CS286 AI for Science and Engineering

Lecture 4: Traditional Machine Learning (Part 2)

Jie Zheng (郑杰)

PhD, Associate Professor

School of Information Science and Technology (SIST), ShanghaiTech University

August, 2020

Outline

- Introduction to Decision Tree (DT)
- Introduction to Ensemble Learning
  - Bagging and Pasting
  - Random Forests
  - Boosting
  - Stacking

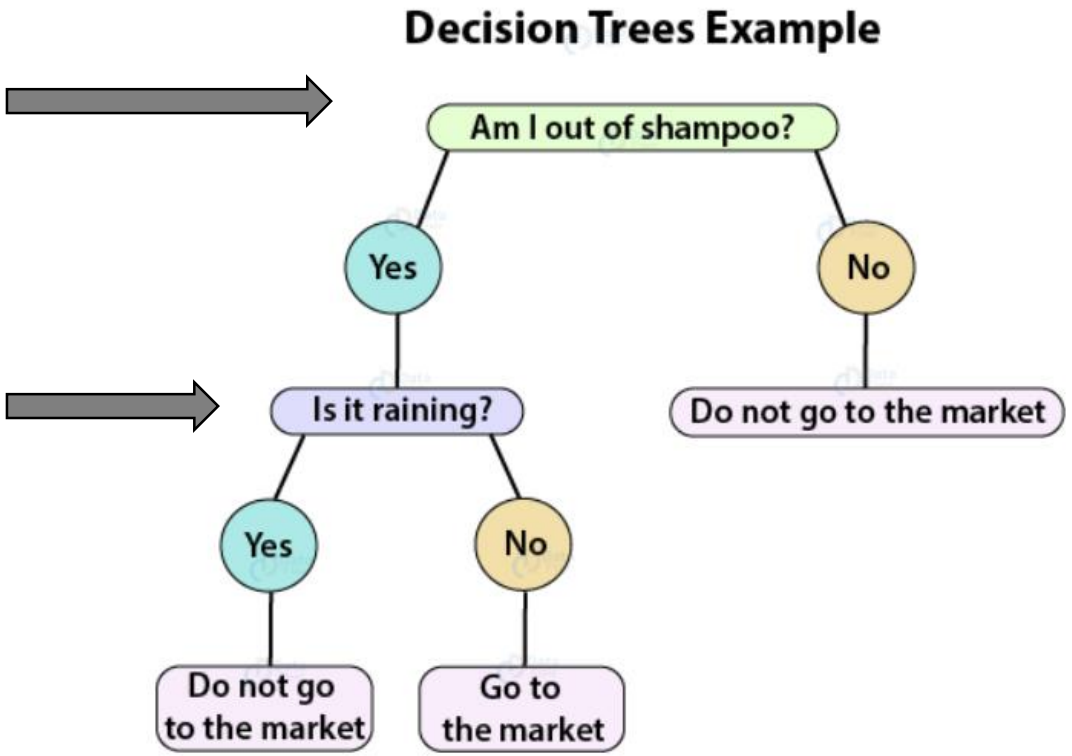
Decision Tree

Tasks: classification & regression

- Classification
  - Specify the class to which data elements belong to, or predict a class for an input variable
  - Best used when the output has finite and discrete values
- Regression
  - To estimate the relationships among variables
- Applications
  - Marketing and sales
  - Medical diagnosis
  - Reducing churn rate
  - Anomaly & fraud detection

An example decision tree (DT)

- Suppose you have to go to the market to buy shampoo
- First, you assess if you really need the product, i.e. whether you are out of shampoo or not. Suppose you will only buy shampoo if you run it out
- If you do not have the shampoo, you will evaluate the weather outside to see if it is raining or not. If it is not raining, you will go; otherwise, you won’ t



Training a DT

- Dataset: iris dataset
  - Iris.data:
    - The rows being the samples
    - The columns being: Sepal Length, Sepal Width, Petal Length and Petal Width
    - Stored in a 150x4 numpy.ndarray
  - Iris.target
    - There are 3 types of irises: Setosa, Versicolour, and Virginica

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
```

→ Import python package

