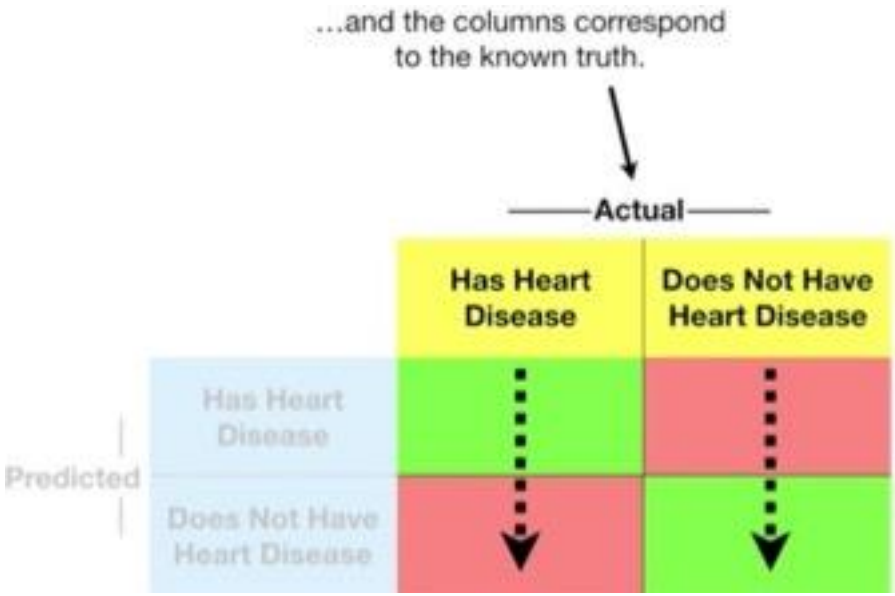
Testing Data

- **What to Evaluate?**
  - Accuracy is measured in terms of error rate
  - Details of errors are shown in a confusion matrix.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease		
	Does Not Have Heart Disease		

Evaluating Classifier Accuracy





Evaluating Classifier Accuracy

In summary, a **Confusion Matrix** tells you what your machine learning algorithm did right…

…and what it did wrong.

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	False Positives
	Does Not Have Heart Disease	False Negatives	True Negatives

Evaluating Classifier Accuracy

Model 1:
True Positives: 142
False Positives: 22
False Negatives: 29
True Negatives: 110

Accuracy: 252/303

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	False Positives
	Does Not Have Heart Disease	False Negatives	True Negatives

	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	142	22
Does Not Have Heart Disease	29	110

Model 1

Model 2:
True Positives: 139
False Positives: 20
False Negatives: 32
True Negatives: 112

Accuracy: 251/303

	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

Model 2

Evaluating Classifier Accuracy

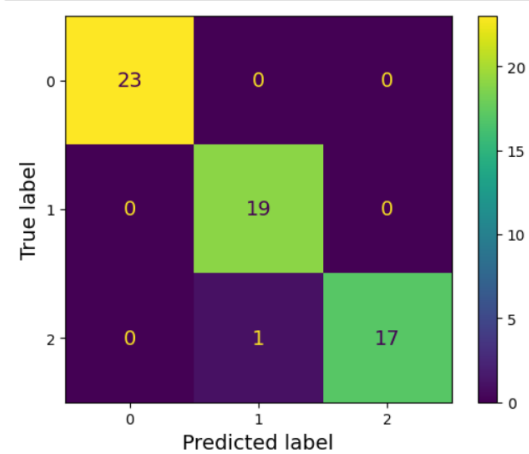
```
y_pred = tree_clf.predict(X_test)
print("Predicted Labels:", y_pred)  #Predicted labels the testing points
print("True Labels:     ", y_test) #True labels
print("Testing Accuracy:", metrics.accuracy_score(y_test, y_pred))  #1 mistake .. 59/60 = 0.983
```

```
Predicted Labels: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0 1 1 2 1 2 1 2 1 0 2 1 0 0 0 1]
True Labels:     [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0 1 2 2 1 2 1 2 1 0 2 1 0 0 0 1]
Testing Accuracy: 0.9833333333333333
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)  # 1 wrong prediction is made .. Accuracy
```

```
array([[23,  0,  0],
       [ 0, 19,  0],
       [ 0,  1, 17]], dtype=int64)
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)  # 1 wrong prediction is made .. Accuracy
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=tree_clf.classes_)
disp.plot()
plt.show()
```



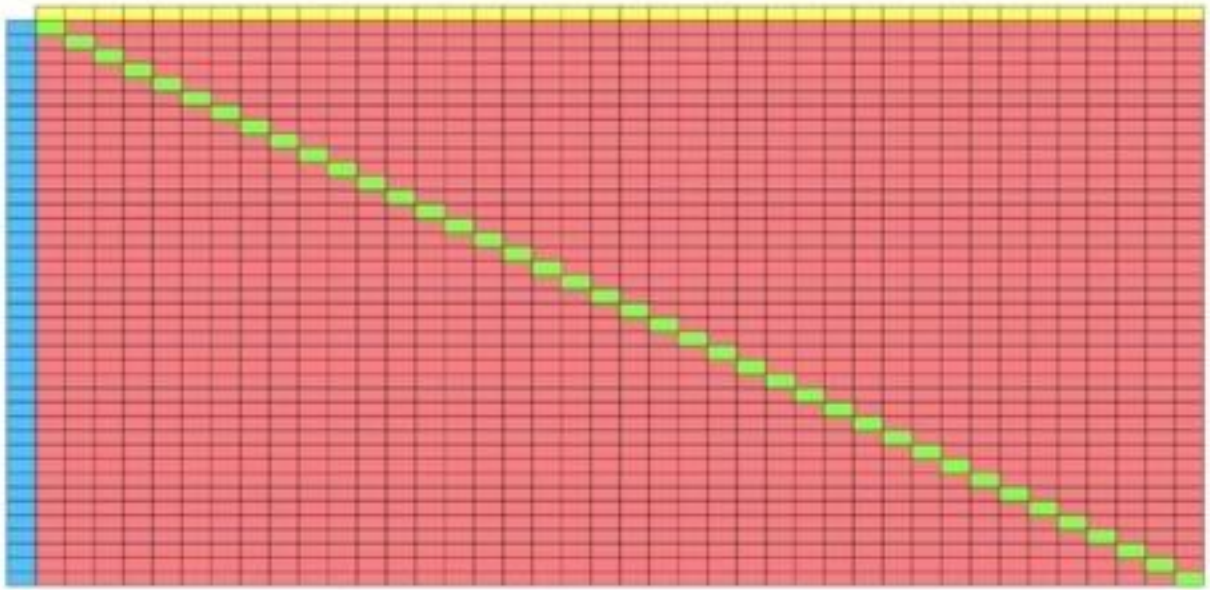
Evaluating Classifier Accuracy

If we had 4 things to choose from, we get a confusion matrix with 4 rows and 4 columns…

	Thing 1	Thing 2	Thing 3	Thing 4
Thing 1				
Thing 2				
Thing 3				
Thing 4				

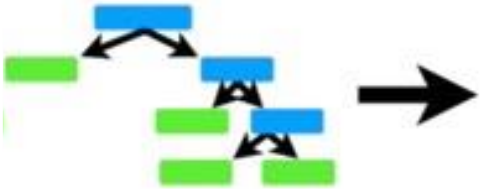
Evaluating Classifier Accuracy

…and if we had 40 things to choose from, we get a confusion matrix with 40 rows and 40 columns.



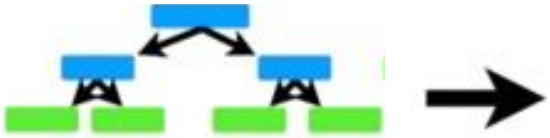
Evaluating Classifier Accuracy

Accuracy= 252/ 303



	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	142	22
Does Not Have Heart Disease	29	110

These two **Confusion Matrices** are very similar and make it hard to choose which machine learning method is a better fit for this data.



	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

Accuracy= 251/ 303

Evaluating Classifier Accuracy

- **What to Evaluate?**
  - Accuracy of model T is measured in terms of error rate

		Actual	
		Has Heart Disease	Does Not Have Heart Disease
Predicted	Has Heart Disease	True Positives	False Positives
	Does Not Have Heart Disease	False Negatives	True Negatives

	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	142	22
Does Not Have Heart Disease	29	110

Model 1

$$ErrorRate(T) = \frac{FP+FN}{TP+FP+TN+FN}$$

$$AccuracyRate(T) = 1 - ErrorRate(T)$$

$$AccuracyRate(T) = \frac{TP+TN}{TP+FP+TN+FN}$$

	Has Heart Disease	Does Not Have Heart Disease
Has Heart Disease	139	20
Does Not Have Heart Disease	32	112

Model 2

Error metrics for skewed classes

Cancer classification example

Train a binary classification model. (cancer / no cancer)

You find that you got 1% error on test set, i.e., 99% correct diagnoses!

