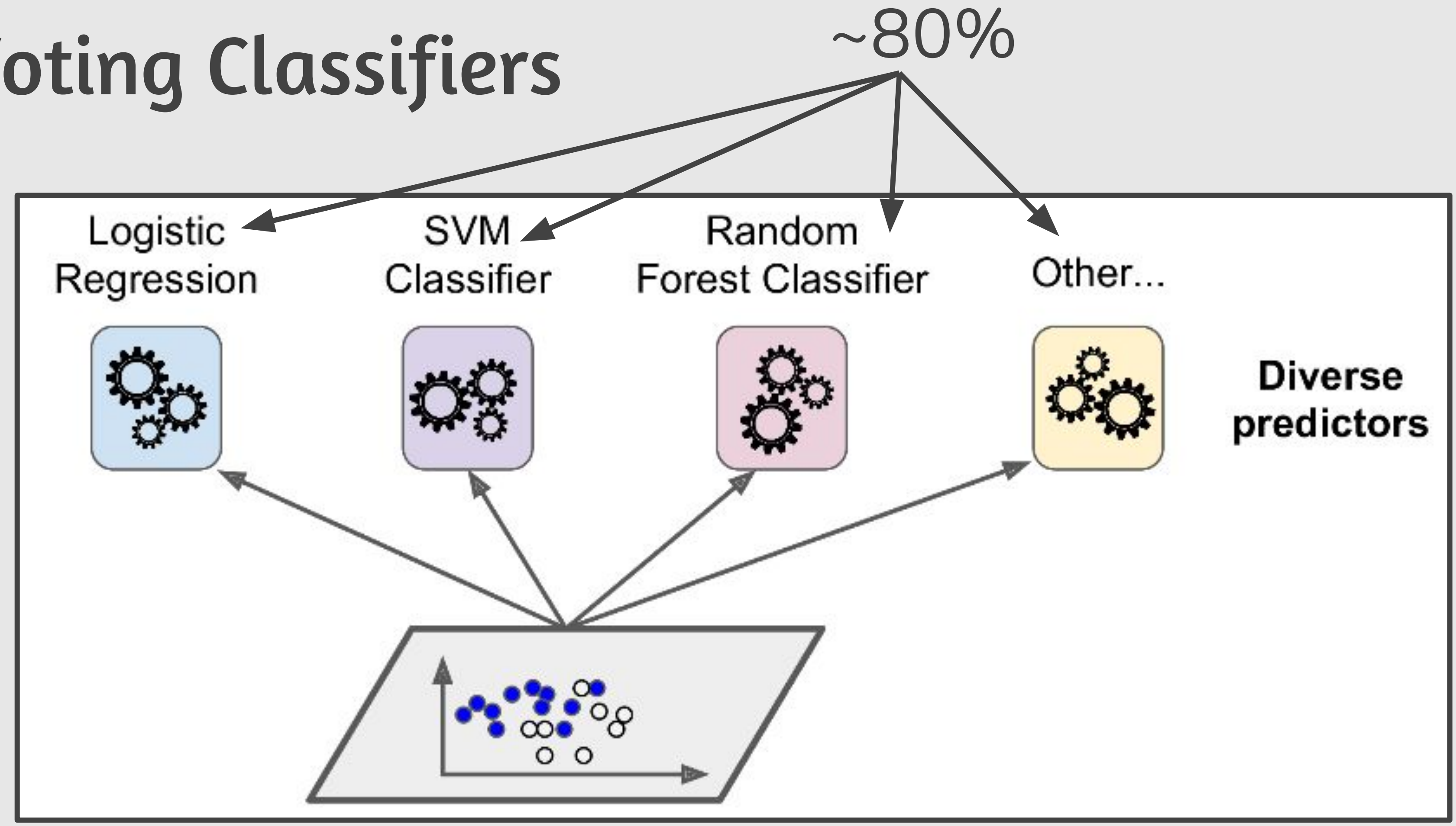
Recall from last time ...

Ensemble Learning

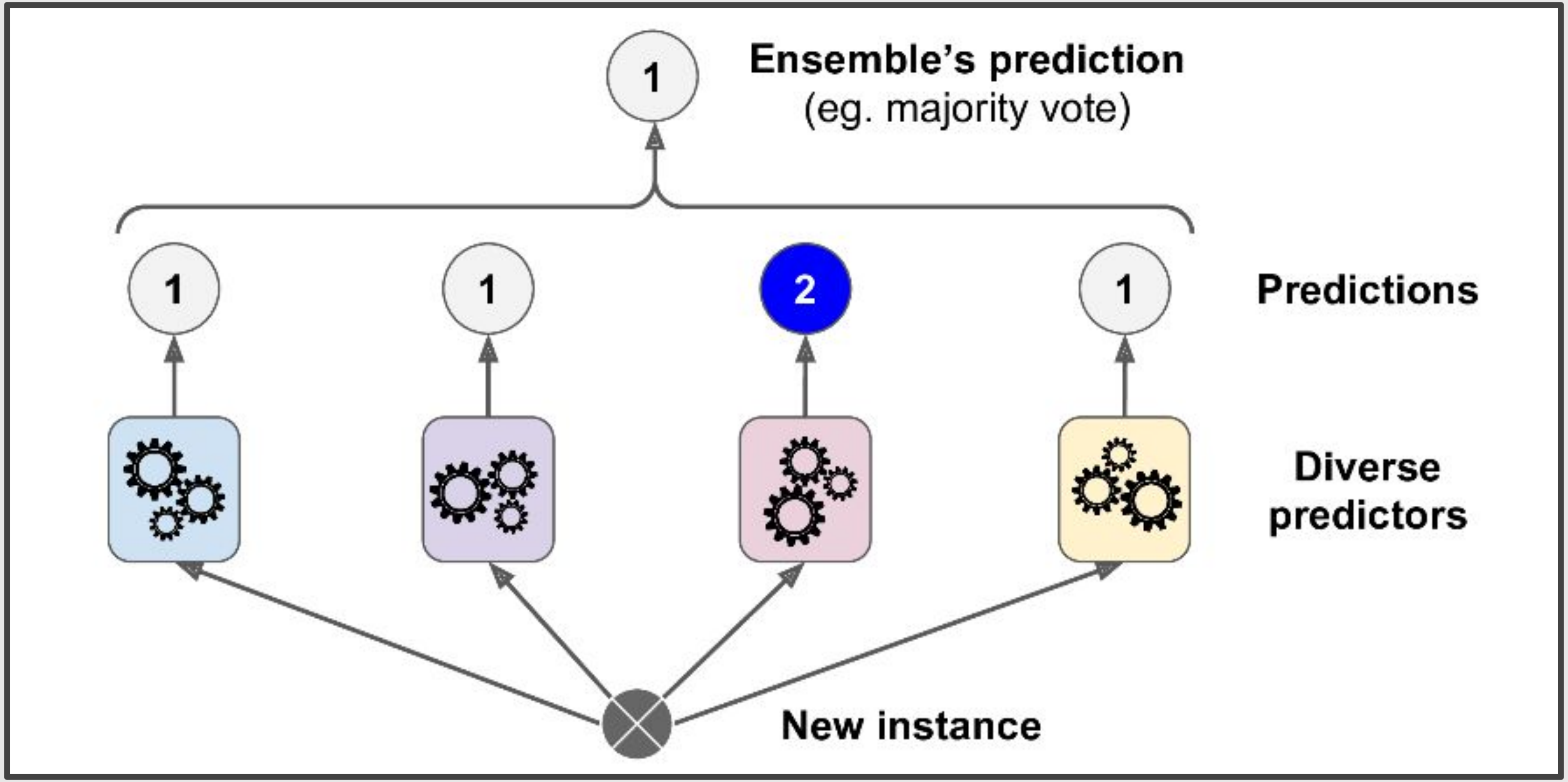
- Multiple learning algorithms **to obtain better predictive performance** than could be obtained from any learning algorithms individually.

Voting Classifiers



Voting Classifiers

Hard/Soft voting classifier



Voting Classifiers

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()
voting_clf = VotingClassifier(
        estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
                    voting='hard'
    )
voting_clf.fit(X_train, y_train)
```

Ensemble Learning

- Types: Bagging (and Pasting), Boosting, and Stacking

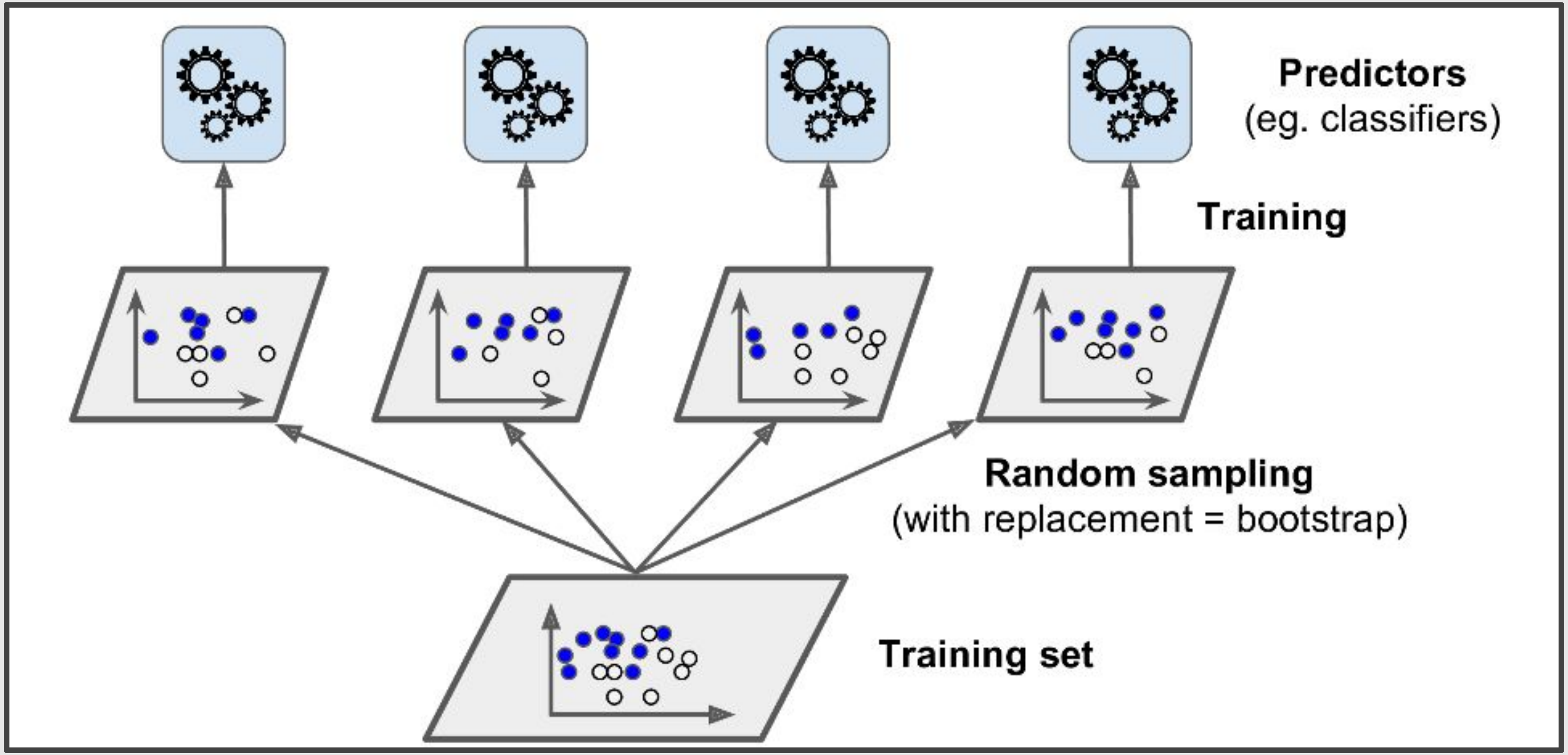
Bagging & Pasting

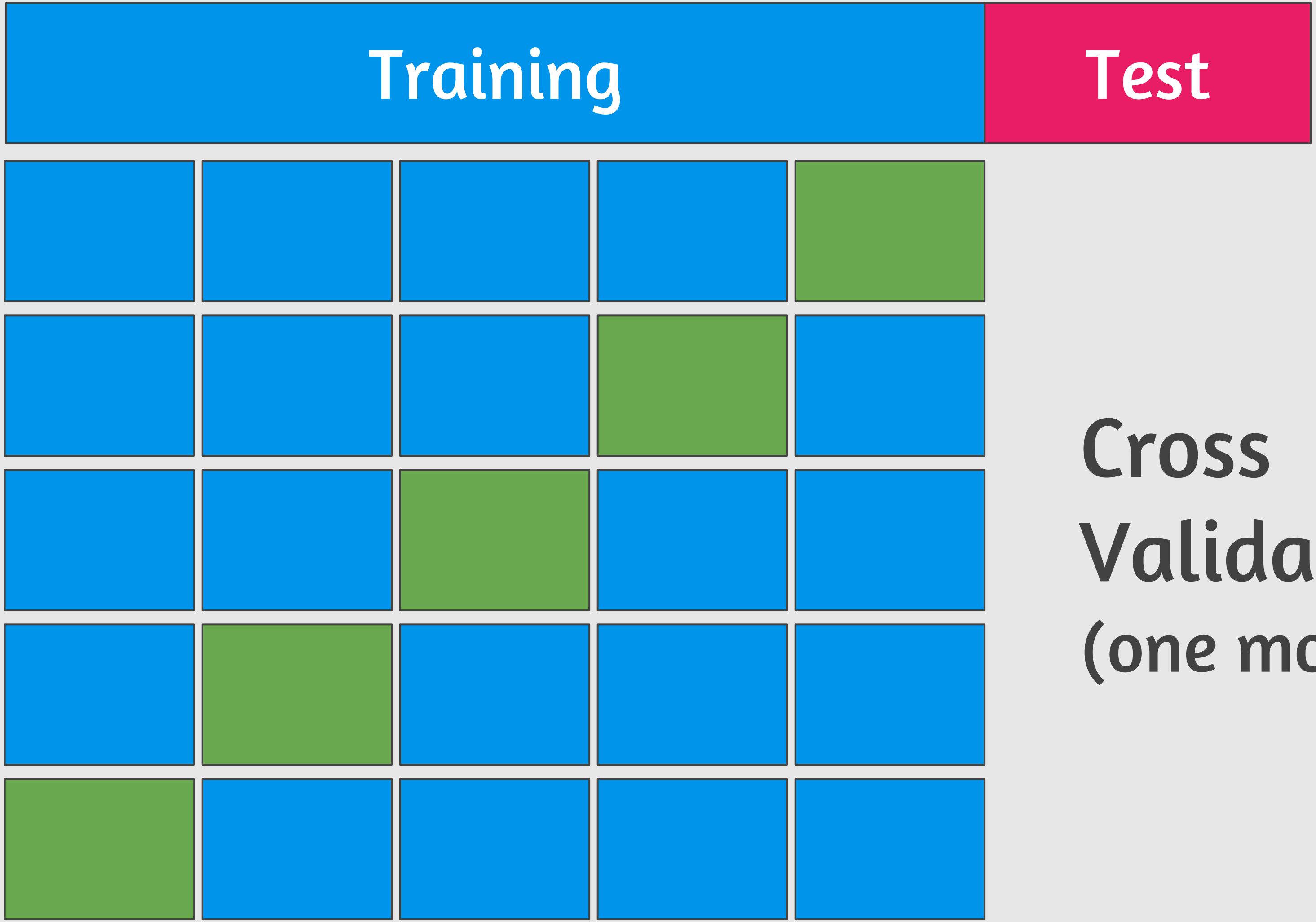
Bagging and Pasting

- Use **the same training algorithm** for every predictor, but to train them on different random subsets of the training set.