```
iris = load_iris()
iris = load_iris()
```

→ Import iris dataset

```
X = iris.data[:, 2:] # petal length and width
y = iris.target
tree_clf = DecisionTreeClassifier(max_depth=2)
tree_clf.fit(X, y)
```

→ Use the information of petal length and width to train a DT

Hyperparameter

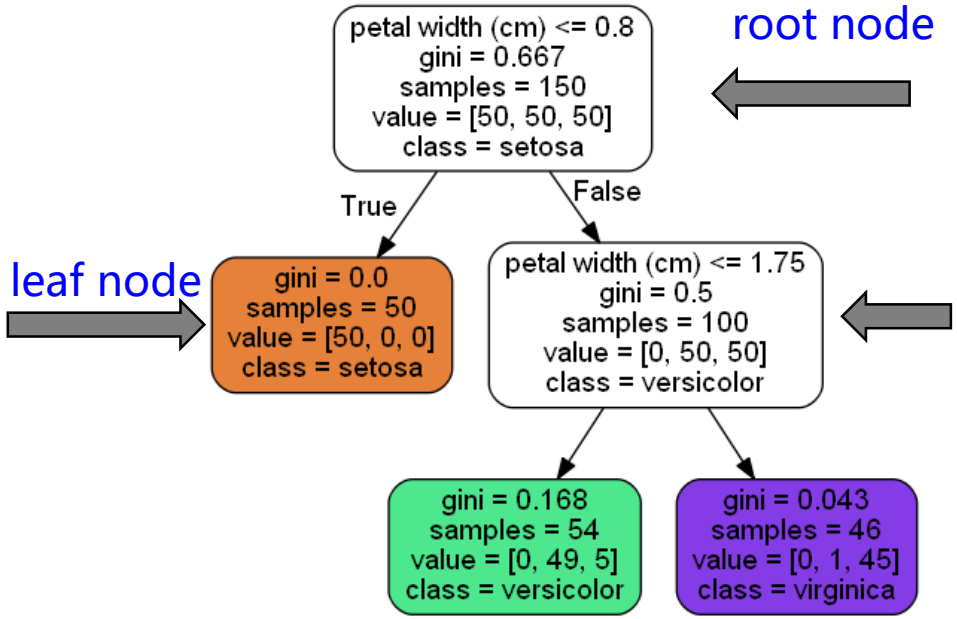
Visualizing a DT

- You can visualize the trained Decision Tree by first using the export_graphviz() method to output a graph definition file called iris_tree.dot
- Then you can convert this dot file to a variety of formats such as PDF or PNG using the dot command-line tool from the graphviz package

```
from sklearn.tree import export_graphviz

image_path = "D:\\lectures\\iris_tree.dot"