	Actual Cancer	Actual No Cancer
Cancer	80	180
No Cancer	20	19720

Accuracy = 19800/20,000 = 99%

But, what if only 0.50% of patients (in train and test sets) have cancer!
A baseline (non-learning) algorithm that always classify patients as having no cancer will be 99.5% accurate…

	Actual Cancer	No Cancer
Cancer	0	0
No Cancer	100	19900

Accuracy = 19900/20,000 = 99.5%

Precision/Recall

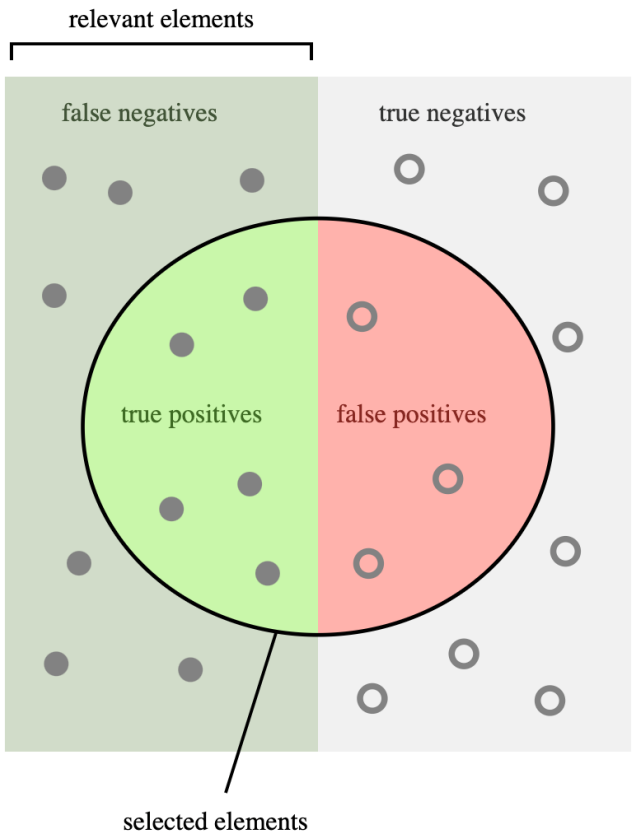
In presence of rare class that we want to detect

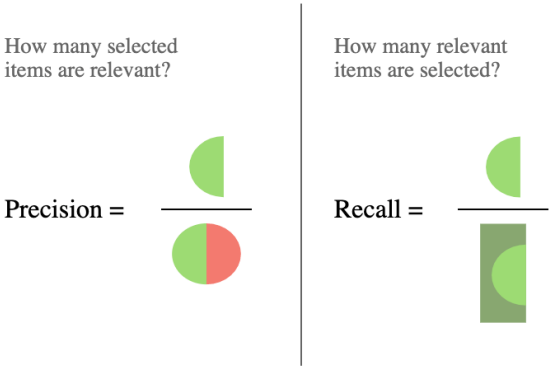
Precision (P):
(for example, of all patients where we predicted that they have cancer, what fraction actually has cancer?)

$$P = \frac{True\ Positives}{Predicted\ Positives} = \frac{TP}{TP + FP}$$

Recall (R):
(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$R = \frac{True\ Positives}{Actual\ Positives} = \frac{TP}{TP + FN}$$





Trading off precision and recall

Suppose we want to predict (cancer) only if very confident.

→ High precision, lower recall.

Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

→ Higher recall, lower precision

F_1 Score (F score)

How to compare precision/recall numbers?

	Precision(P)	Recall (R)	~~Average~~	F_1 Score
Algorithm 1	0.5	0.4	~~0.45~~	0.444
Algorithm 2	0.7	0.1	~~0.4~~	0.175
Algorithm 3	0.02	1.0	~~0.51~~	0.0392

← Predicts y=1 all the time!

Average: $\cancel{\frac{P+R}{2}}$

$$F_1 \text{ Score} = 2\frac{PR}{P+R}$$ (harmonic mean)

- If p=0 or R=0 → F-score = 0
- If p=1 and R=1 → F-score = 1

Evaluating Classifier Accuracy

- **How to Evaluate?**
  - Normally, a separate independently sampled test set is used to measure accuracy of a classification model
  - Evaluation methods:
    - Holdout Method: divide data set (e.g., 60-40, or 2/3-1/3) as training and test sets

In [6]:
```
#Take 40% of the data as testing and the remaining 60% as training
X_train, X_test, y_train, y_test = train_test_split(X_iris, y_iris, test_size=0.4, random_state=42)
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()
print("Training Data", X_train.shape)
print("Testing Data", X_test.shape)  #60 points for testing
```

```
Training Data (90, 2)
Testing Data (60, 2)
```

Decision tree trained on X_train

In [7]:
```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
# Accuracy on the training dataset using the score method
print("Training Accuracy:", tree_clf.score(X_train, y_train))
```

```
Training Accuracy: 0.9888888888888889
```

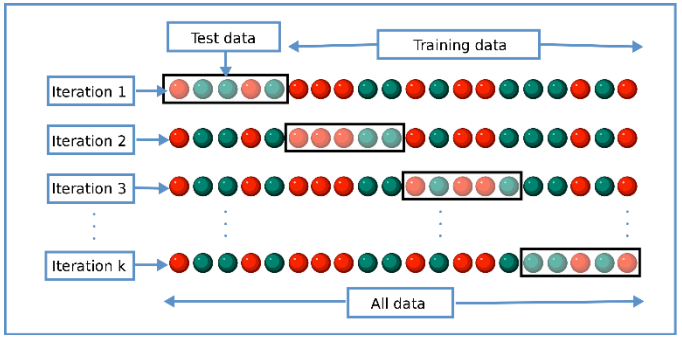
Making Prediction

In [8]:
```
y_pred = tree_clf.predict(X_test)
print("Predicted Labels:", y_pred)  #Predicted labels the testing points
print("True Labels:     ", y_test) #True labels
print("Testing Accuracy:", metrics.accuracy_score(y_test, y_pred))  #1 mistake .. 59/60 = 0.983
```

```
Predicted Labels: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0 1 1 2 1 2 1 2 1 0 2 1 0 0 0 1]
True Labels:      [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0 1 2 2 1 2 1 2 1 0 2 1 0 0 0 1]
Testing Accuracy: 0.9833333333333333
```

Evaluating Classifier Accuracy

- **How to Evaluate?**
  - Normally, a separate independently sampled test set is used to measure accuracy of a classification model
  - Evaluation methods:
    - Cross Validation: the data set is divided into k equal-size partitions. For each round of decision tree induction, one partition is used for testing and the rest used for training. After k rounds, the average error rate is used.
      - Leave-one-out: a special case for small size data sets.



Cross Validation

Every single point is used for testing once

In [12]:
```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_clf, X_iris, y_iris, cv=5)
print(scores)

print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
[0.96666667 0.96666667 0.9        0.96666667 1.        ]
0.96 accuracy with a standard deviation of 0.03
```