- **Bagging** (short for Bootstrap Aggregating): sampling is performed **with** replacement.
- **Pasting**: sampling is performed **without** replacement.

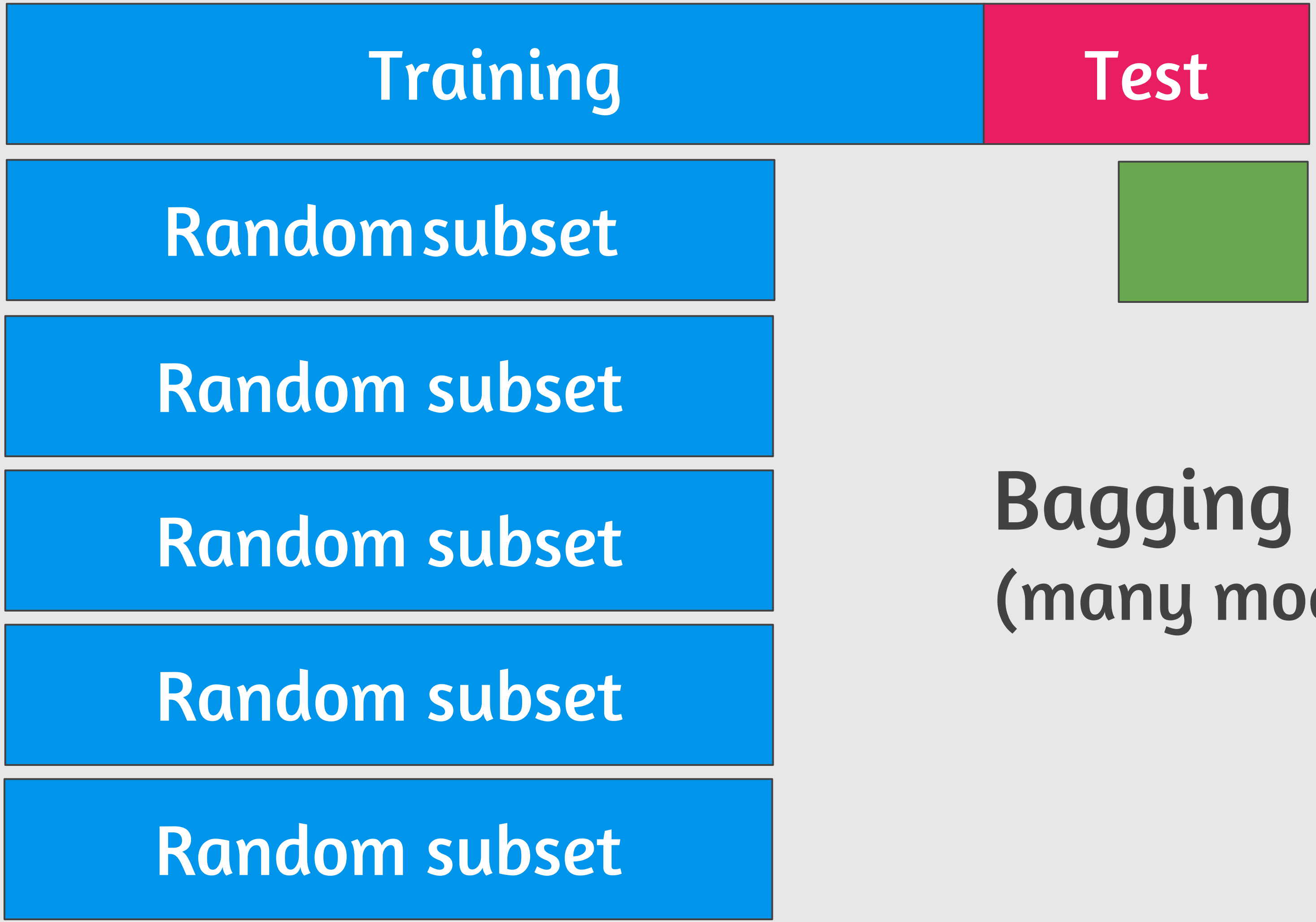
“Bagging Predictors”, L. Breiman (1996): http://goo.gl/o42tml

Bagging and Pasting

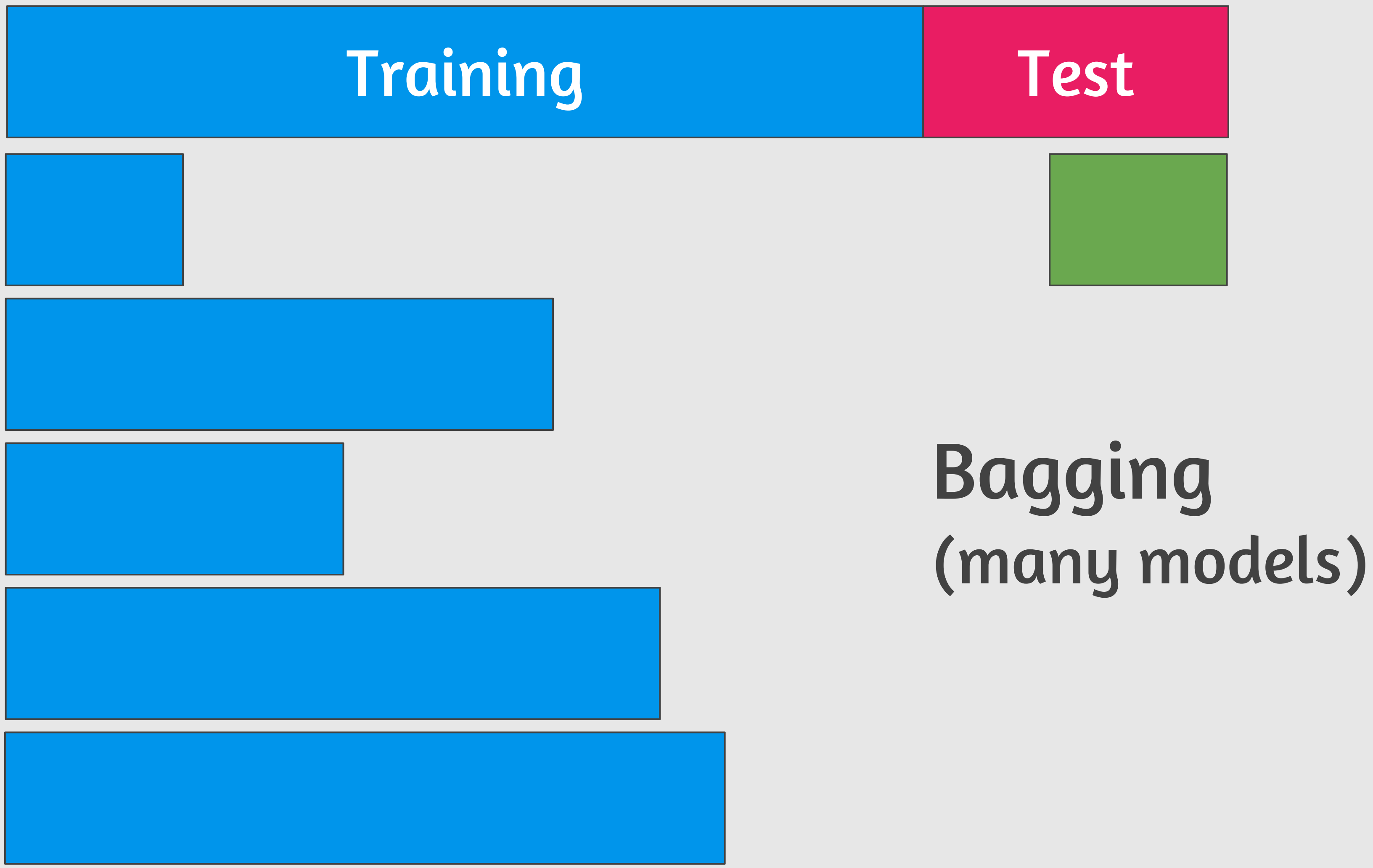




Cross Validation (one model)



Bagging
(many models)


Training
Test
Bagging
(many models)

Bagging and Pasting

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bag_clf = BaggingClassifier(
        DecisionTreeClassifier(), n_estimators=500,
        max_samples=100, bootstrap=True, n_jobs=-1
    )
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Boosting

Boosting

- The general idea of most boosting methods is **to train predictors sequentially**, each trying to correct its predecessor.

- Most popular: AdaBoost and Gradient Boost.

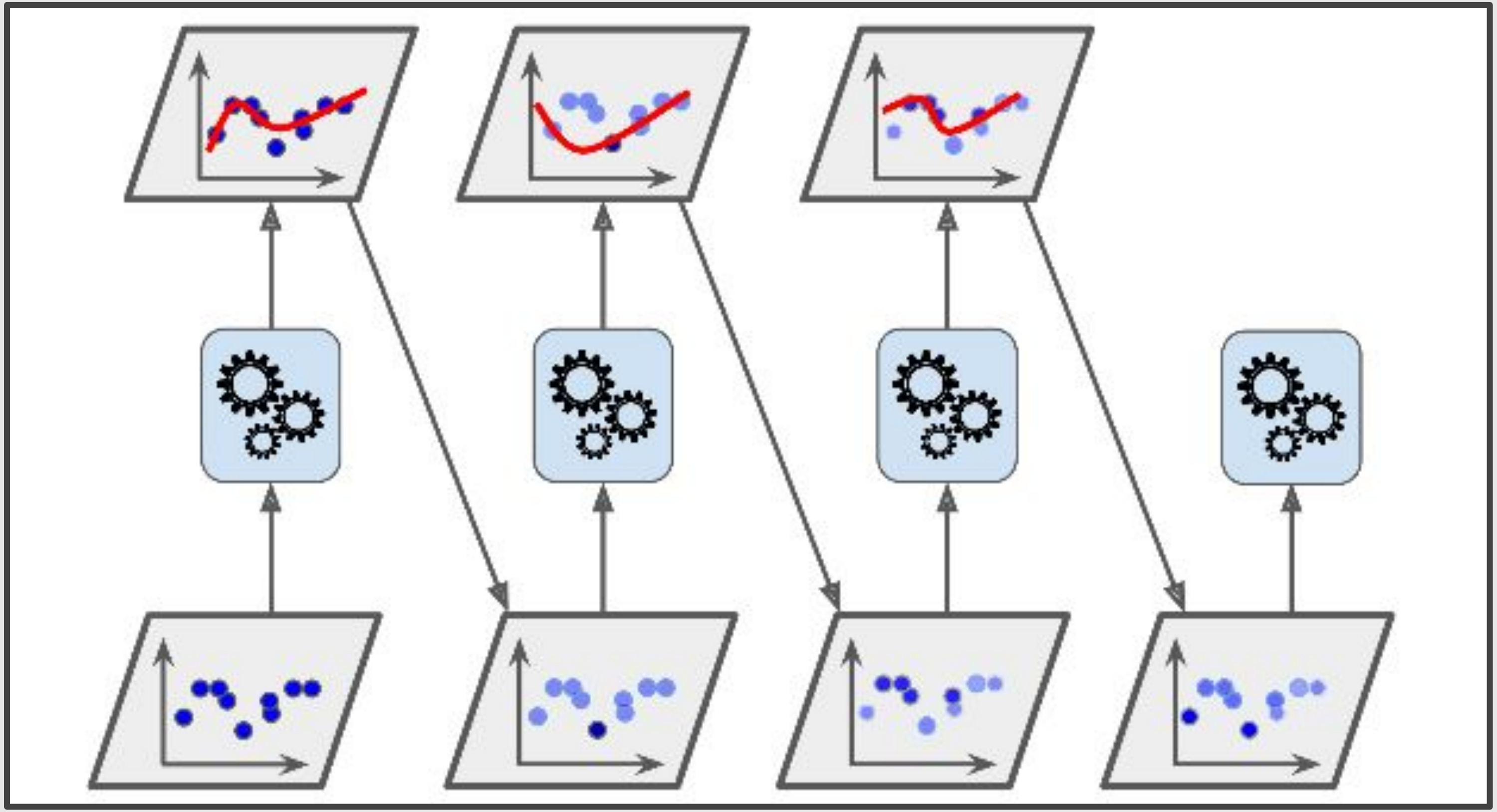
“A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, Y. Freund and R. Schapire (1997): http://goo.gl/OIduRW

AdaBoost [Freund and Schapire, 1997]

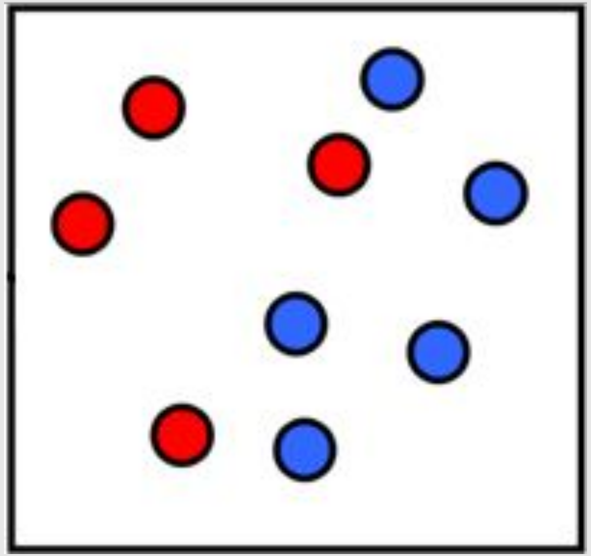
- One way for a new predictor to correct its predecessor is to pay a bit **more attention** to the training instances that **the predecessor underfitted**.