export_graphviz(
    tree_clf,
    out_file=image_path,
    feature_names=iris.feature_names[2:],
    class_names=iris.target_names,
    rounded=True,
    filled=True)
```



root node

Ask whether the flower's petal length is smaller than 2.45 cm

leaf node

It does not ask any questions: you can simply look at the predicted class for that node

The petal length is greater than 2.45 cm -> move down to the root's right child node (depth 1, right)
Ask whether the petal width is smaller than 1.75 cm

Visualizing a DT

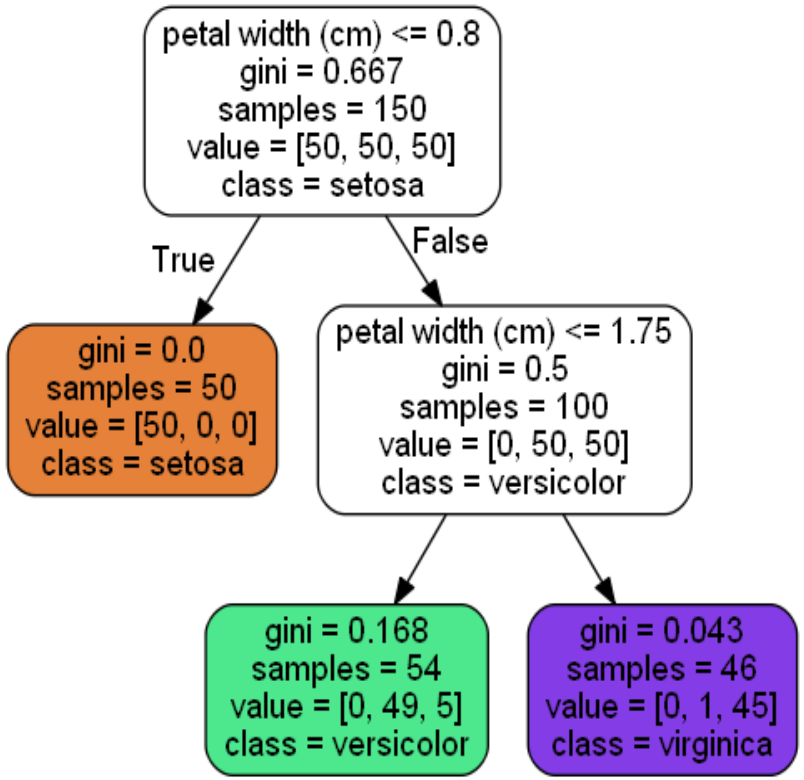
- Attributes of tree nodes:
  - A node's samples attribute counts how many training instances it applies to
  - A node's value attribute tells you how many training instances of each class this node applies to, e.g. 0 Iris-Setosa, 50 IrisVersicolor, and 50 Iris-Virginica
  - Finally, a node's gini attribute measures its impurity (e.g. a node is pure, if its gini is 0)
- How to calculate the gini score G_i of the i^{th} node?

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

  - $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node
  - Example: The depth-2 left node (in green) has a gini score

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$$

  - The depth-1 left node (in orange) applies only to Iris-Setosa training instances, it is pure and its gini score is 0



Gini impurity or entropy

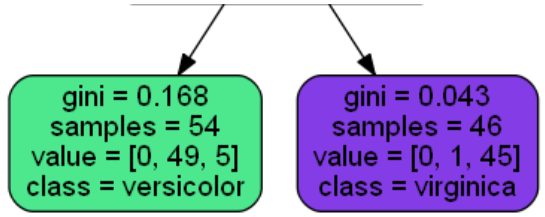
- By default, the Gini impurity measure is used, but you can select the entropy impurity measure instead by setting the criterion hyperparameter to "entropy"