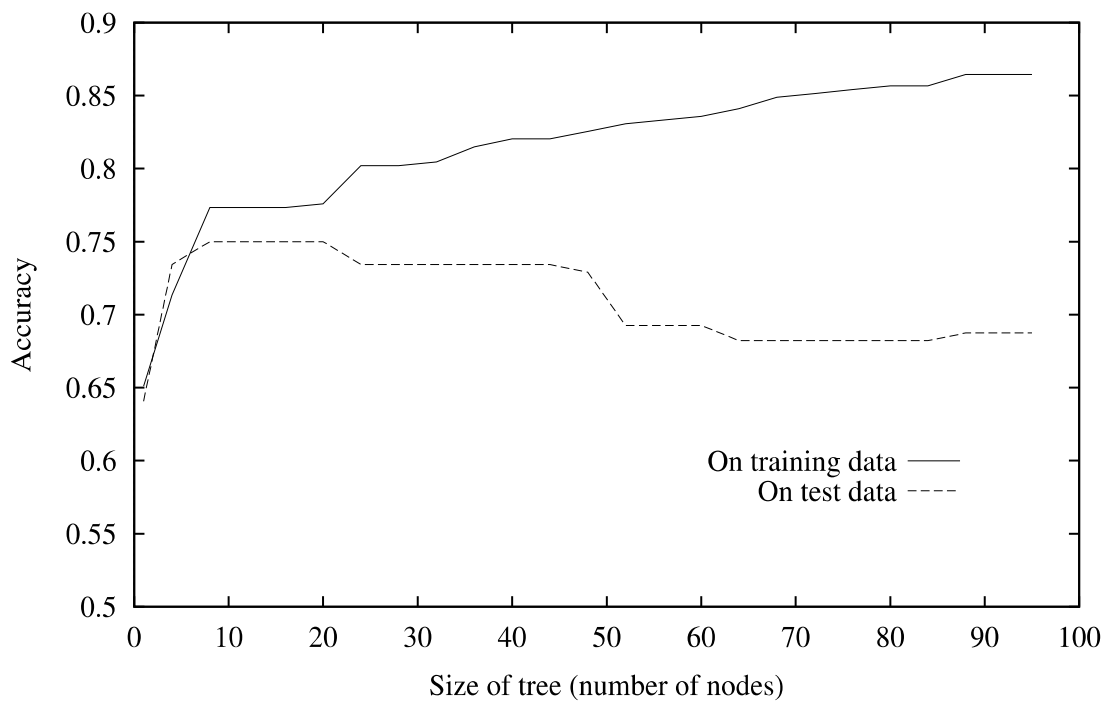
In [18]:
```
#50 cross validation. Each time, train on 147 and test on 3 samples
scores = cross_val_score(tree_clf, X_iris, y_iris, cv=50)
print(scores.round(2))

print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))
```

```
[1.   1.   1.   1.   1.   1.   0.67 1.   1.   1.   1.   1.   1.   1.
 1.   1.   1.   1.   1.   0.67 0.67 1.   1.   1.   1.   1.   0.67 0.67
 1.   0.67 1.   1.   1.   0.67 1.   1.   1.   1.   0.67 1.   1.   1.
 1.   1.   1.   1.   1.   1.   1.   1.  ]
0.95 accuracy with a standard deviation of 0.12
```

Issues in Decision Tree Learning

- Homogeneous classification
  - All leaves are pure, that is have an entropy of 0
  - Leads to **over-fitting** of the training data

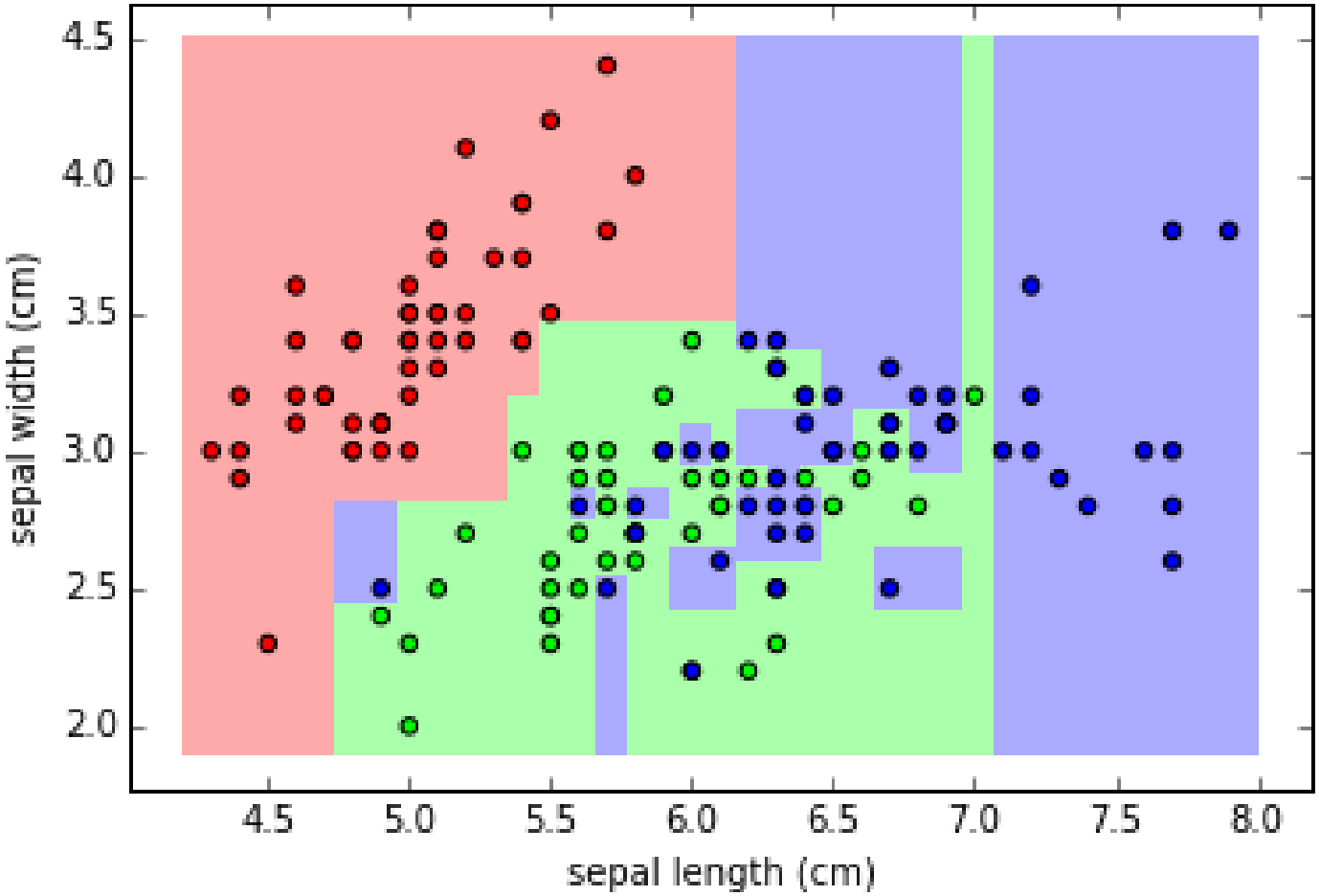


The Problem of Model Overfitting

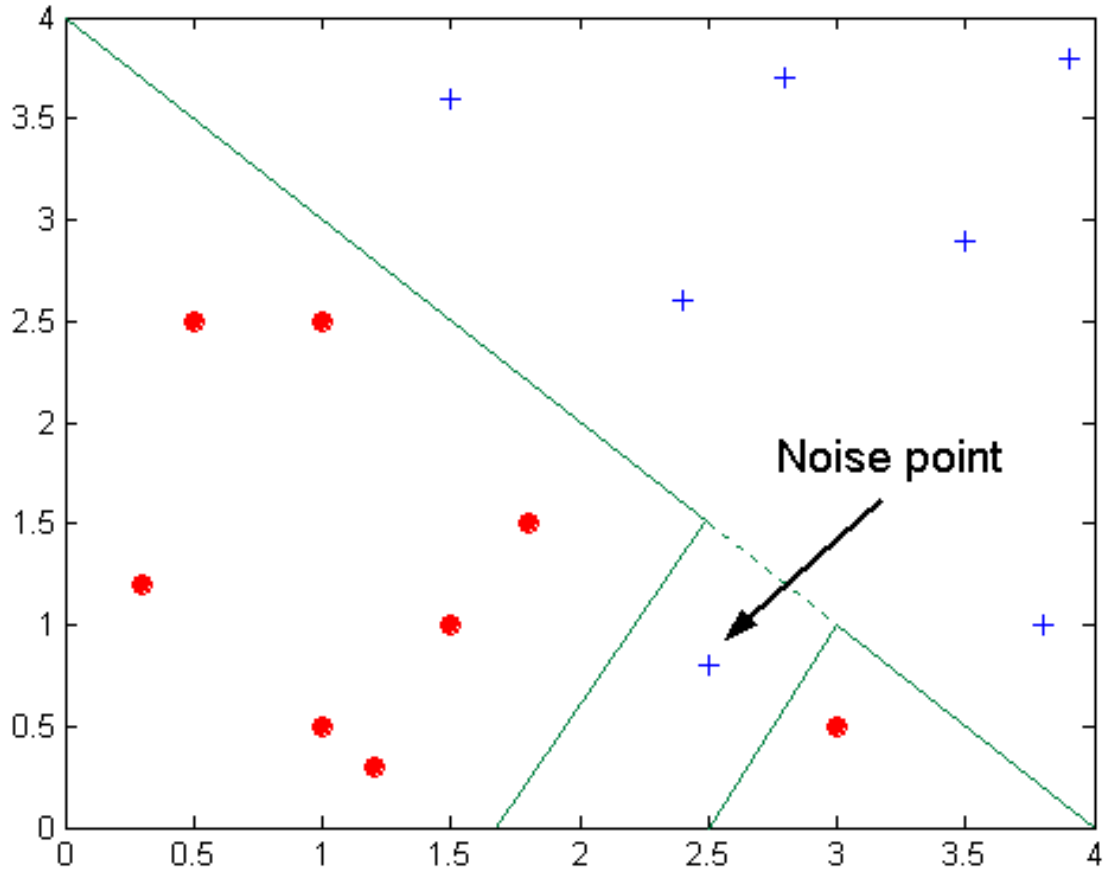
- **Problem Description**
  - All algorithms that build a classification model based on a finite set of training examples tend to have the problem of model overfitting:
    - the model induced from the training examples fits the training examples *too well*.
    - It reflects the specific features of the examples than features of actual data at large.
  - In decision tree induction, a fuller tree with more branches towards the leaf nodes may be built in order to classify those few examples.
  - Using such trees often results in misclassification
  - Causes of the problem include presence of noise data, lack of representative training examples, etc.