“A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, Y. Freund and R. Schapire (1997): http://goo.gl/OIduRW

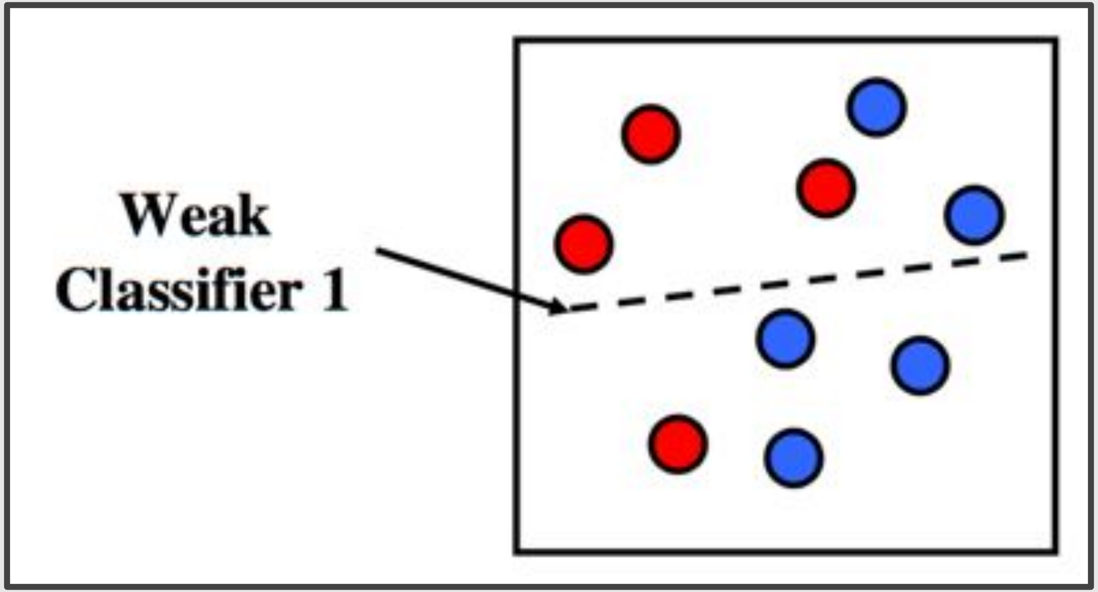
AdaBoost



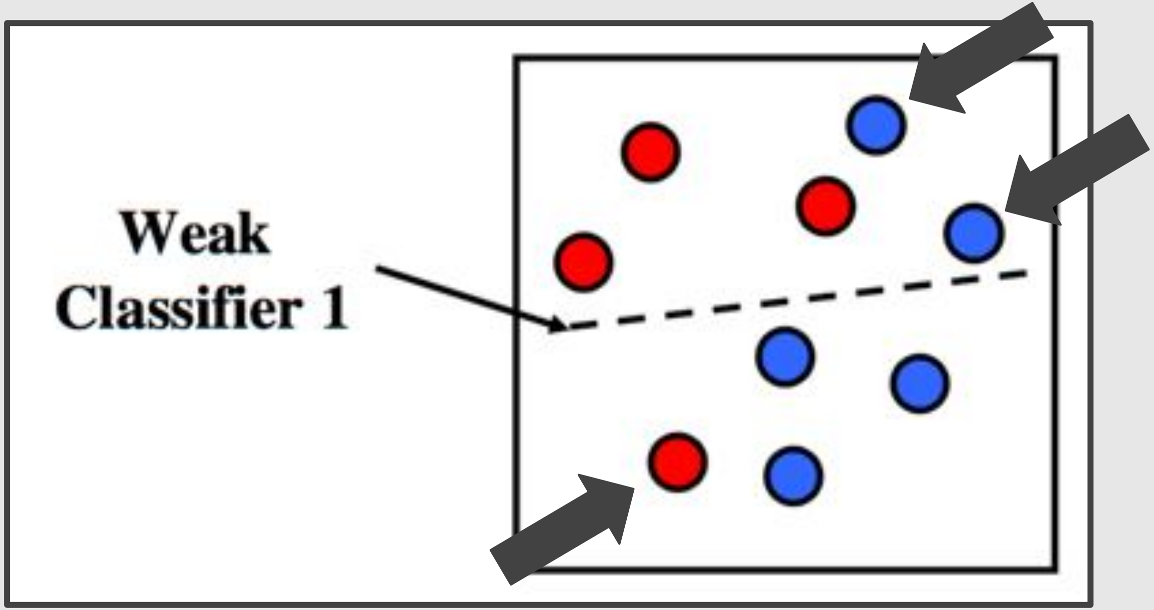
AdaBoost



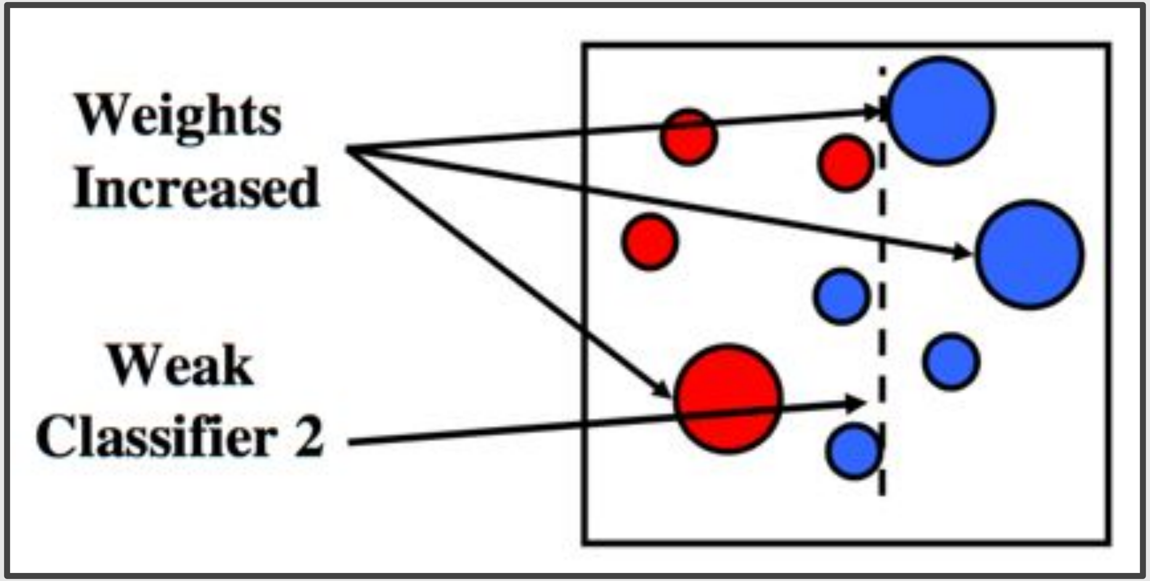
AdaBoost



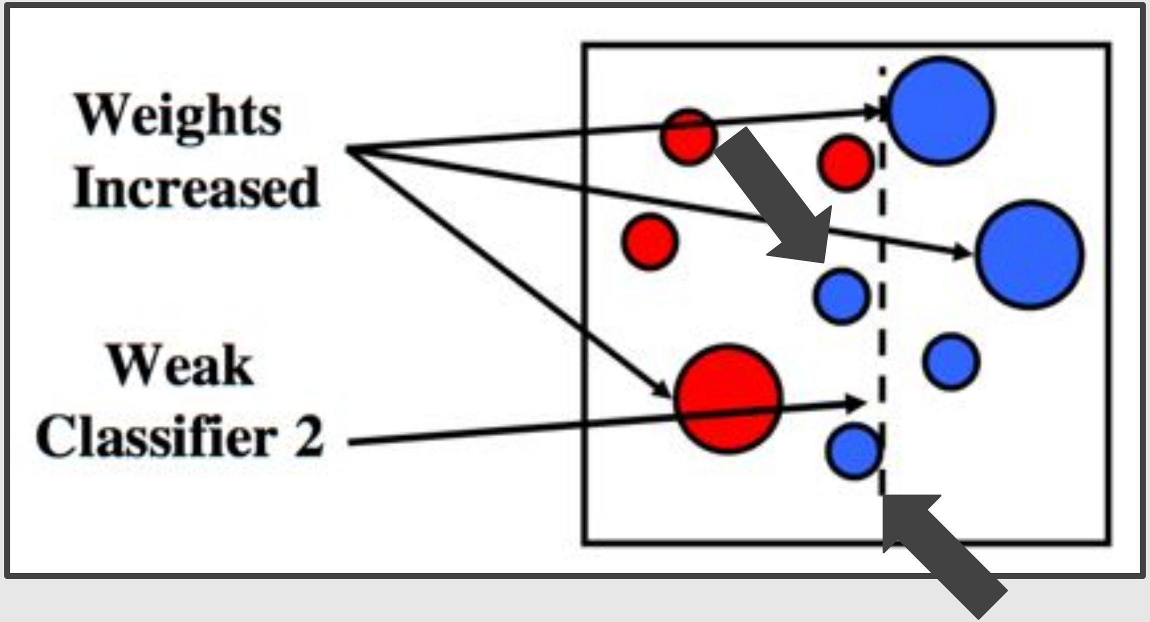
AdaBoost



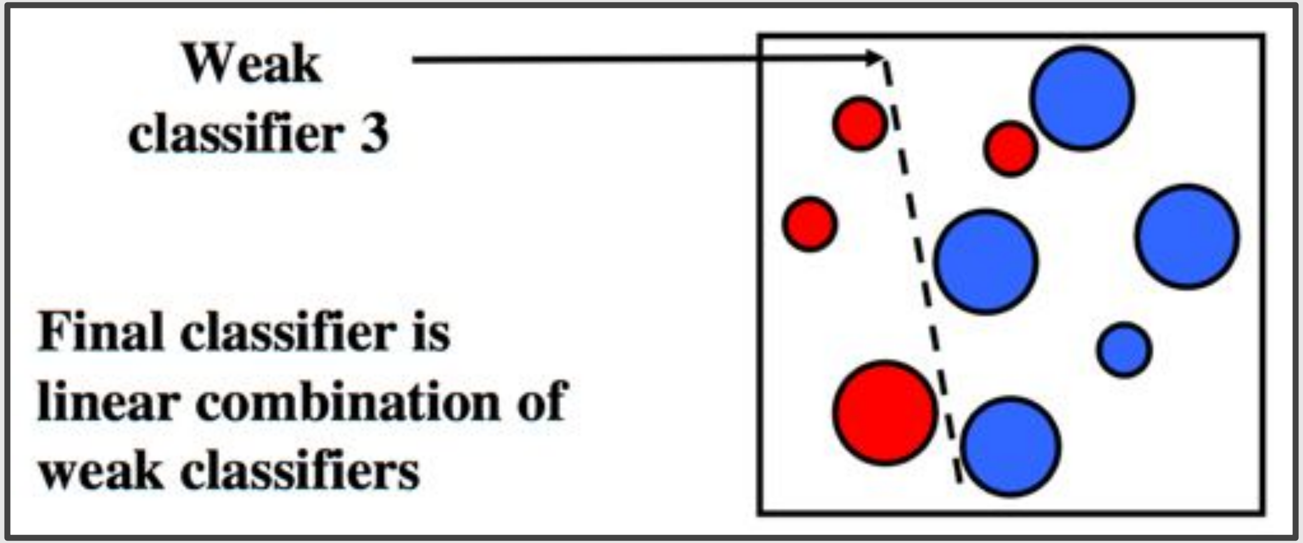
AdaBoost



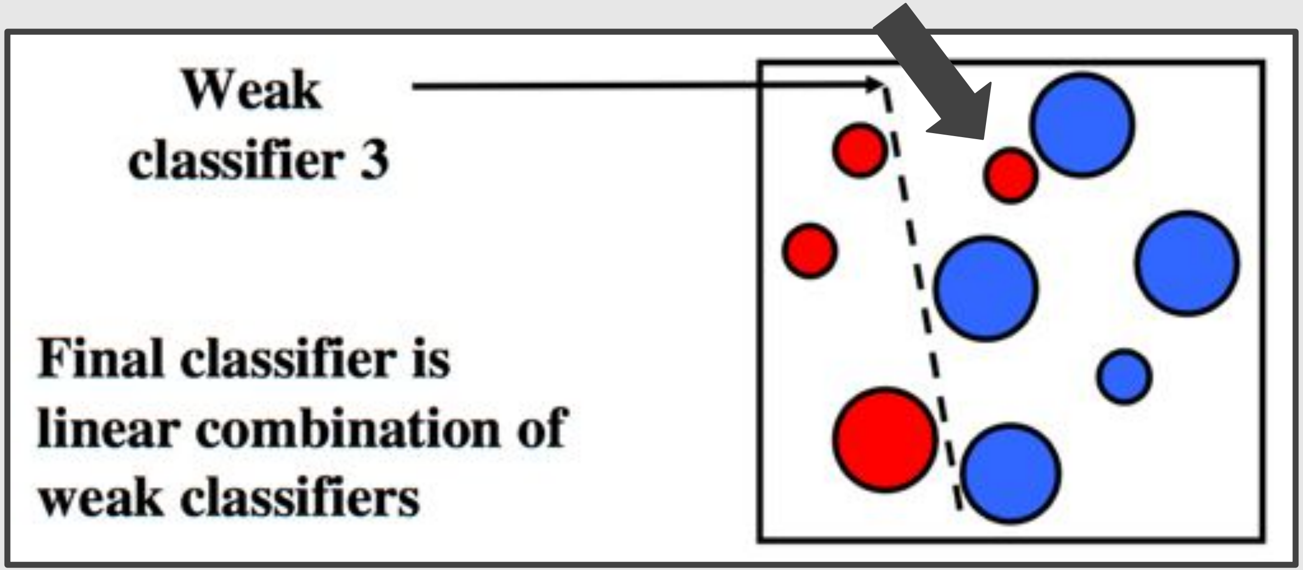
AdaBoost

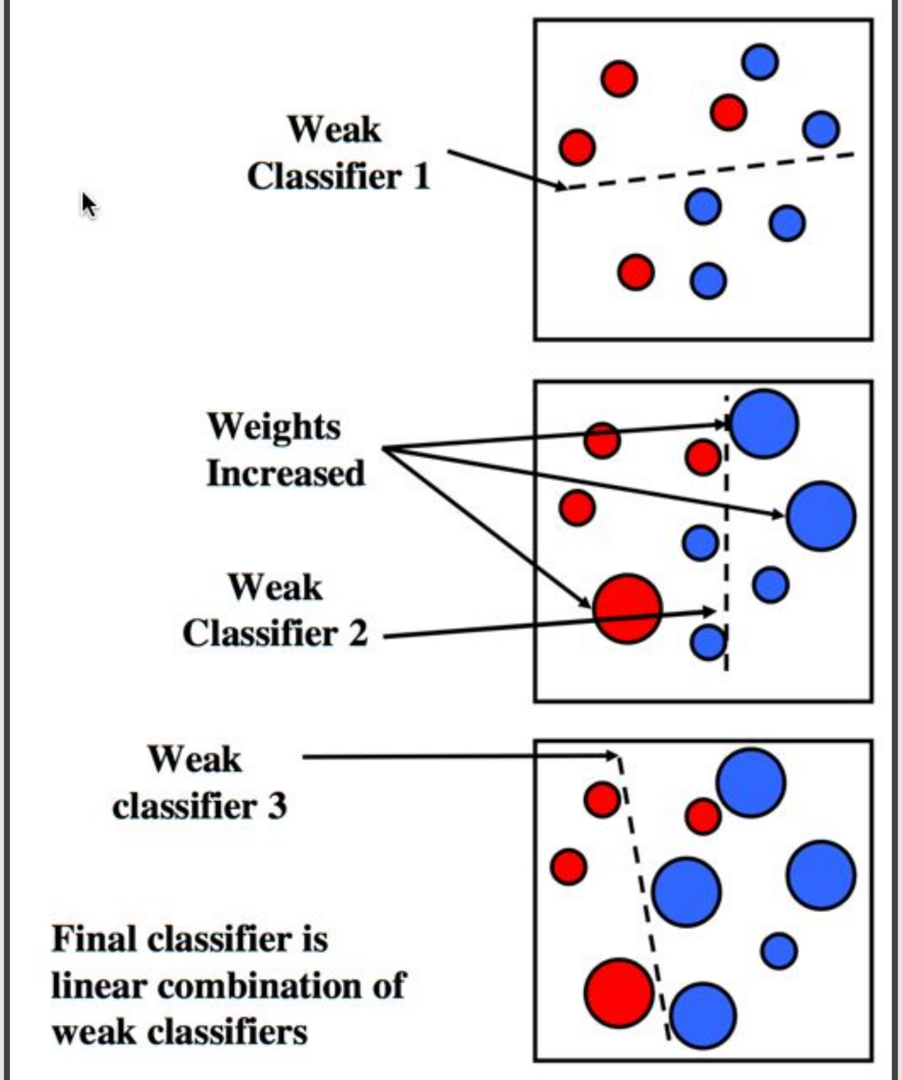


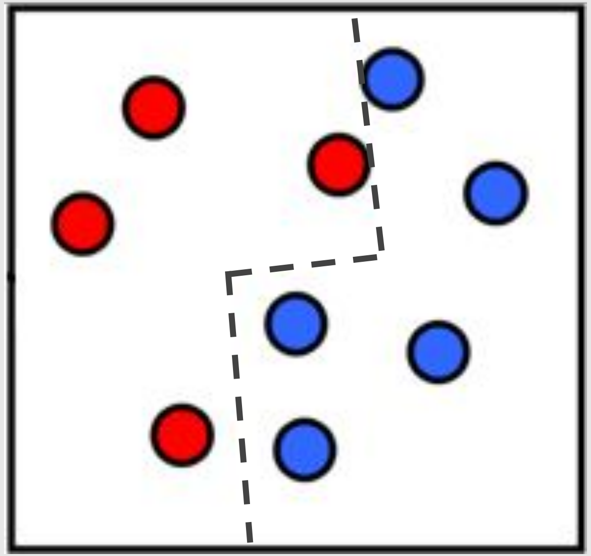
AdaBoost


Weak classifier 3
Final classifier is linear combination of weak classifiers

AdaBoost


Weak
classifier 3
Final classifier is
linear combination of
weak classifiers


Weak Classifier 1
Weights Increased
Weak Classifier 2
Weak classifier 3
Final classifier is linear combination of weak classifiers