  - The concept of entropy originated in thermodynamics as a measure of molecular disorder, later spread to Shannon's information theory. In Machine Learning, it is frequently used as a measure of impurity

Computing entropy

$$H_i = -\sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2 \left(p_{i,k}\right)$$

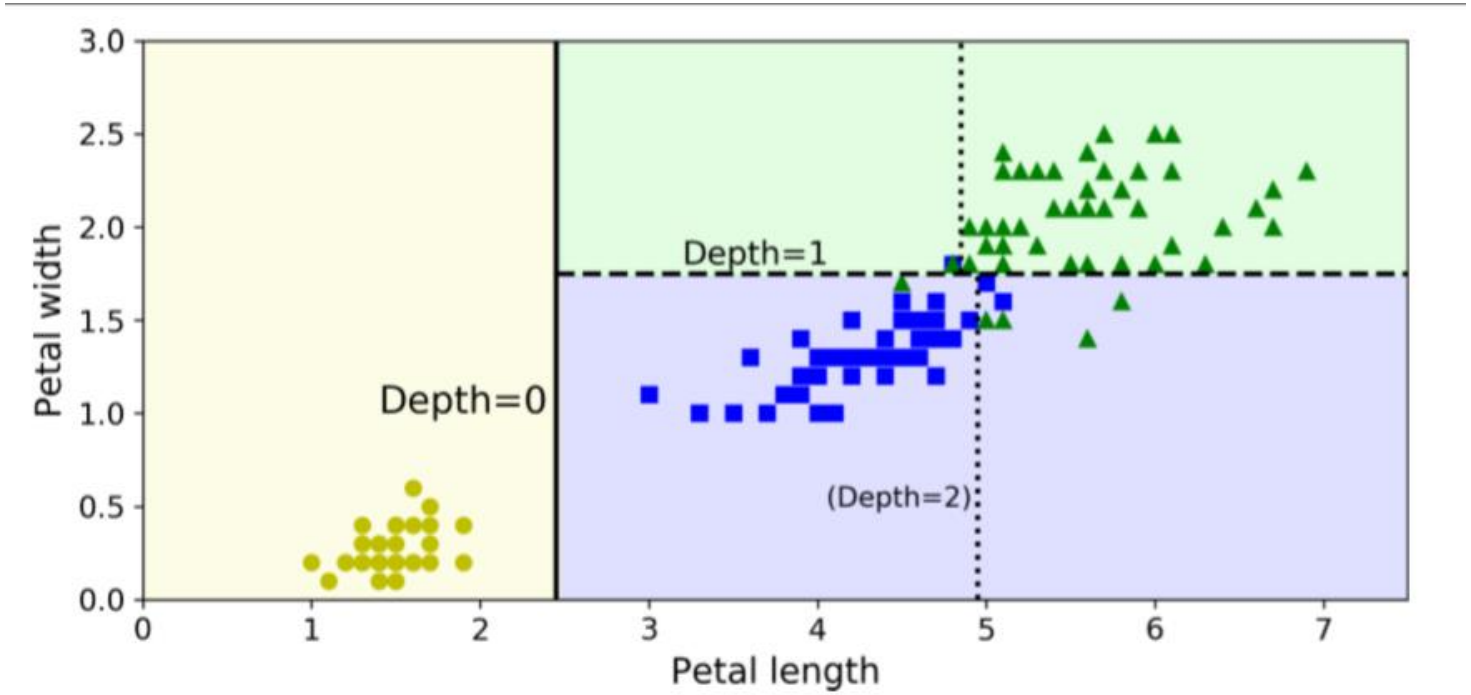
For example, the depth-2 left has an entropy equal to

$$-\frac{49}{54} \log_2 \left(\frac{49}{54}\right) - \frac{5}{54} \log_2 \left(\frac{5}{54}\right) \approx 0.445.$$



DT’s decision boundaries

- The thick vertical line: the decision boundary of the root node (depth 0, petal length = 2.45 cm)
- The dashed line: the depth-1 right node splits it at petal width = 1.75
- The dotted lines: if you set max_depth to 3, then the two depth-2 nodes would each add another decision boundary



Estimating class probabilities

- Estimating the probability that an instance belongs to a particular class k:
  - First it traverses the tree to find the leaf node for this instance
  - Then it returns the ratio of training instances of class k in this node
  - Example: Suppose you have found a flower whose petals are 5 cm long and 1.5 cm wide.
    - The corresponding leaf node is the depth-2 left node
    - The Decision Tree should output the following probabilities:

    0% for Iris-Setosa (0/54), 90.7% for Iris-Versicolor (49/54), and 9.3% for Iris-Virginica (5/54).

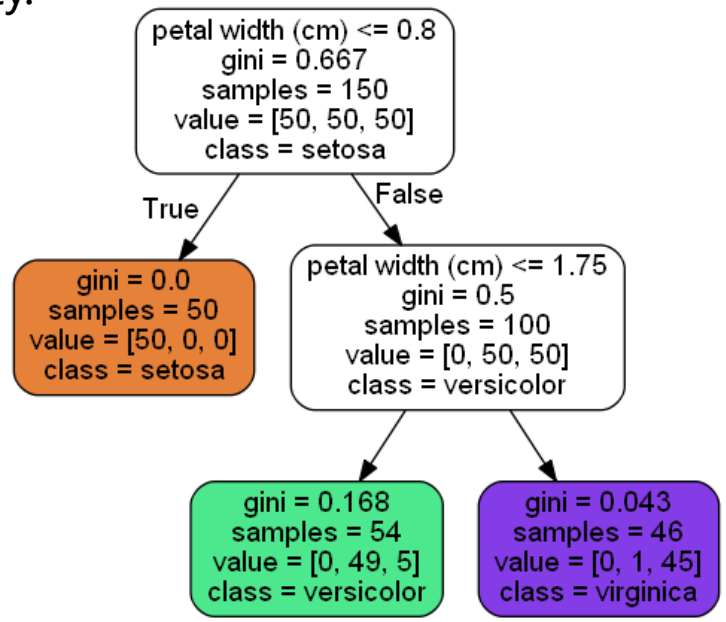
  - It should output Iris-Versicolor (class 1) since it has the highest probability.