Overfitting

If we keep splitting we will be reducing the error, but…

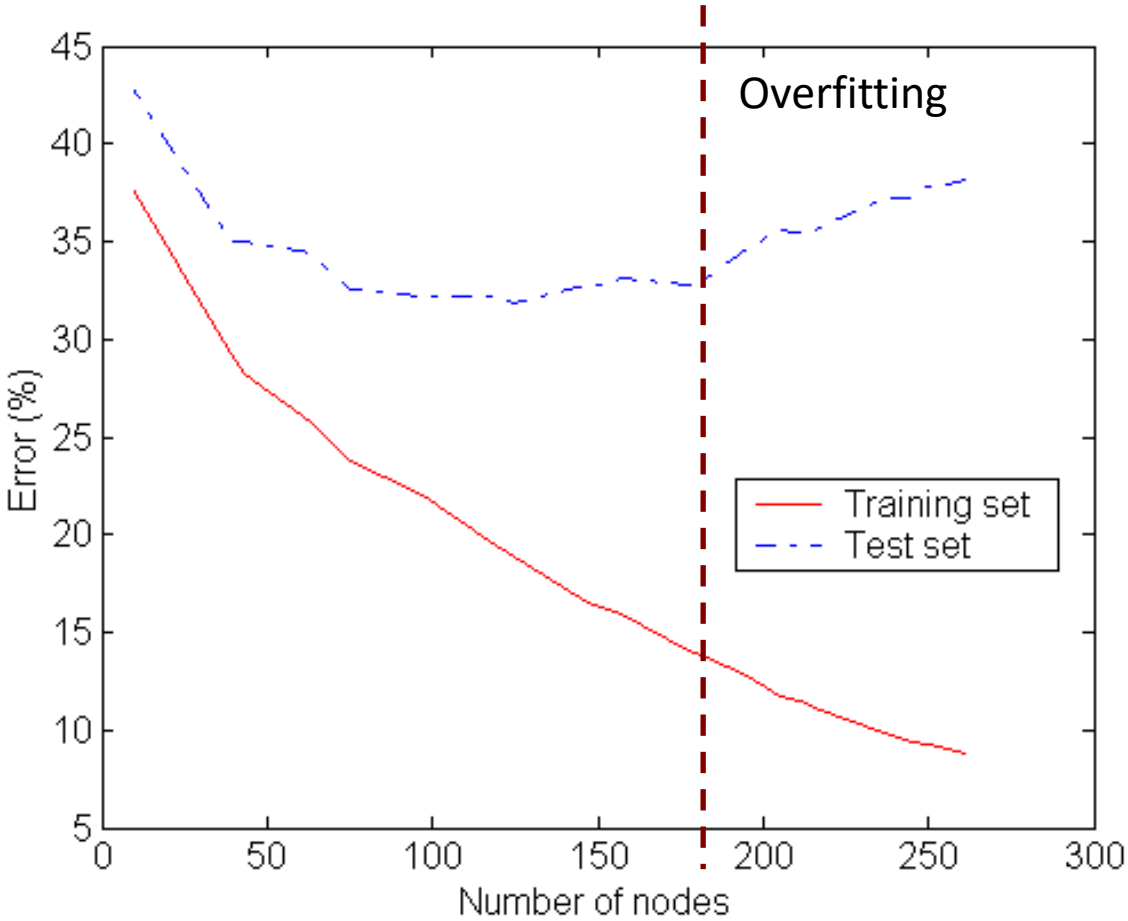


Overfitting due to Noise



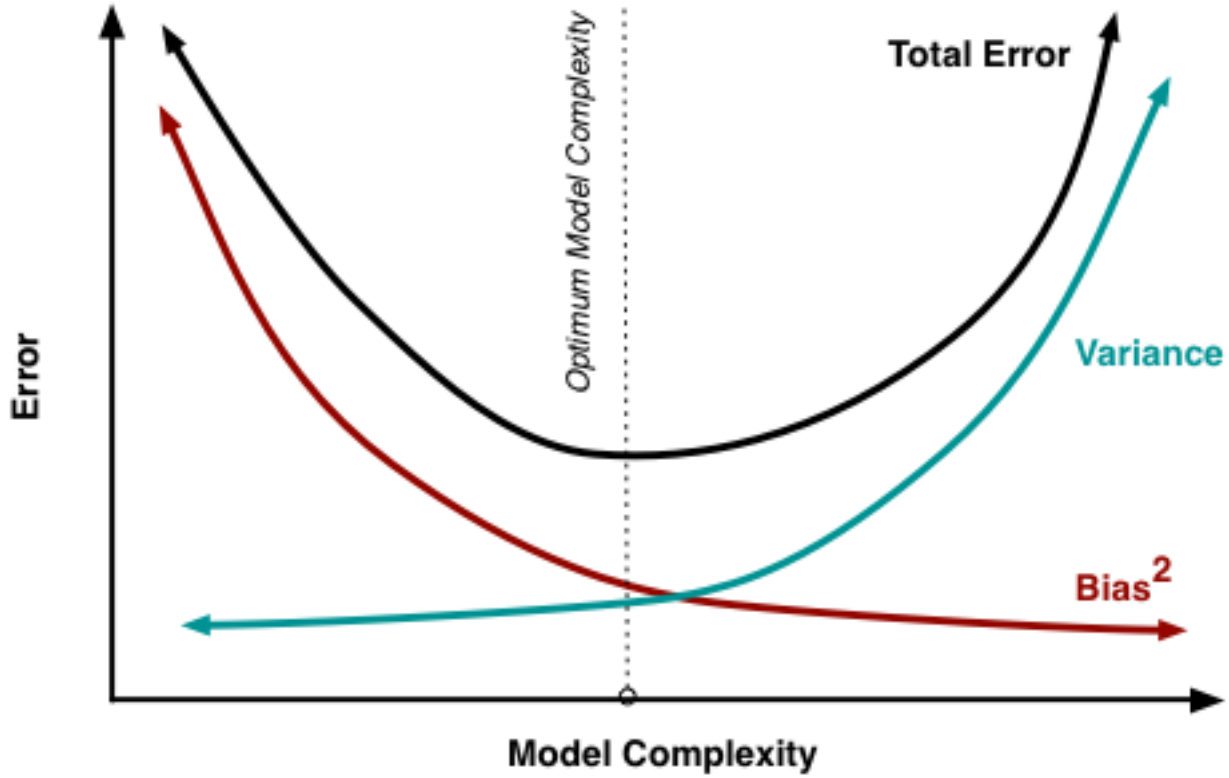
Decision boundary is distorted by noise point

Underfitting (High Bias) vs. Overfitting (High Variance)



Underfitting: when model is too simple, both training and test errors are large

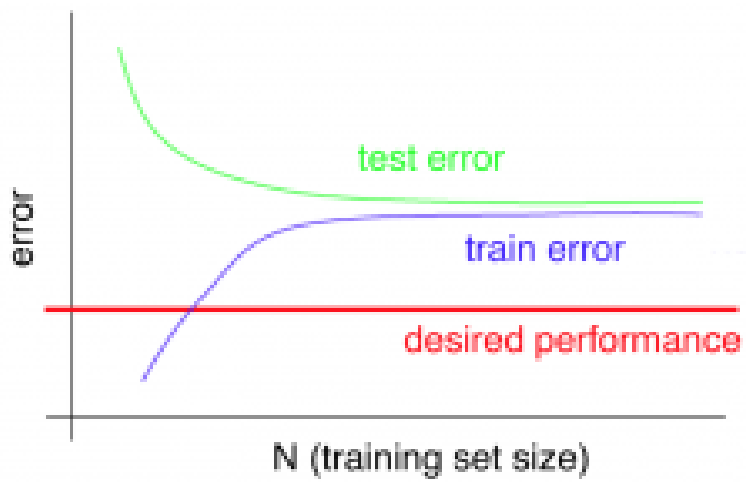
Trade-off: bias-variance errors



More on Bias vs. Variance

If a learning algorithm is suffering from **high bias**, getting more training data will not **(by itself)** help much.