AdaBoost

1. Assign every observation, x_i, an initial weight value, $w_i = \frac{1}{n}$, where n is the total number of observations.
2. Train a "weak" model. (most often a decision tree)
3. For each observation:
    3.1. If predicted incorrectly, w_i is increased
    3.2. If predicted correctly, w_i is decreased
4. Train a new weak model where observations with greater weights are given more priority.
5. Repeat steps 3 and 4 until observations perfectly predicted or a preset number of trees are trained.

ChrisAlbon

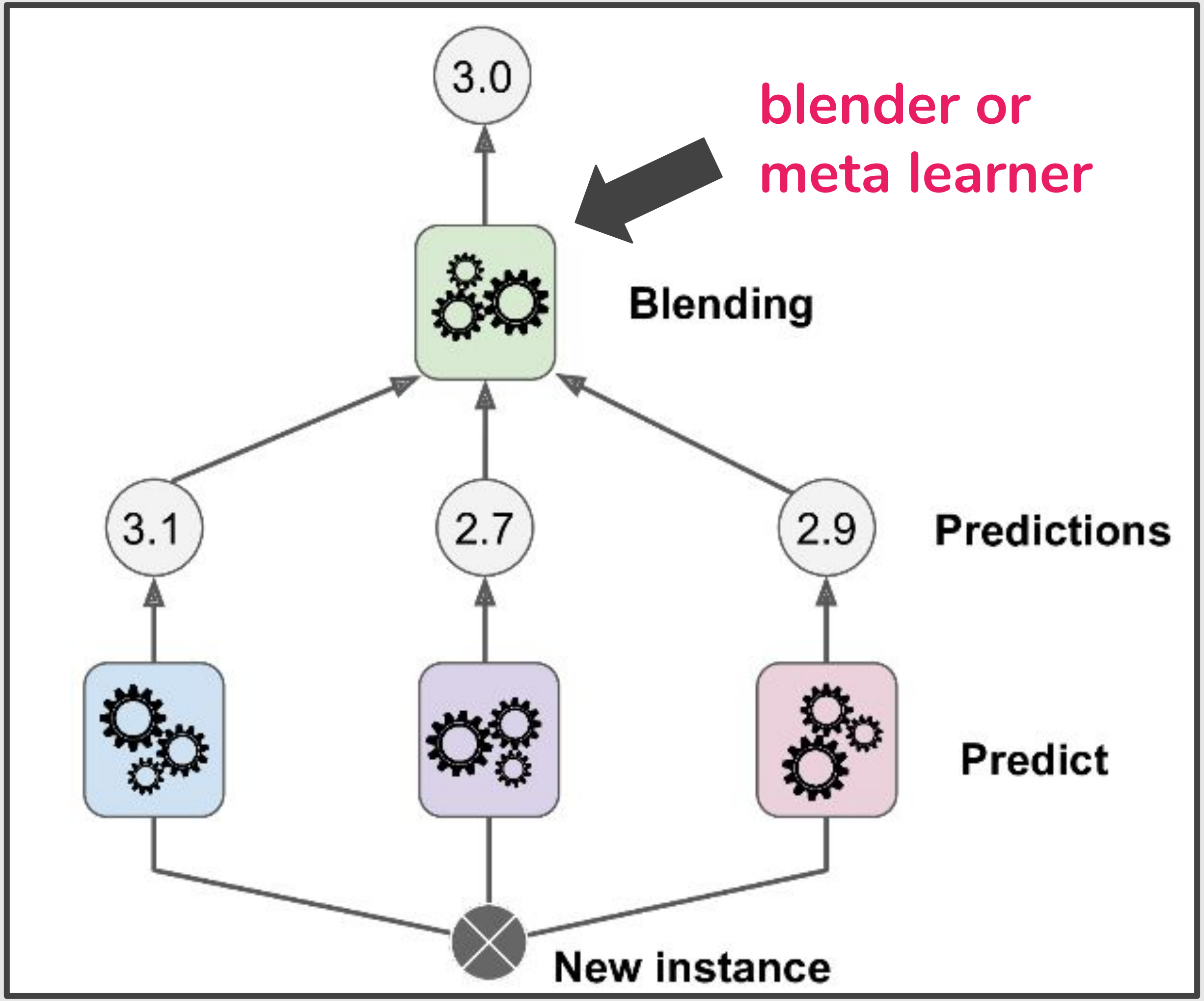
Stacking

Stacking [Wolpert, 1992]

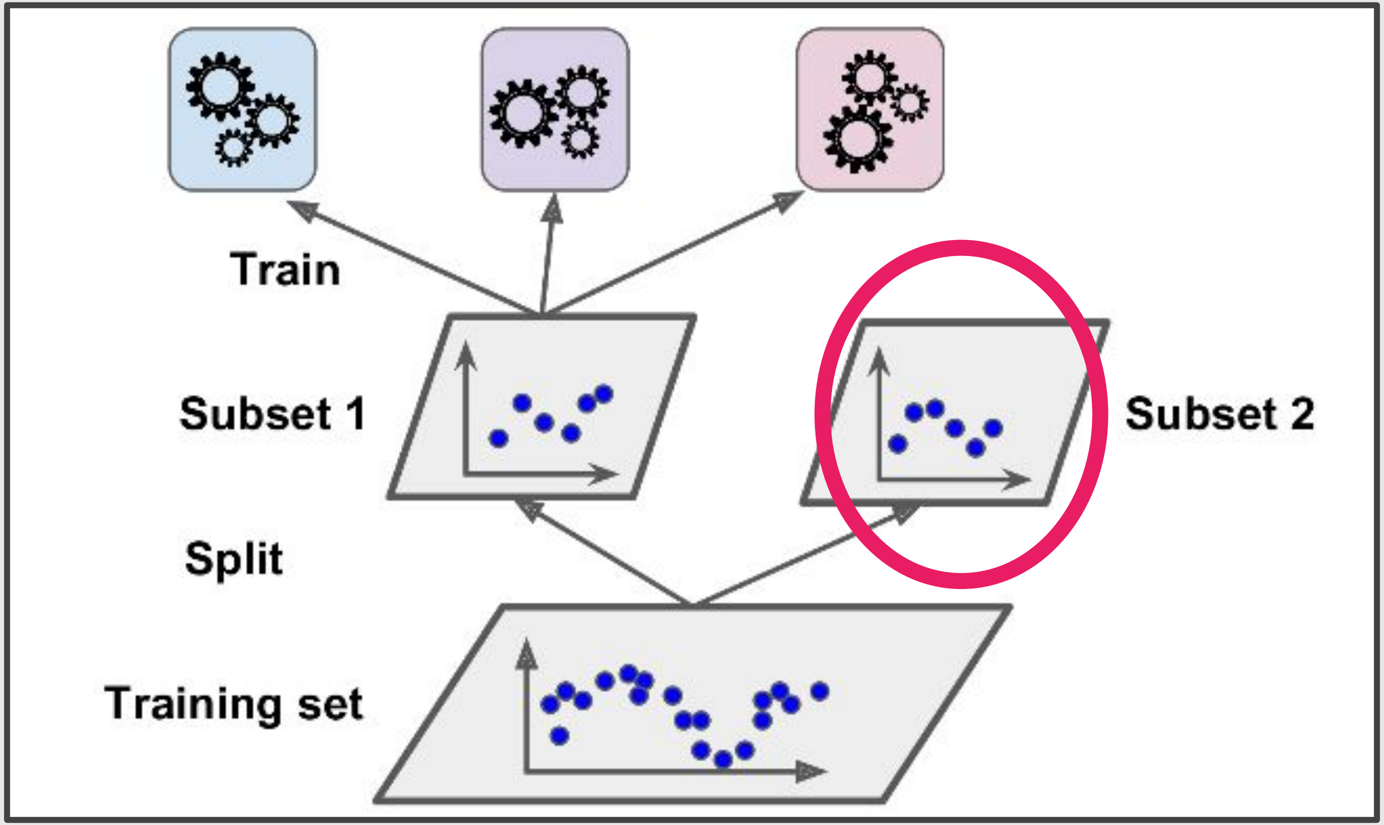
- Stacking (short for Stacked Generalization)
- Instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, we **train a model to perform this aggregation**.