```
tree_clf.predict_proba([[5, 1.5]])
```

```
array([[0.        , 0.90740741, 0.09259259]])
```

```
tree_clf.predict([[5, 1.5]])
```

```
array([1])
```



CART Algorithm (classification)

- Scikit-Learn uses the CART algorithm to train Decision Trees
  - 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")
  - After that, it splits the subsets using the same logic, then the sub-subsets and so on, recursively.
  - It stops recursing once it reaches the maximum depth (defined by the max_depth hyperparameter), or if it cannot find a split that will reduce impurity
- How does it choose *k* and t_k ?
  - It searches for the pair (k, t_k) that produces the purest subsets

CART cost function for classification

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

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

CART Algorithm (regression)

- The CART algorithm works mostly the same way as earlier, except that instead of trying to split the training set to minimize impurity, it tries to split the training set in a way that minimizes the MSE (mean squared error)

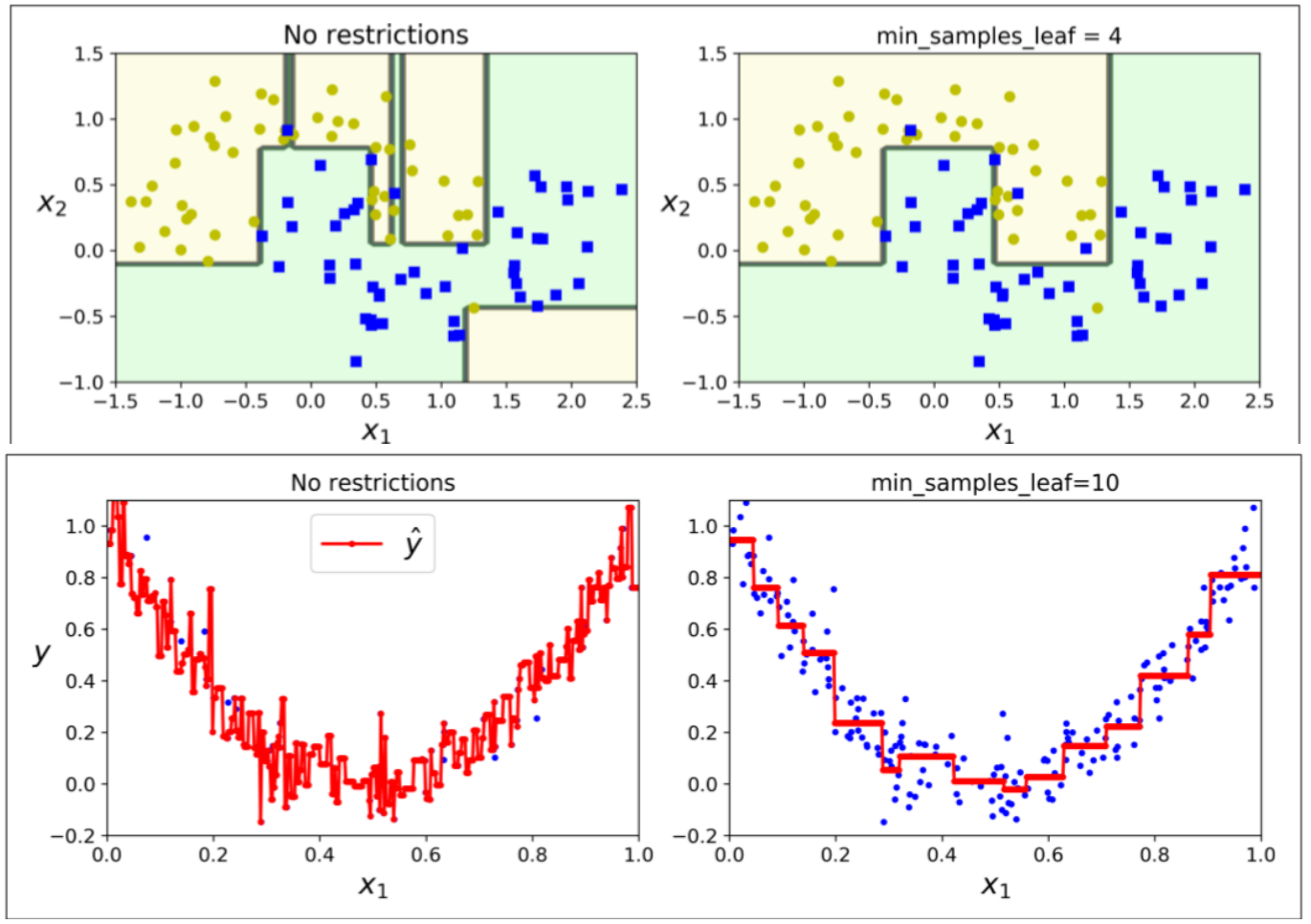
CART cost function for regression

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

$$\begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Regularization Hyperparameters

- Decision Trees make very few assumptions about the training data (as opposed to linear models, which obviously assume that the data is linear, for example)