Typical learning curve for high bias (at fixed model complexity)




```python
#Generate data raning from 1000 to 5000. Split into training and testing.
#Build a decision tree of one node only
for i in range(1,6):
    X_moons_train, y_moons_train = make_moons(n_samples=1000*i, noise=0.3, random_state=42)
    X_moons_train, X_moons_test, y_moons_train, y_moons_test = train_test_split(X_moons, y_moons, test_size=0.5)
    tree_clf = DecisionTreeClassifier(max_depth=1,random_state=42)
    tree_clf.fit(X_moons_train, y_moons_train)
    print("Training Dataset", 1000*i/2, print("Testing Dataset", 1000*i/2))
    print("Training Accuracy", tree_clf.score(X_moons_train, y_moons_train))
    print("Testing Accuracy", tree_clf.score(X_moons_test, y_moons_test))
```

```
Testing Dataset 500.0
Training Dataset 500.0 None
Training Accuracy 0.8133333333333334
Testing Accuracy 0.7866666666666666
Testing Dataset 1000.0
Training Dataset 1000.0 None
Training Accuracy 0.8266666666666667
Testing Accuracy 0.84
Testing Dataset 1500.0
Training Dataset 1500.0 None
Training Accuracy 0.8266666666666667
Testing Accuracy 0.8
Testing Dataset 2000.0
Training Dataset 2000.0 None
Training Accuracy 0.8133333333333334
Testing Accuracy 0.8
Testing Dataset 2500.0
Training Dataset 2500.0 None
Training Accuracy 0.8666666666666667
Testing Accuracy 0.8
```

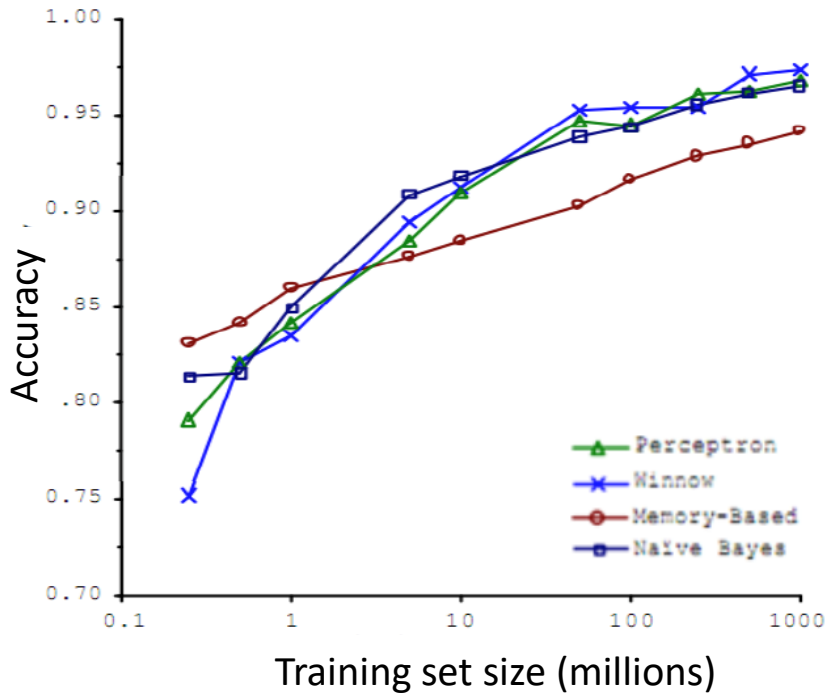
More on Bias vs. Variance

If a learning algorithm is suffering from **high variance**, getting more training data is likely to help.

Typical learning curve for high variance (at fixed model complexity)

Having more data…

Michele Banko and Eric Brill. 2001. **Scaling to very very large corpora for natural language disambiguation**. In Proceedings of the 39th Annual Meeting on Association for Computational Linguistics (ACL '01). Association for Computational Linguistics, USA, 26–33.



"It's not who has the best algorithm that wins. It's who has the most data."

Diagnosing ML Models

- **Simple model** → prone to underfitting (high bias & low variance).
  - It is computationally cheaper.
  - The model fits poorly consistently
  - Fixes:
    - Add features
      - to have sufficient information, e.g., Predict housing price from only size and no other features is hard even for human to do.
    - Build more complex model, e.g., larger trees, ANN with higher number of hidden layers, etc.

- **Complex model** → prone to overfitting (high variance – low bias).
  - It is computationally expensive.
  - Fixes:
    - More training examples
    - Smaller set of features
    - Build less complex models, e.g., shorter trees, ANN with less number of hidden layers, etc.

- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

Pruning: Simplifying a Decision Tree

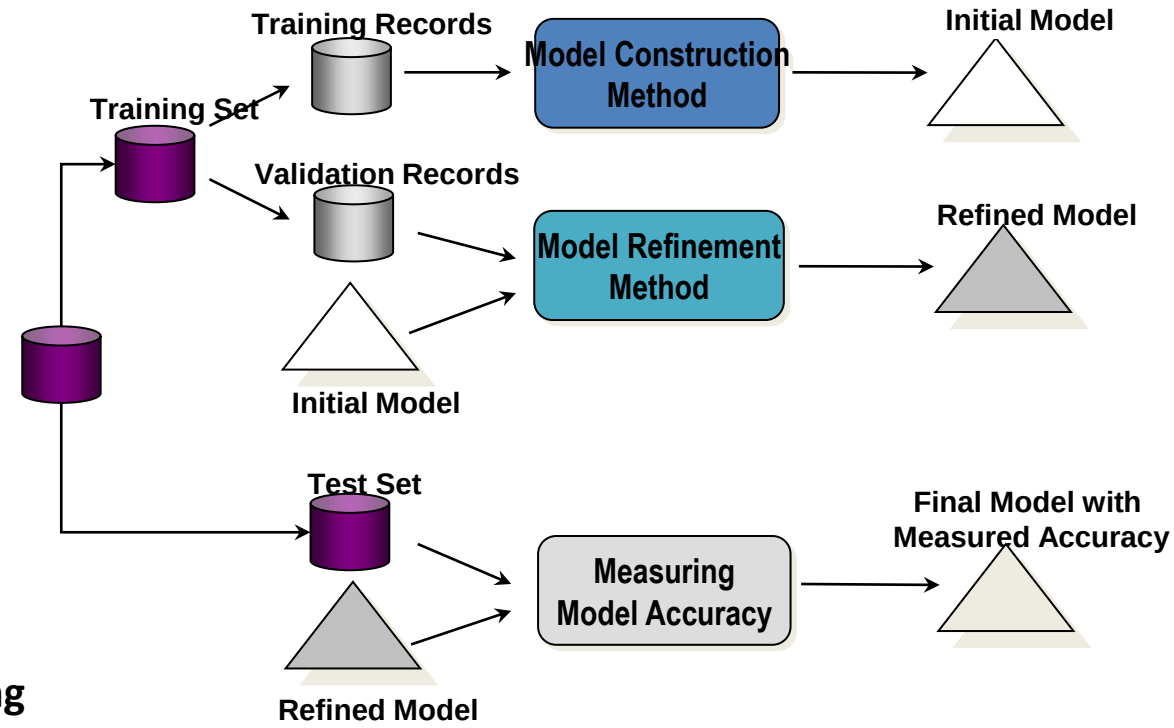
- Tree Pruning: Remove subtrees for better generalization (decrease variance)

- **Approaches**
  - **Pre-pruning** (forward pruning): Early stopping rule: tree construction is halted at some stage
    - Typical stopping conditions for a node:
      - Stop if all instances belong to the same class
      - Stop if all the attribute values are the same
    - Do not split a node if this would result in the goodness measure (e.g., using χ^2 test, Gini, or information gain) falling below a threshold
      - Difficult to choose an appropriate threshold
    - Stop at a specified max. tree size (depth)
  - **Post-pruning** (backward pruning or just pruning): Grow the whole tree then "cut back" certain subtrees and replace them with leaf nodes, making the tree smaller and more robust

- Prepruning is faster, postpruning is more accurate (requires a separate pruning set)

Tree Pruning

- Use an independently sampled validation set to assist tree pruning



- Pruning methods:
  - **reduced error pruning**
  - cost complexity pruning
  - pessimistic pruning
  - …and many others

Tree Pruning

- **Reduced Error Pruning Method**

1. Classify all validation examples using the tree. Note down errors at non-leaf nodes

2. For every non-leaf node, count the number of errors if the subtree is replaced by *the best possible* leaf.

3. Choose the subtree that has the largest reduction of the number of errors to prune, if any of its subtree do not reduce the number of errors. Repeat this step until any further pruning increases the number of errors.