“Stacked Generalization”, D. Wolpert (1992): http://goo.gl/9I2NBw

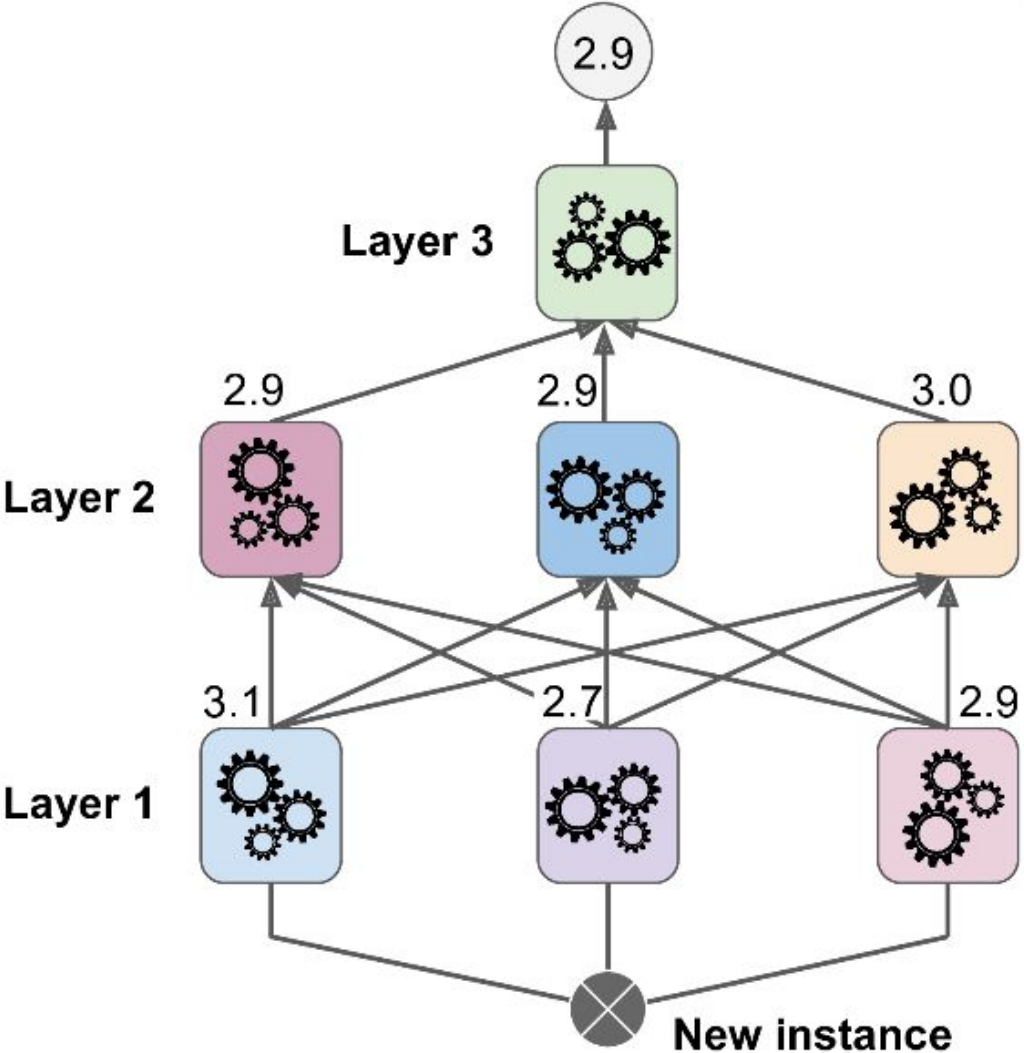
Stacking


3.0
blender or
meta learner
Blending
3.1
2.7
2.9
Predictions
Predict
New instance

To train the blender, a common approach is to use a **hold-out set**.



Multi-layer Stacking Ensemble



Random Forests

Machine Learning and Pattern Recognition

Prof. Sandra Avila

Institute of Computing (IC/Unicamp)

MC886/MO444, November 6, 2018

Decision Tree

Decision Tree & Random Forest

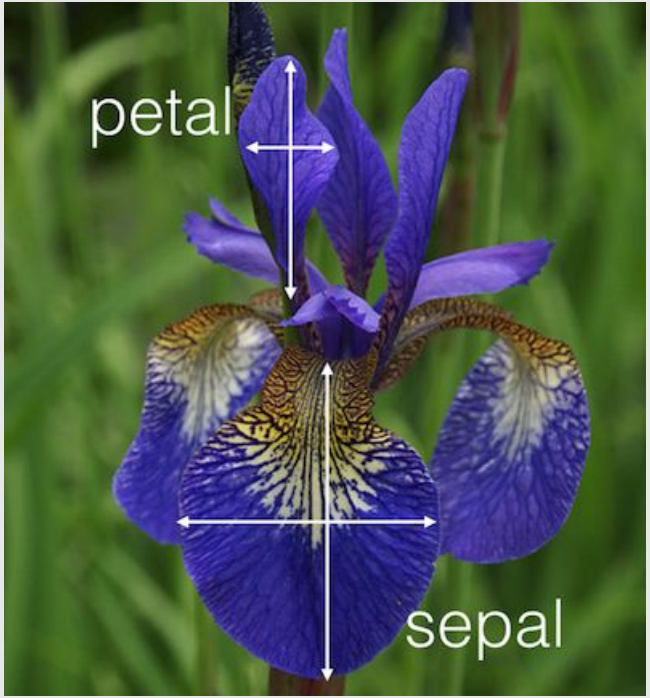
- **Decision Trees** are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multi-output tasks.

Decision Tree & Random Forest

- Decision Trees are versatile Machine Learning algorithms that can perform both classification and regression tasks, and even multi-output tasks.

- **Random Forest is an ensemble of Decision Trees**, generally trained using the Bagging method (or sometimes Pasting).

Decision Tree: Iris Dataset



http://sebastianraschka.com/Articles/2014_python_lda.html

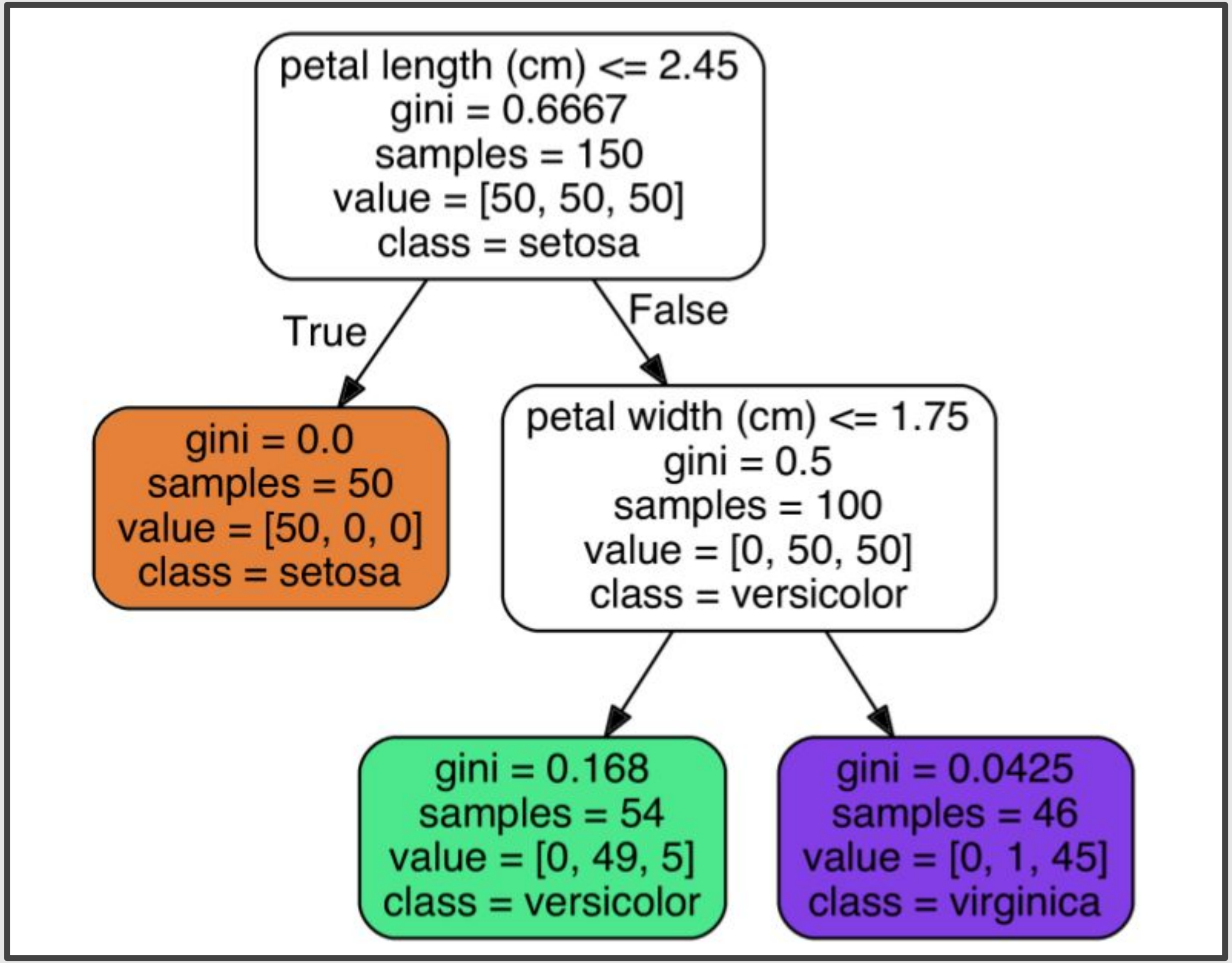
150 iris flowers from three different species.

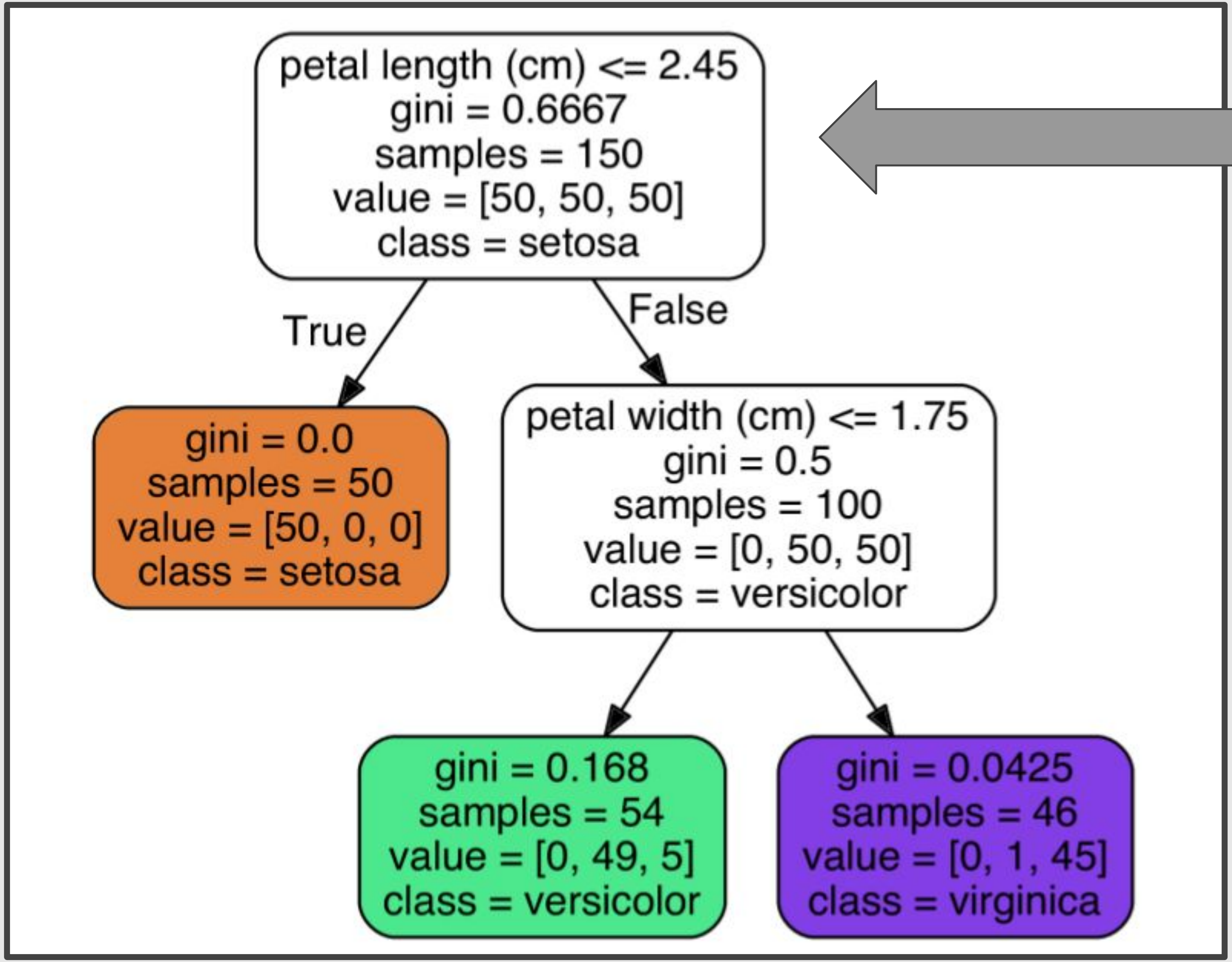
The three classes in the Iris dataset:

1. Iris-setosa (n=50)
2. Iris-versicolor (n=50)
3. Iris-virginica (n=50)

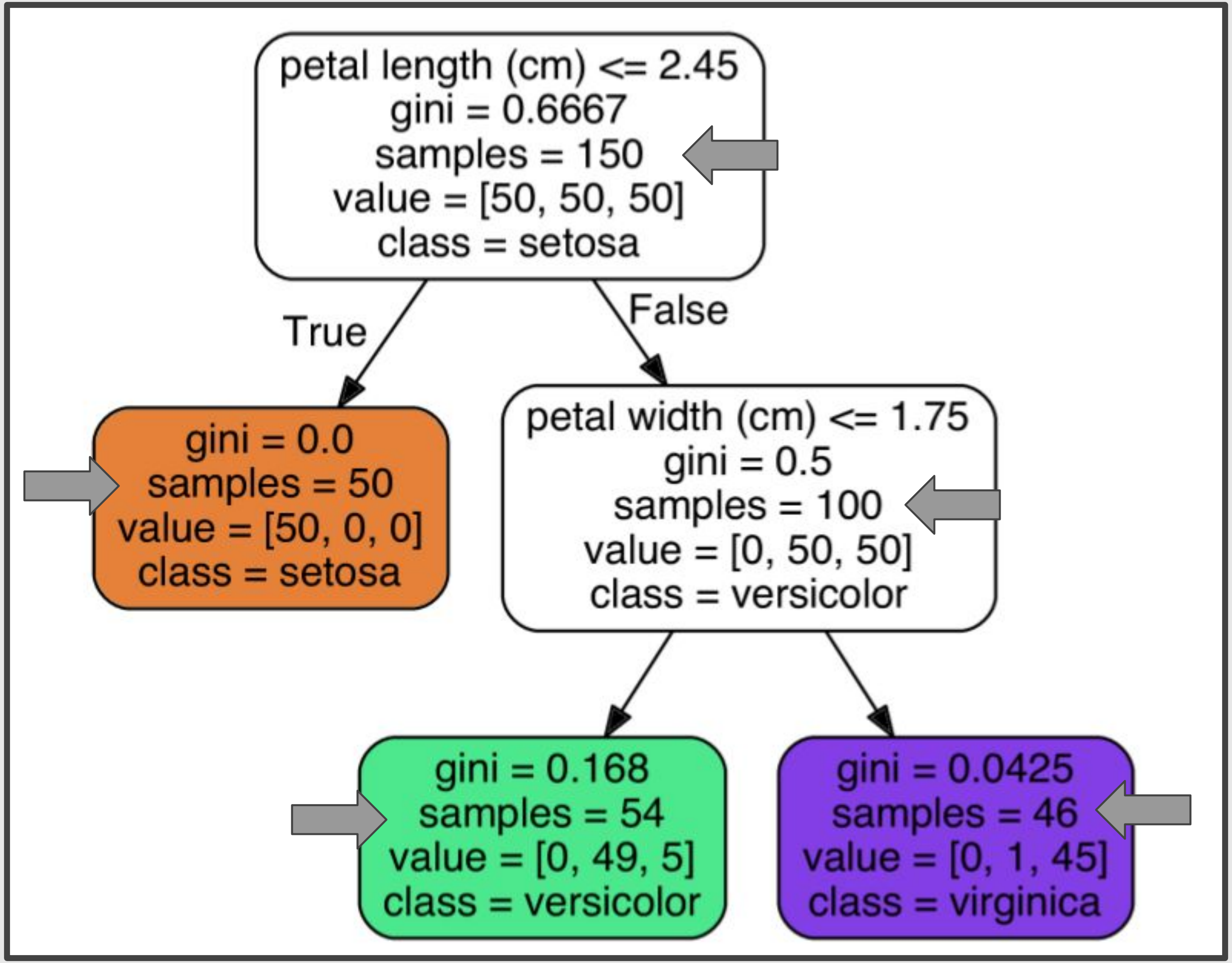
The four features of the Iris dataset:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

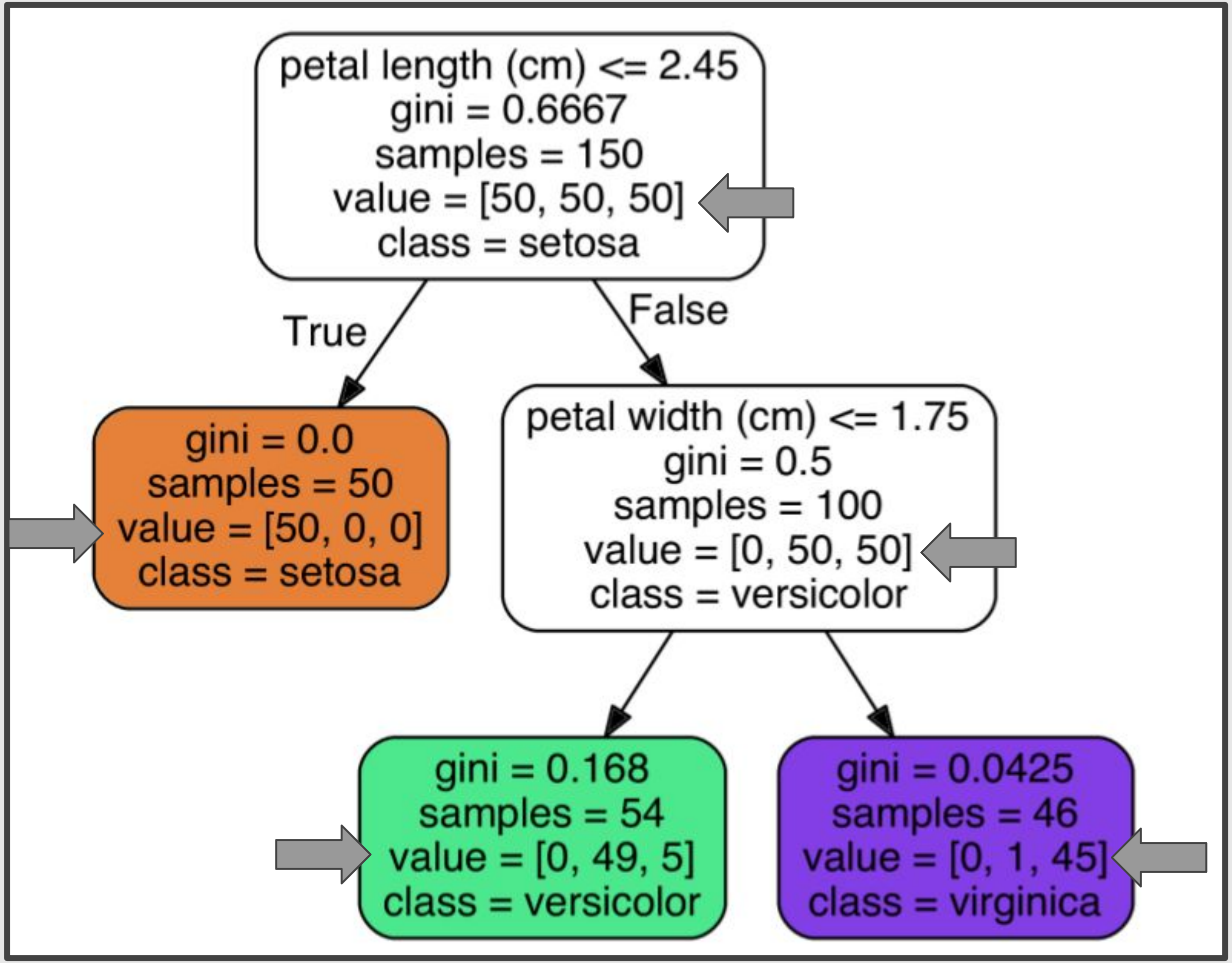

petal length (cm) <= 2.45
gini = 0.6667
samples = 150
value = [50, 50, 50]
class = setosa
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
class = setosa
petal width (cm) <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
class = versicolor
gini = 0.168
samples = 54
value = [0, 49, 5]
class = versicolor
gini = 0.0425
samples = 46
value = [0, 1, 45]
class = virginica



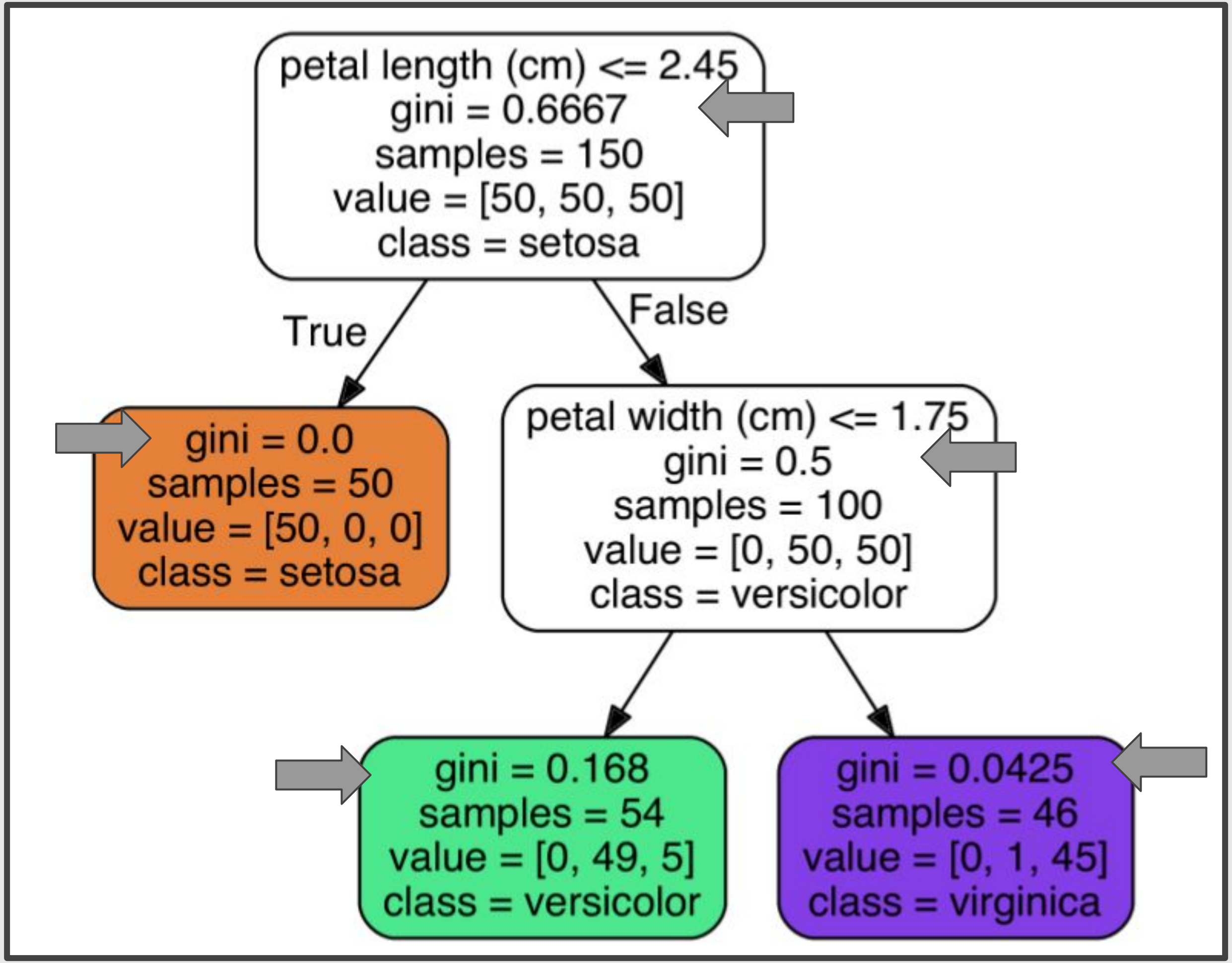
This node asks whether the flower's petal length is smaller than 2.45 cm



A **node's samples** attribute counts how many training instances it applies to.



A **node's value** attribute tells you how many training instances of each class this node applies to.



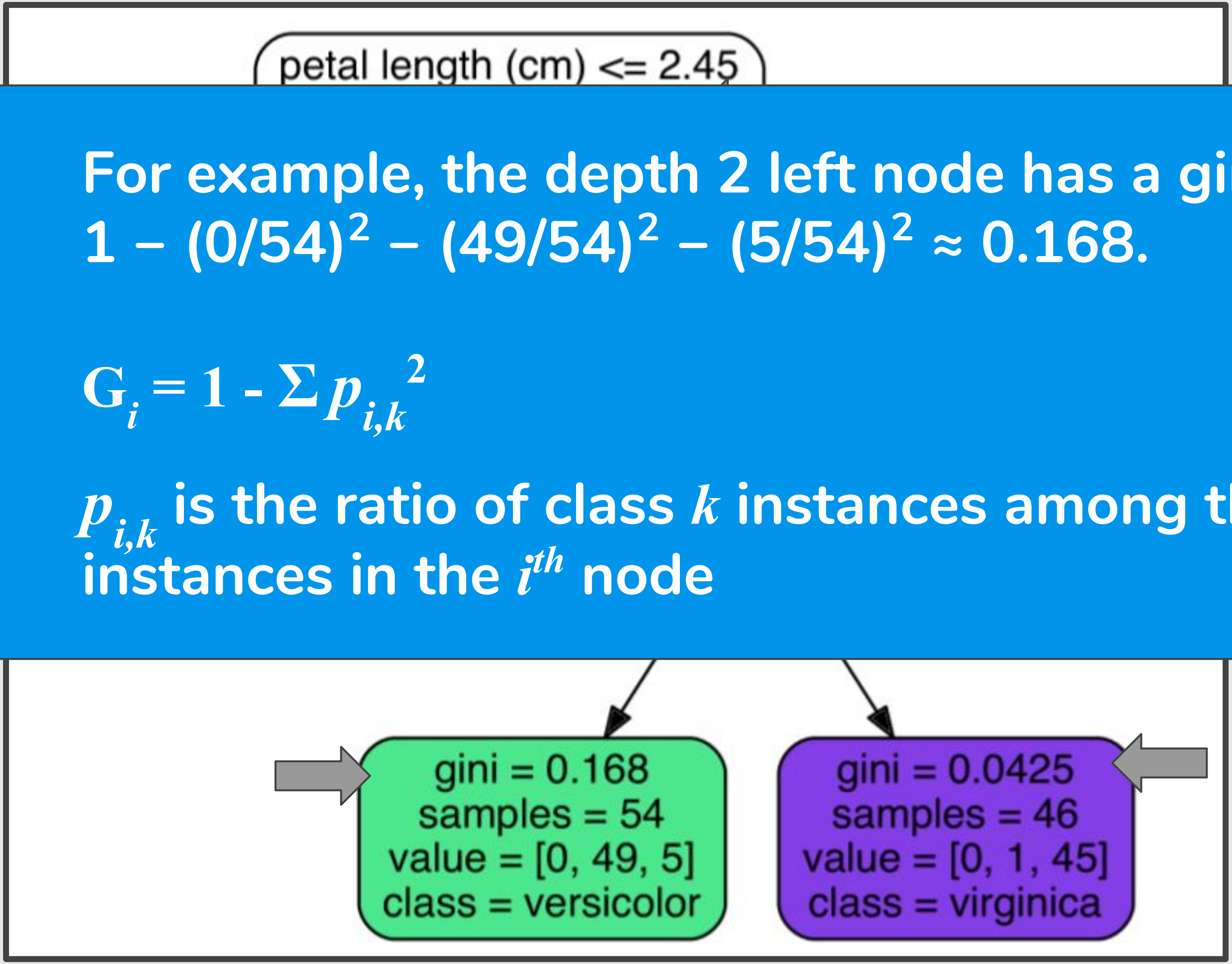
A **node's gini** attribute measures its impurity.

"pure" (gini=0): all training instances belong to the same class.

For example, the depth 2 left node has a gini score equal to $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$.

$$G_i = 1 - \Sigma\, p_{i,k}{}^2$$

$p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node

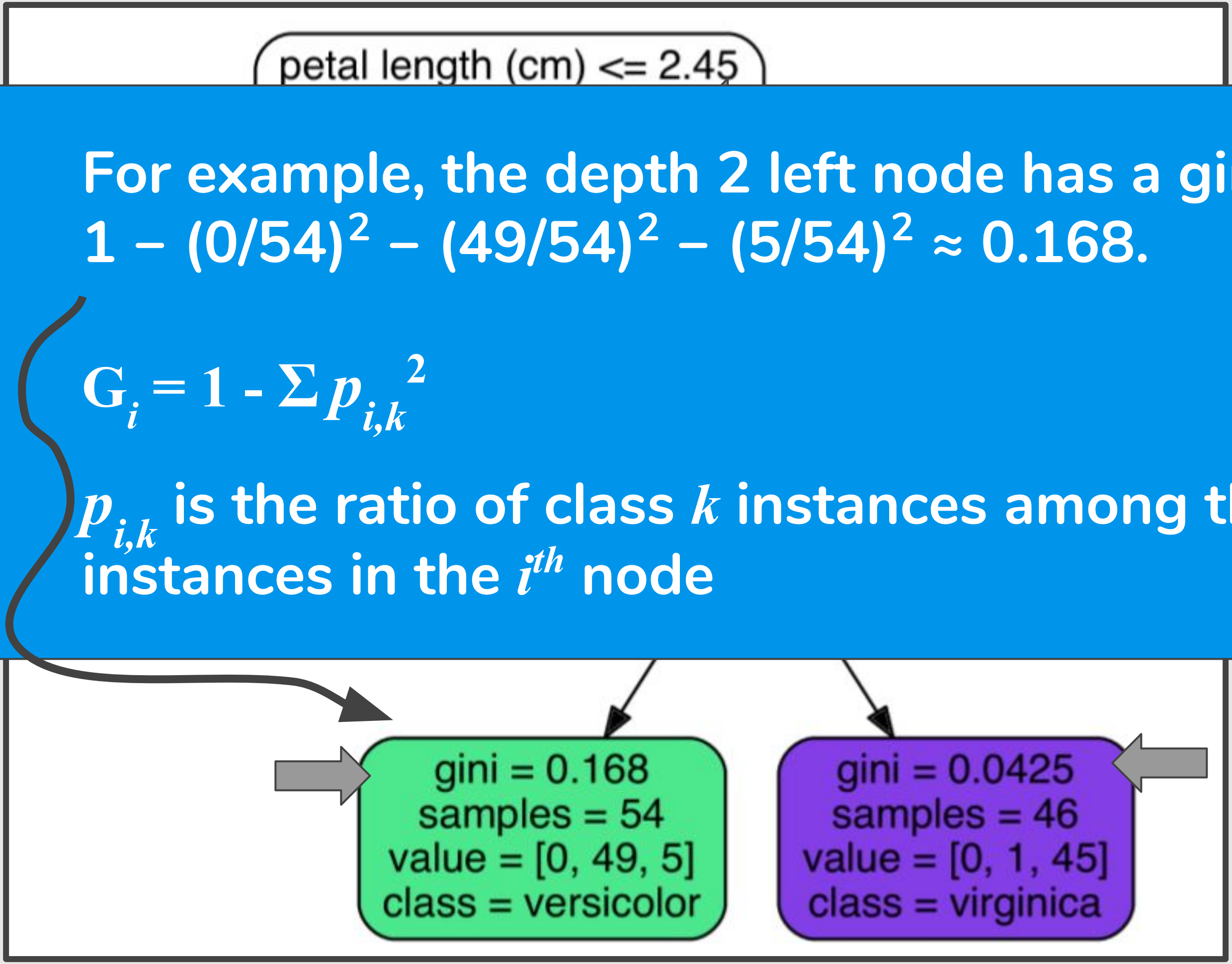


same class.

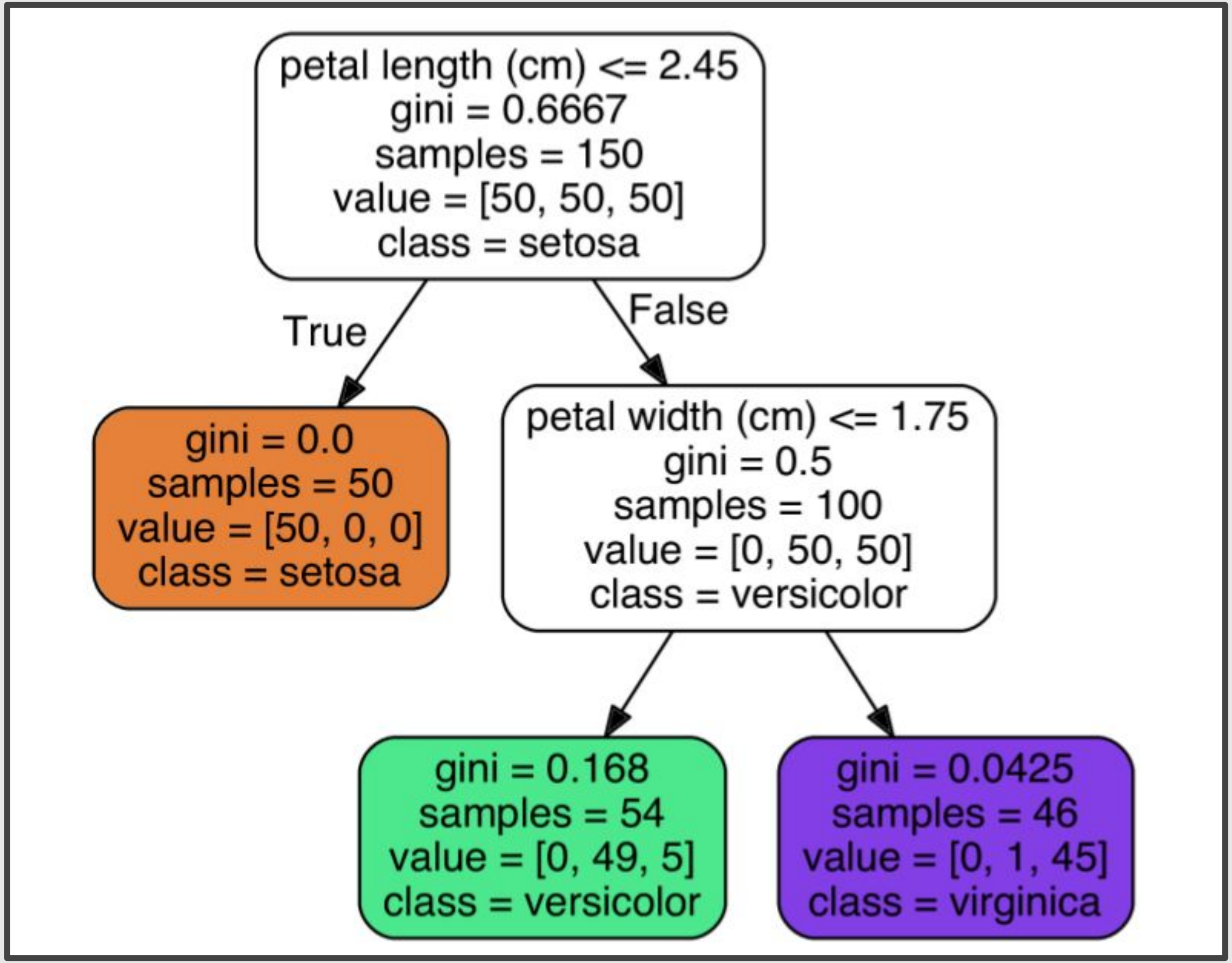
For example, the depth 2 left node has a gini score equal to $1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$.

$$G_i = 1 - \Sigma p_{i,k}{}^2$$

$p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node

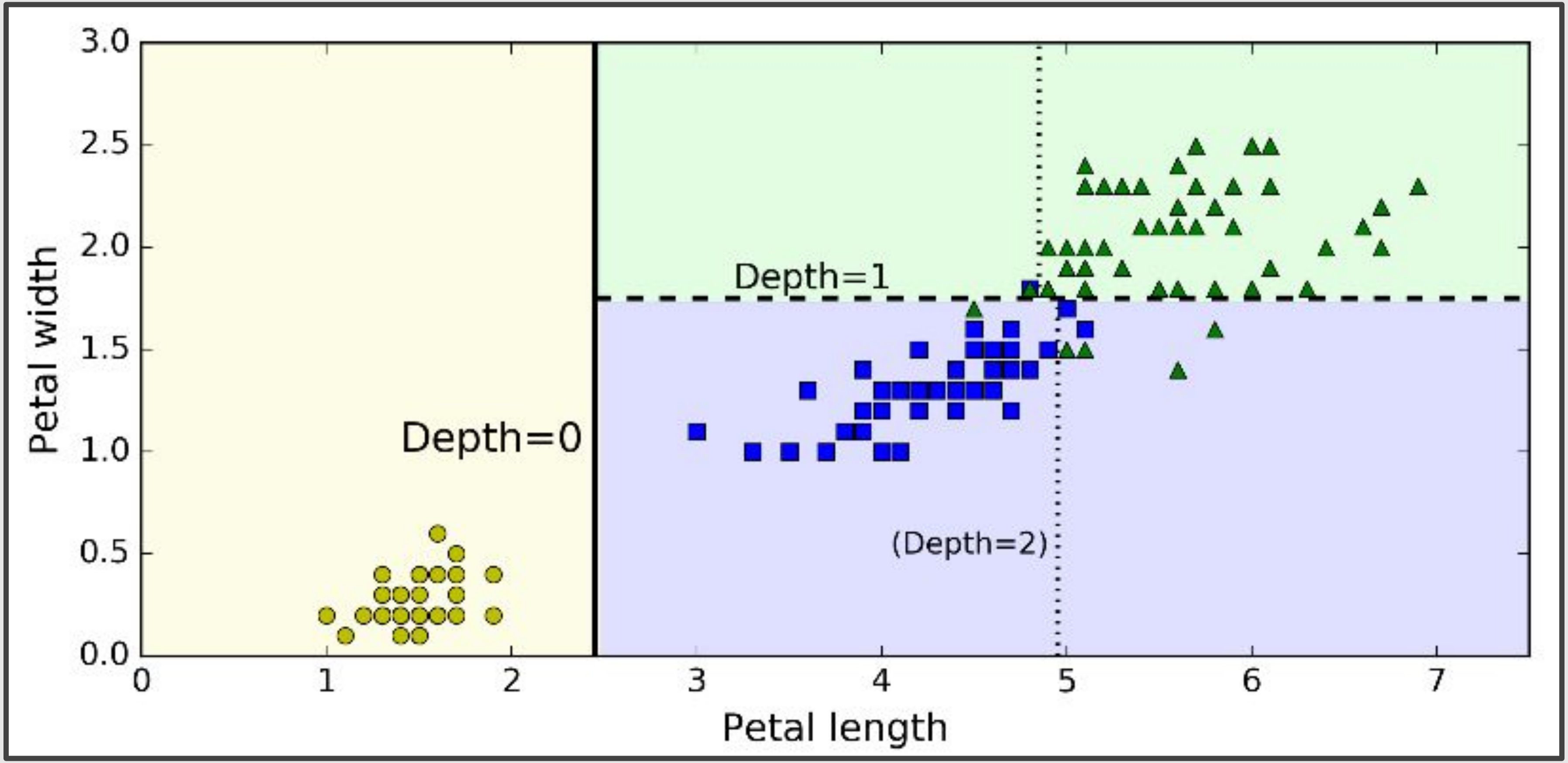


same class.



A **node's gini** attribute measures its impurity.

"pure" (gini=0): all training instances belong to the same class.



3.0
2.5
2.0
1.5
1.0
0.5
0.0
Petal width
Depth=0
Depth=1
(Depth=2)
0
1
2
3
4
5
6
7
Petal length

The CART Algorithm

- Classification And Regression Tree (CART) algorithm.

The CART Algorithm

- Classification And Regression Tree (CART) algorithm.
- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g. “petal length ≤ 2.45 cm”).

The CART Algorithm

- Classification And Regression Tree (CART) algorithm.
- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g. “petal length ≤ 2.45 cm”).
- How does it choose k and t_k?

The CART Algorithm

- Classification And Regression Tree (CART) algorithm.
- The idea is really quite simple: the algorithm first splits the training set in two subsets using a single feature k and a threshold t_k (e.g. “petal length ≤ 2.45 cm”).
- How does it choose k and t_k?

  It searches for the pair (k, t_k) that produces the purest subsets (weighted by their size).