(a) Prune the tree even when there is a zero reduction in errors.
(b) There may be a number of subtrees with the same error reduction. In this case, choose to prune the largest subtree.

Tree Pruning

- **Reduced Error Pruning (Example)**

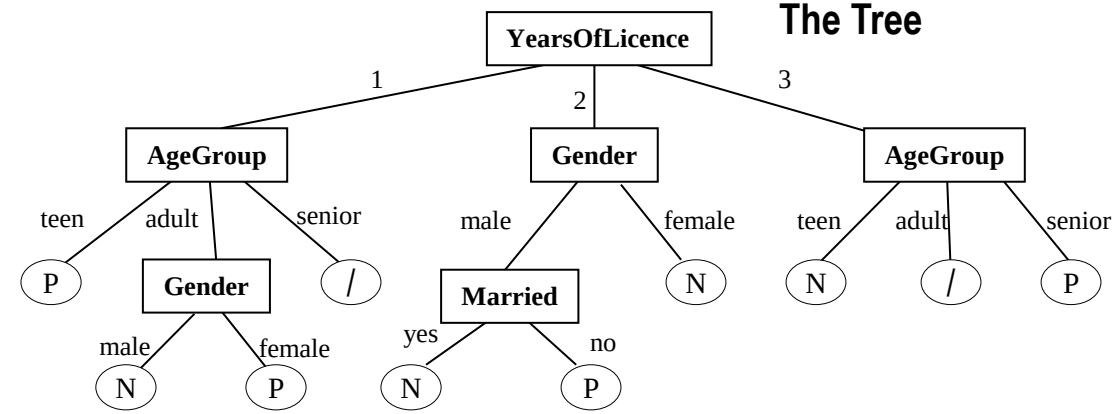
Training set

AgeGroup	Gender	Married	YearsOfLicience	Class
teen	male	no	1	P
teen	male	yes	1	P
teen	female	no	2	N
adult	male	no	2	P
adult	female	yes	2	N
adult	male	yes	1	N
teen	female	yes	2	N
teen	male	no	3	N
adult	female	yes	1	P
senior	male	yes	2	N
senior	female	yes	2	N
senior	male	no	3	P

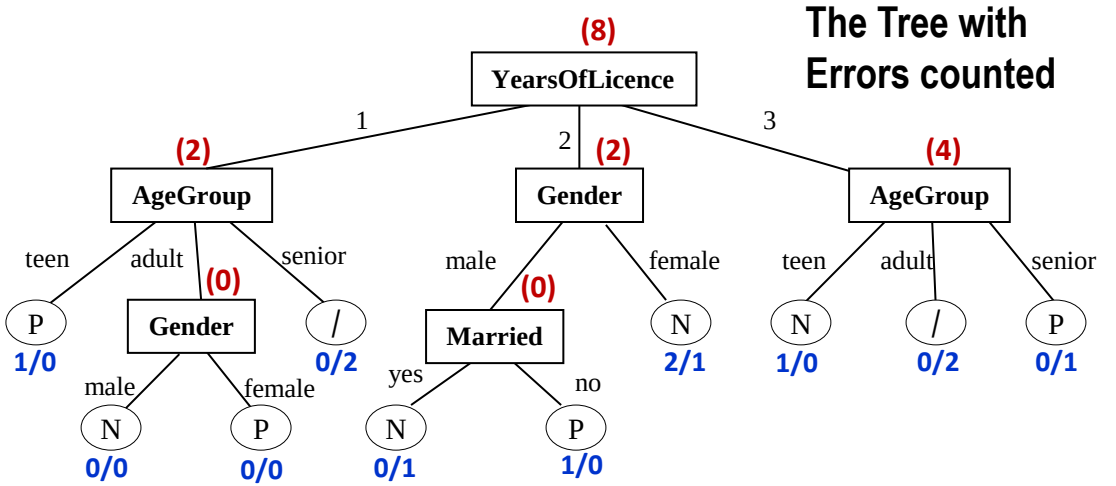
AgeGroup	Gender	Married	YearsOfLicience	Class
teen	male	no	2	P
teen	male	no	3	P
teen	female	no	2	P
teen	female	yes	2	P
adult	female	no	3	N
adult	female	yes	3	N
adult	female	no	2	N
teen	female	yes	1	P
senior	male	yes	1	N
senior	female	yes	1	N
senior	female	yes	3	N
adult	male	yes	2	N