The CART Algorithm

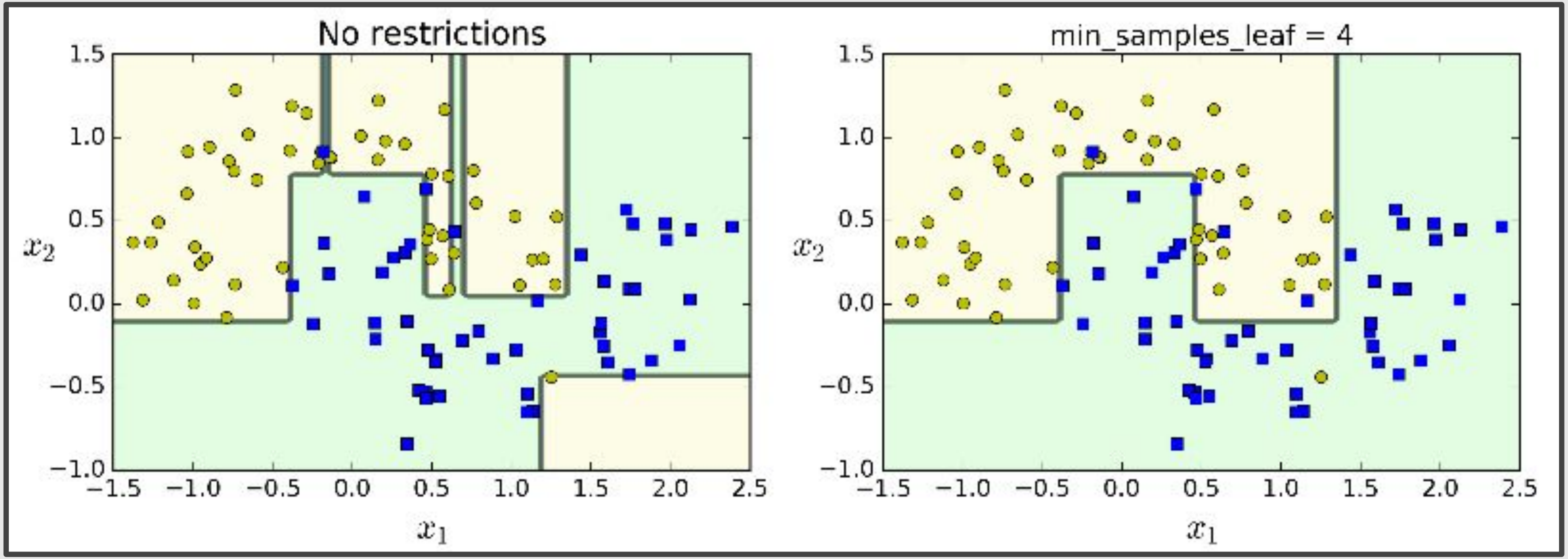
$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

$$\text{where} \begin{cases} G_{\text{left/right}} \text{ measures the impurity of the left/right subset,} \\ m_{\text{left/right}} \text{ is the number of instances in the left/right subset.} \end{cases}$$

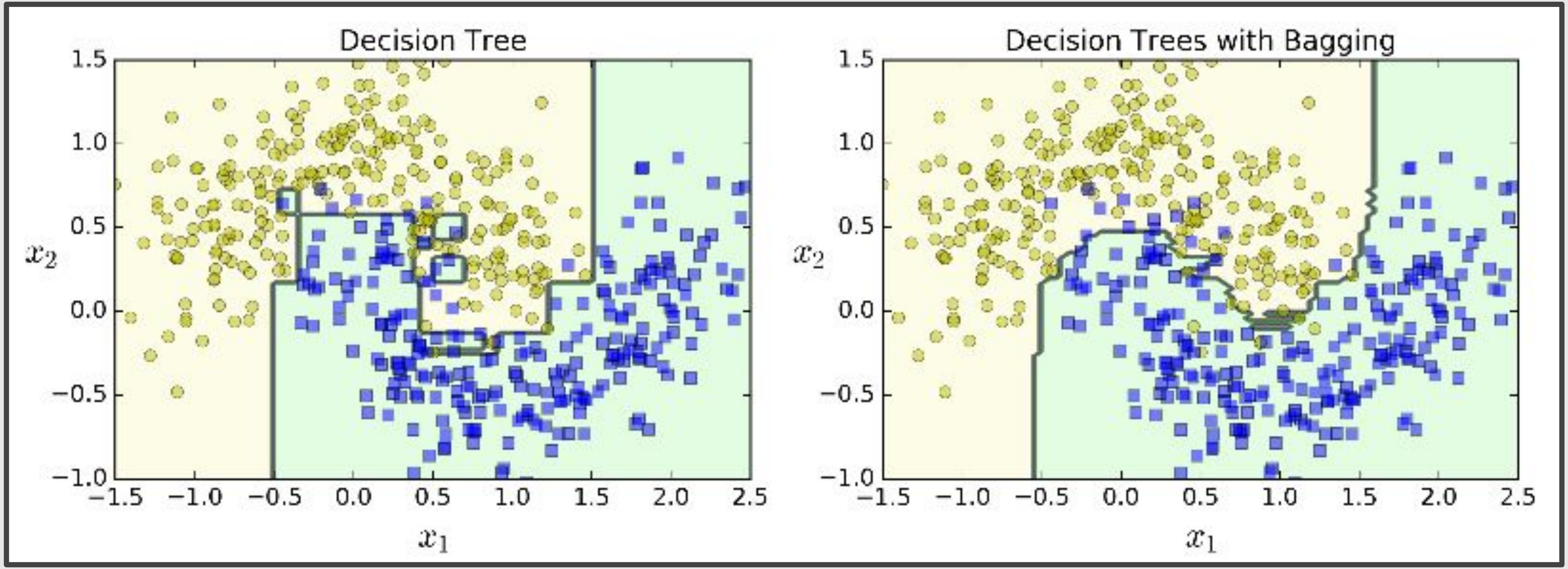
CART cost function for classification

It stops recursing once it reaches the maximum depth (hyperparameter), or if it cannot find a split that will reduce impurity.

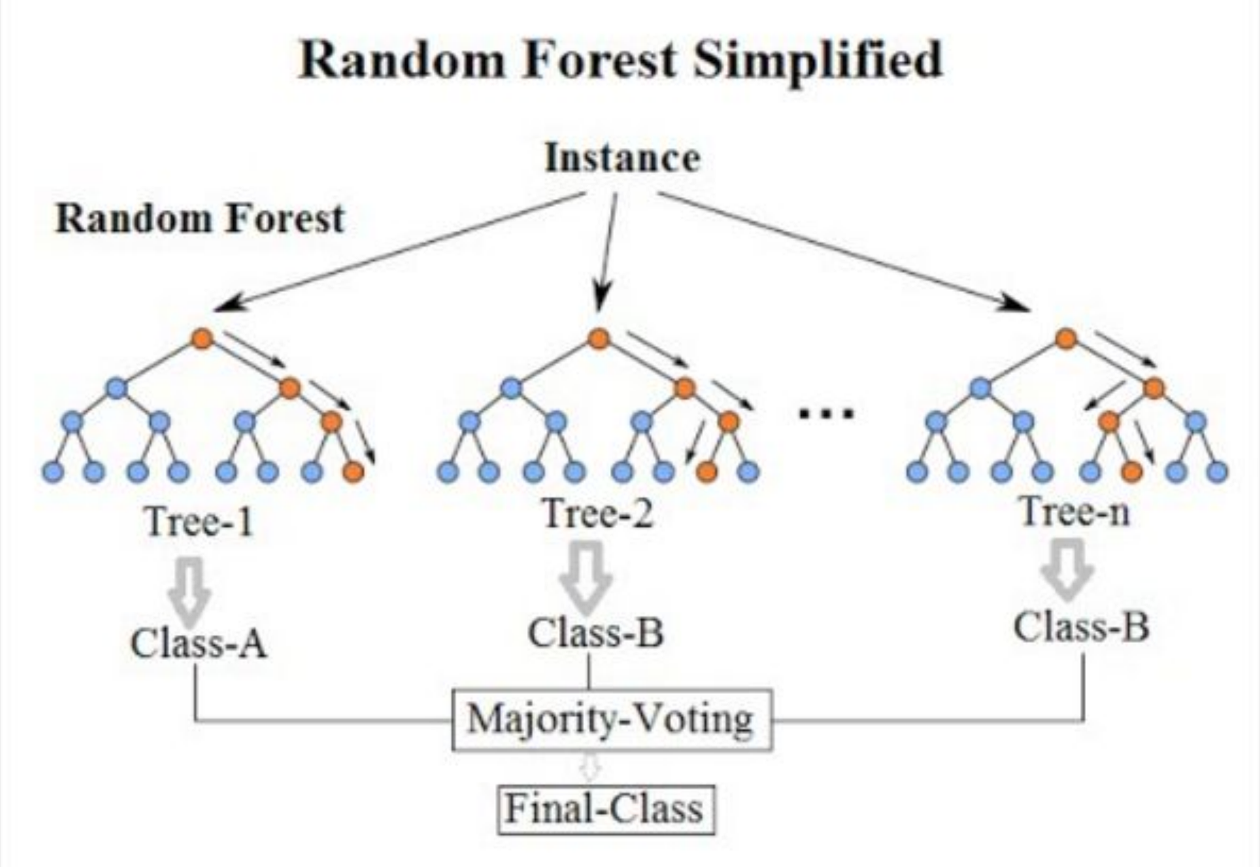
Regularization



Regularization



Random Forest



https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

Random Forest [Ho, 1995]

- Random Forest is an ensemble of Decision Trees, generally trained using the Bagging method.

“Random Decision Forests”, Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest [Ho, 1995]

- Random Forest is an ensemble of Decision Trees, generally trained using the Bagging method.

- **Extra randomness when growing trees**:
  - Instead of searching for the very best feature when splitting a node, it searches for the best feature among a **random subset of features**.

"Random Decision Forests", Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest [Ho, 1995]

1. Assume number of cases in the training set is *N*. Then, sample of these *N* cases is taken at random but with replacement.

“Random Decision Forests”, Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest [Ho, 1995]

2. If there are *M* input variables, a number *m*<*M* is specified such that at each node, *m* variables are selected at random out of the *M*.

   The best split on these *m* is used to split the node. The value of *m* is held constant while we grow the forest.

“Random Decision Forests”, Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest [Ho, 1995]

3. Each tree is grown to the largest extent possible and there is no pruning.

4. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

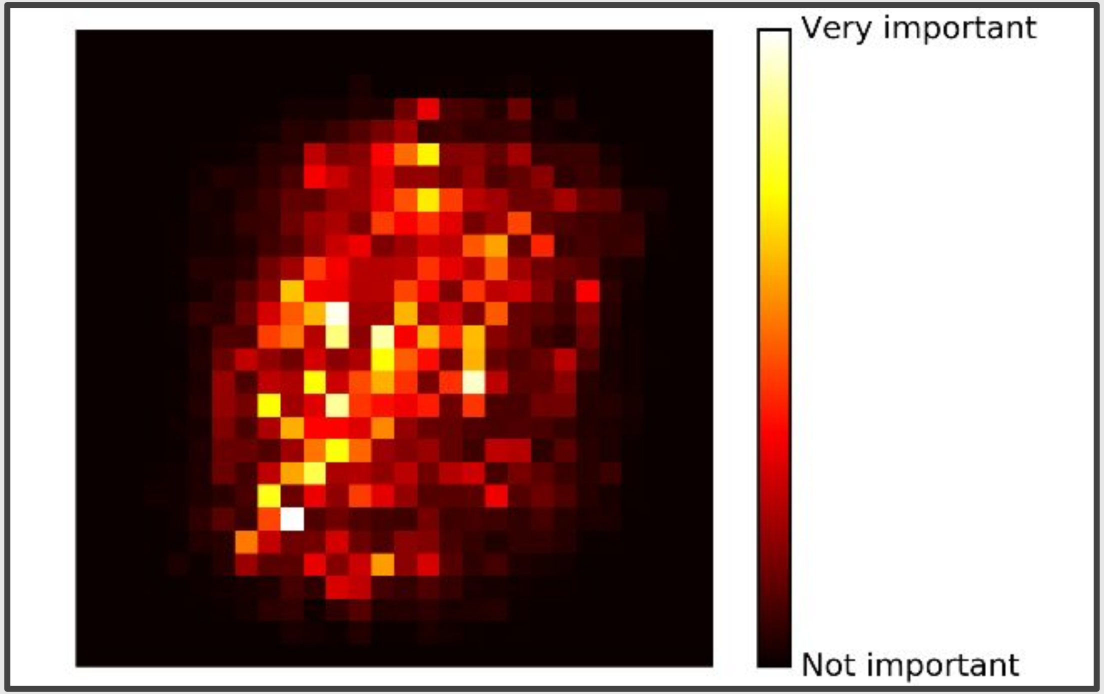
“Random Decision Forests”, Tin Kam Ho (1995): http://goo.gl/zVOGQ1

Random Forest: Feature Importance

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)

sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Random Forest: Feature Importance



Forward Thinking: Building Deep Random Forests

Kevin Miller, Chris Hettinger, Jeffrey Humpherys, Tyler Jarvis, and David
Department of Mathematics
Brigham Young University
Provo, Utah 84602
millerk5@byu.edu, hettinger@math.byu.edu, jeffh@math.byu.
jarvis@math.byu.edu, david.kartchner@math.byu.edu

Abstract

Distributed Deep Forest and its Application to Automatic Detection of Cash-out Fraud

Ya-Lin Zhang[†], Jun Zhou[‡], Wenhao Zheng[†], Ji Feng[†], Longfei Li[‡], Ziqi Liu[‡], Ming Li[†], Zhiqiang Zhang[‡], Chaochao Chen[‡], Xiaolong Li[‡], Zhi-Hua Zhou[†]
[†]National Key Lab for Novel Software Technology, Nanjing University, China
[†]{zhangyl, zhengwh, fengj, lim, zhouzh}@lamda.nju.edu.cn
[‡]Ant Financial Services Group, China
[‡]{jun.zhoujun, longyao.llf, ziqiliu, lingyao.zzq, chaochao.ccc, xl.li}@antfin.com

Training Big Random Forests with Little Resources

Fabian Gieseke
Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
fabian.gieseke@di.ku.dk

Christi
Department of C
University of
Copenhage
igel@di

ABSTRACT

Without access to large compute clusters, building random forests on large datasets is still a challenging problem. This is, in particular, the case if fully-grown trees are desired. We propose a simple yet effective framework that allows to efficiently construct ensembles of huge trees for hundreds of millions or even billions of training instances using a cheap desktop computer with commodity hardware. The basic idea is to consider a multi-level construction scheme, which builds top trees for small random subsets of the available data and which subsequently distributes all training instances to the top trees' leaves for further processing. While being conceptually simple, the overall efficiency crucially depends on the particular implementation of the different phases. The practical merits of our

ensembles in a parallel or distr
dividual compute nodes (e.g., by
node). While this can significa
such frameworks naturally req
ing environments. Further, the
might cause problems in case
large to fit into the main memo

In this work, we propose a
scheme for building random fo
scale. The main idea is to build
phases: Starting with a top tree
the data, one subsequently dist
leaves of that tree. For each lea

Deep Forest: Towards an Alternative to Deep Neural Networks*

Zhi-Hua Zhou and **Ji Feng**
National Key Lab for Novel Software Technology, Nanjing University, Nanjing 210023, China
{zhouzh, fengj}@lamda.nju.edu.cn

Abstract

In this paper, we propose gcForest, a decision tree

ample, even when several authors all use convolutional neural networks [LeCun *et al.*, 1998; Krizhenvsky *et al.*, 2012; Simonyan and Zisserman, 2014], they are actually using dif-

References

Machine Learning Books

- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Chap. 6 & 7
- Pattern Recognition and Machine Learning, Chap. 14
- Pattern Classification, Chap 8 & 9 (Sec. 9.5)

- https://towardsdatascience.com/random-forest-in-python-24d0893d51c0