Validation set

The Tree

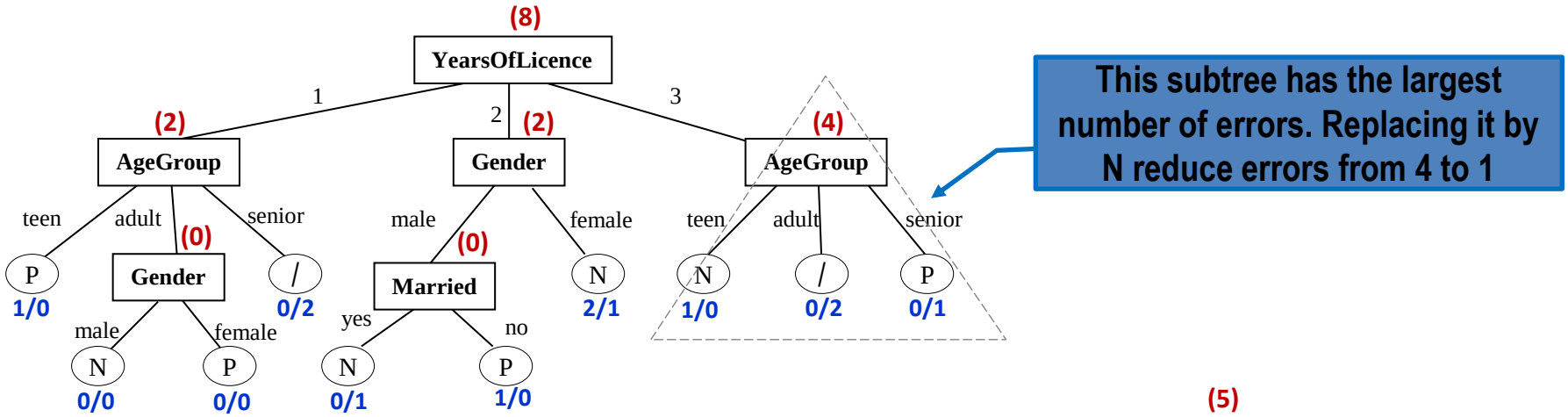


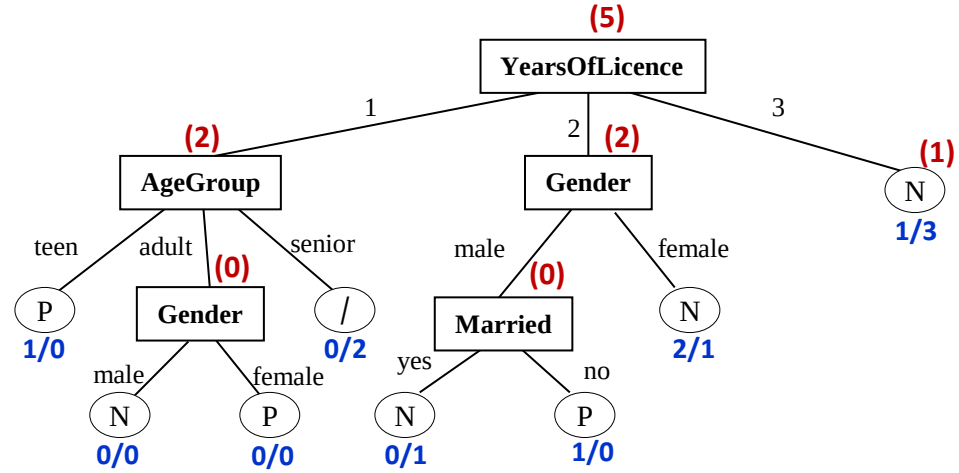
The Tree with Errors counted



Tree Pruning

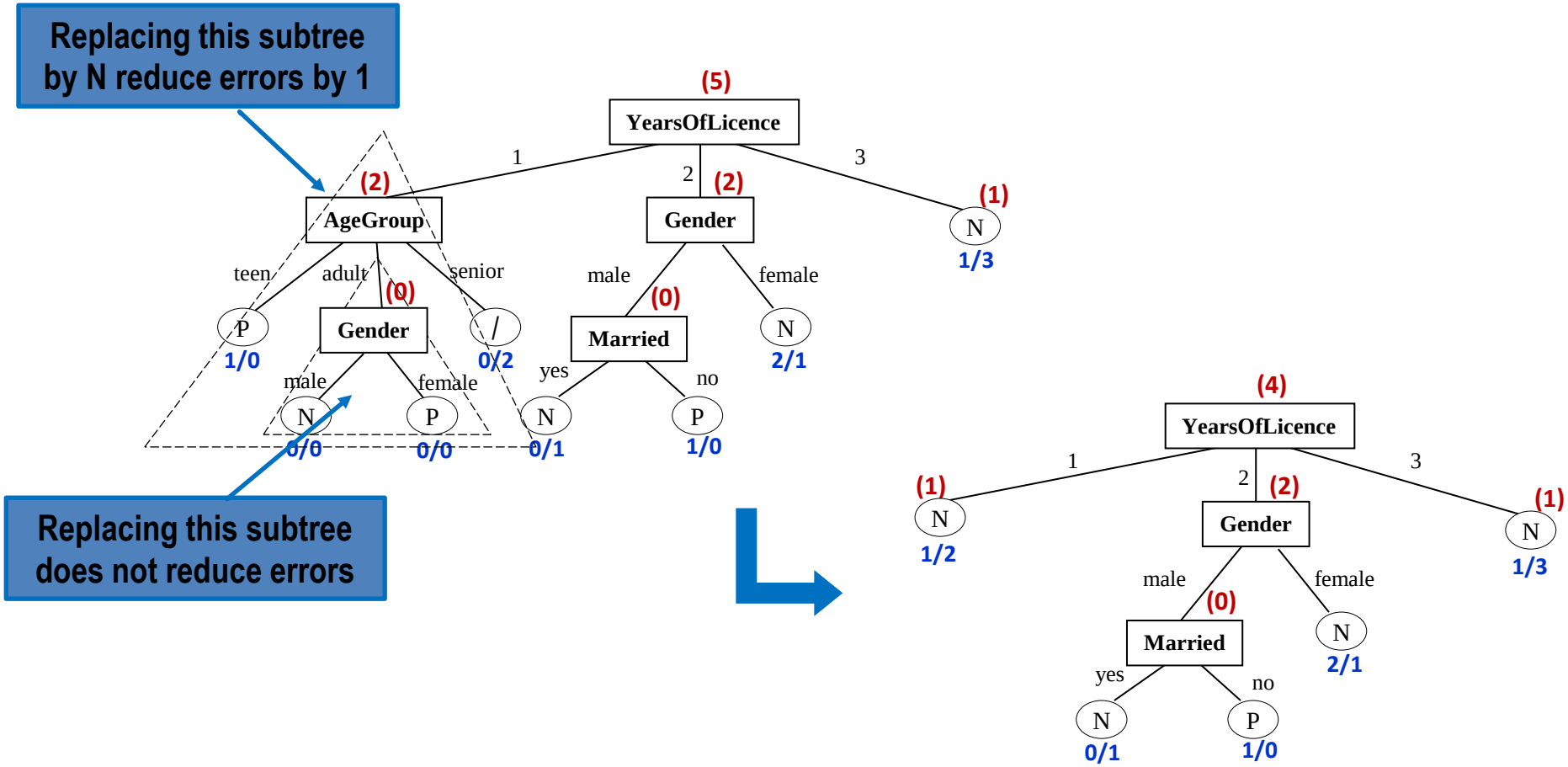
- **Reduced Error Pruning (Example)**





Tree Pruning

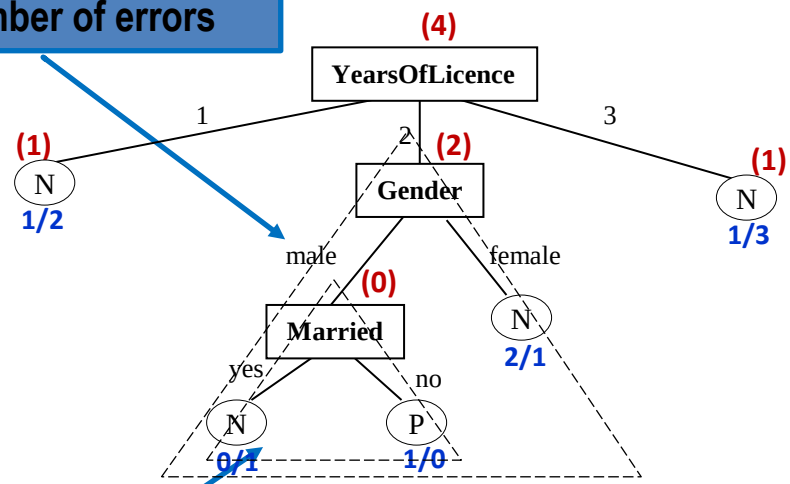
- **Reduced Error Pruning (Example)**



Tree Pruning

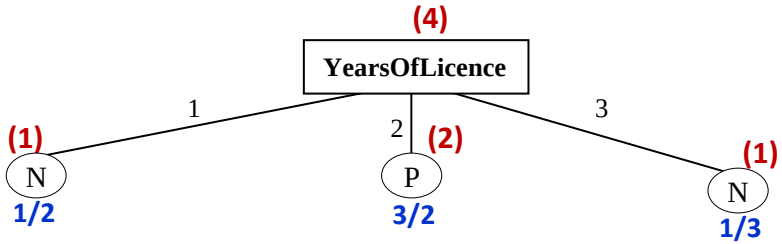
- **Reduced Error Pruning (Example)**

Replacing this subtree by P has the same number of errors



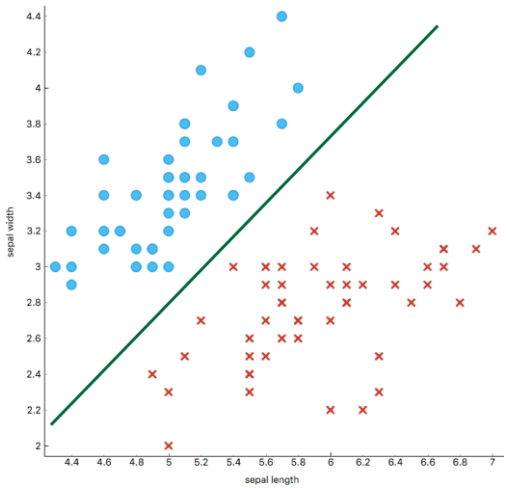
Replacing this subtree increase errors

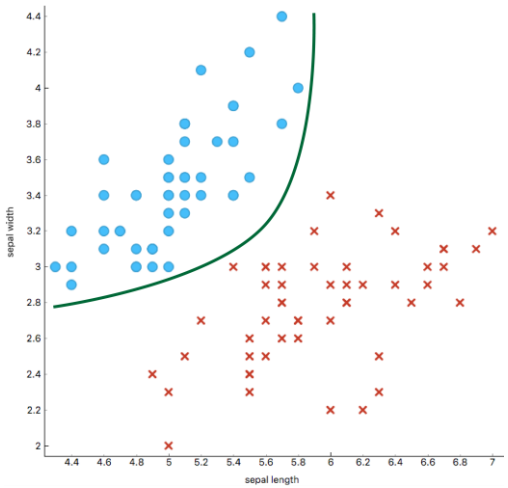
Further pruning the tree increases the number of errors and hence pruning terminates

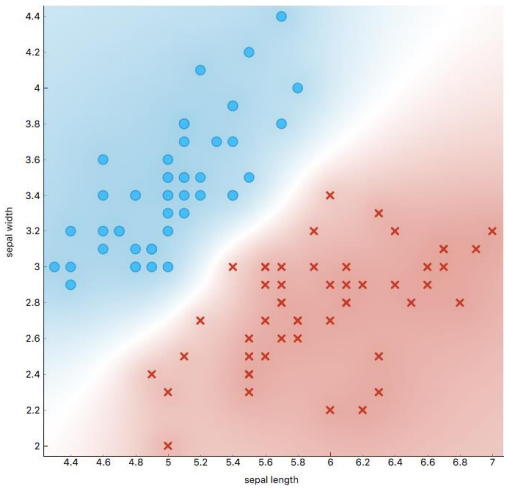


Occam's Razor

- The principle of preferring the **simplest hypothesis** that is (reasonably) consistent with the training data
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - For complex models, there is a greater chance that it was fitted accidentally by errors in data
  - Therefore, one should include model complexity when evaluating a model







Decision Tree Induction Algorithms (Summary)

- A tree is a graphical representation of a set of rules
- Classification Trees are models for predicting or classifying new records
- A decision tree is induced from a set of training examples by using tree induction algorithms such as CHAD, ID3, …
- These algorithms differ mainly on the attribute selection measure used; GINI Index, Information gain, …
- Tree pruning is a way of reducing the effect of model overfitting. There exists a number of tree pruning methods

Decision Tree Induction Algorithms (Summary)

- Pros:
  - Similar to human decision making
  - Resulting tree is often simple, transparent and easy to interpert
  - There exists a common procedural framework for building a tree
  - Tree induction algorithm is relatively computationally inexpensive
  - Produce rules that are easy to interpret & implement
  - Variable selection & reduction is automatic

Decision Tree Induction Algorithms (Summary)

- Cons:
  - Must use discrete (or discretized) attributes
  - The greedy approach does not guarantee best solution
    - Since the process deals with one variable at a time, no way to capture interactions between variables
  - Rectilinear decision boundries → limited expressiveness of the resulting model
    - May not perform well where there is structure in the data that is not well captured by horizontal or vertical splits
  - Non robust. Sensitive to small changes in the data
  - Trees generally do not have the same level of predictive accuracy as some of the other classification approaches
  - Decision trees suffer from a problem of errors propagating throughout a tree
    - Since decision trees work by a series of local decisions, what happens when one of these local decisions is wrong?
      - Every decision from that point on may be wrong
      - We may never return to the correct path of the tree

Decision Tree Induction Algorithms (Summary)

- **Popular algorithms:**
  - CART (Classification and Regression Tree), M5, and M5' Algorithm
    - Produce regression tree
    - Use Gini Index of Impurity as attribute selection measure
  - ID3 Classification Tree Family: C4.5, ID4, ID5, C4.8 and C5.0
    - C4.5: Famous and popular algorithm that addresses many issues with ID3 by using:
      - Information Gain Ratio :to address the induction bias of information gain
      - Pruning (pre/post): to address over-fitting, and help to reduce the impact of noise
      - Can handle numeric (or continuous) valued attributes
  - CHAID Algorithm
    - Older than CART
    - Use Chi-square test (χ^2) as attribute selection measure and to limit tree growth, i.e., splitting stops when purity improvement is not statistically significant
  - SLIQ, SPRINT, C-SEP, PUBLIC, .. others..
- The algorithms mainly differ in attribute selection measures adopted
- Studies show that there are only marginal differences among the attribute selection measures w.r.t. model accuracy

Decision Tree Induction in Practice

- **Key Stages in a ML Project with Decision Tree Induction**
  - Selection of target attribute
  - Inclusion of descriptive attributes
  - Ensuring sufficient coverage of all classes and all descriptive values
  - Data transformation, if necessary, to suit the solution
  - Using a suitable test option
  - Setting solution parameters (minimum number of leaf node examples, tree size, pruned or unpruned)
  - Evaluating model using appropriate measures
- Constructing a classification model is normally an iterative process
- Ideally a simple and robust model of significantly higher accuracy should be chosen

Ensemble Learning: Power of the crowds

Ensemble methods

- A single decision tree does not perform well
- But, it is super fast
- What if we learn multiple trees?

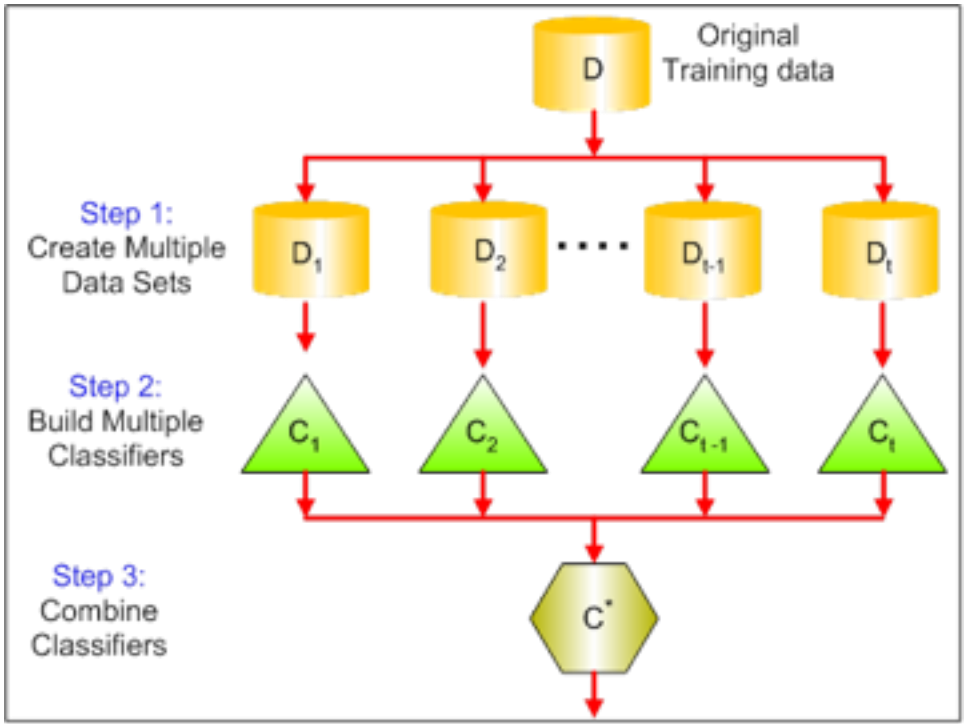
We need to make sure they do not all just learn the same

- Methods:
  - Bagging
  - Boosting
  - Stacking

Bagging

Bootstrap **agg**regat**ing:**

- Multiple realizations of the data
  - Multiple samples (sampling with replacement)
- Train multiple models (hundreds) each with a different data sample
- Calculate predictions multiple times and take the average/majority vote with equal weight
- **Random forest**: an implementation of bagging



- Pros:
  - Improve accuracy over single models; reduces overfitting (variance)
  - Normally uses one type of model; decision trees (no pruning) are popular
  - Easy to parallelize
- Cons:
  - Difficult to interpret the resulting model.

Bagging

One Single Tree

In [27]:

```
#One Single Tree
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
print("Accuracy:",tree_clf.score(X_test, y_test))
```

```
Accuracy: 0.8716666666666667
```

Random Forest

In [28]:

```
randomForest_clf = RandomForestClassifier(n_estimators=100, random_state=42,max_samples=480)
randomForest_clf.fit(X_train, y_train)
print("Accuracy:",randomForest_clf.score(X_test, y_test))
```

```
Accuracy: 0.9166666666666666
```

Boosting

- Incrementally building an ensemble
- Training each new model to emphasize the training instances that previous models mis-classified.
  - Equal weight is given to the sample training data (D_1) at the very starting round.
  - This data (D_1) is then given to a base learner (L_1).
  - The mis-classified instances by L_1 are assigned a weight higher than the correctly classified instances.
  - This boosted data (D_2) is then given to second base learner (L_2) and so on.
  - The results are then combined in the form of weighted voting, based on how well a learner performs.

- **Adaboost (Adaptive Boosting)**: an implementation of boosting
  - AdaBoost (with decision trees "stumps" as the weak learners) is often referred to as the best out-of-the-box classifier

- Pros:
  - Reduces bias, and also variance
  - In some cases, boosting has been shown to yield better accuracy than bagging

- Cons:
  - Tends to be more likely to over-fit the training data
  - Sensitive to noisy data and outliers

Stacking

- Stacking (a.k.a. Stacked Generalization): training a ML model to combine the predictions of several other models trained using the available data
  - Level-0 Models (Base-Models) are typically different (e.g. not all decision trees) and fit on the same dataset (e.g. instead of samples of the training dataset).
  - Level-1 Model (Meta-Model): learns how to best combine the predictions of the base models.
  - Typically yields performance better than any single one of the trained models