- If left unconstrained, the tree structure will adapt itself to the training data, fitting it very closely, i.e. overfitting



Regularizing a Decision Tree Classification / Regression

Regularization Hyperparameters

- maxdepth: Reducing max_depth will regularize the model and thus reduce the risk of overfitting
- min_samples_split: the minimum number of samples a node must have before it can be split
- min_samples_leaf: the minimum number of samples a leaf node must have
- min_weight_fraction_leaf: same as min_samples_leaf but expressed as a fraction of the total number of weighted instances
- max_leaf_nodes: maximum number of leaf nodes
- max_features: maximum number of features that are evaluated for splitting at each node

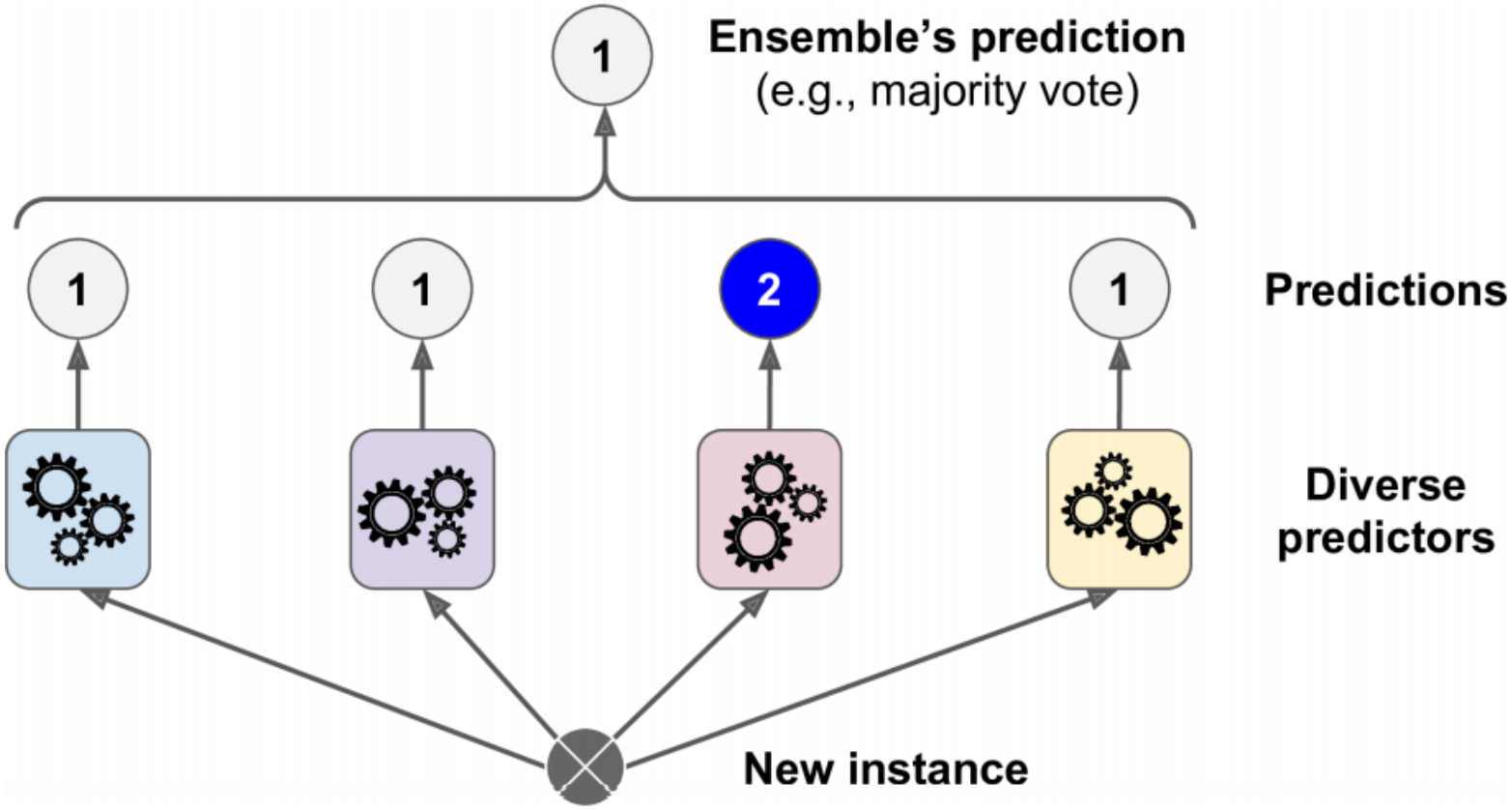
Introduction to Ensemble learning

Motivation: Ensemble is better than single

- Analogies
  - **Wisdom of the crowd**: The collective opinion of a group of individuals tends to be better than that of a single expert (三个臭皮匠，赛过诸葛亮)
  - Elections combine voters' choices to pick a good candidate
  - Students working together on one project
- Intuition
  - Individuals make mistakes but the "majority" may be less likely to
  - Individuals often have partial knowledge; a committee can pool expertise to make better decisions
- Applications
  - Multi-source data integration
  - Biomarker selection
  - Winner solutions in machine learning competitions

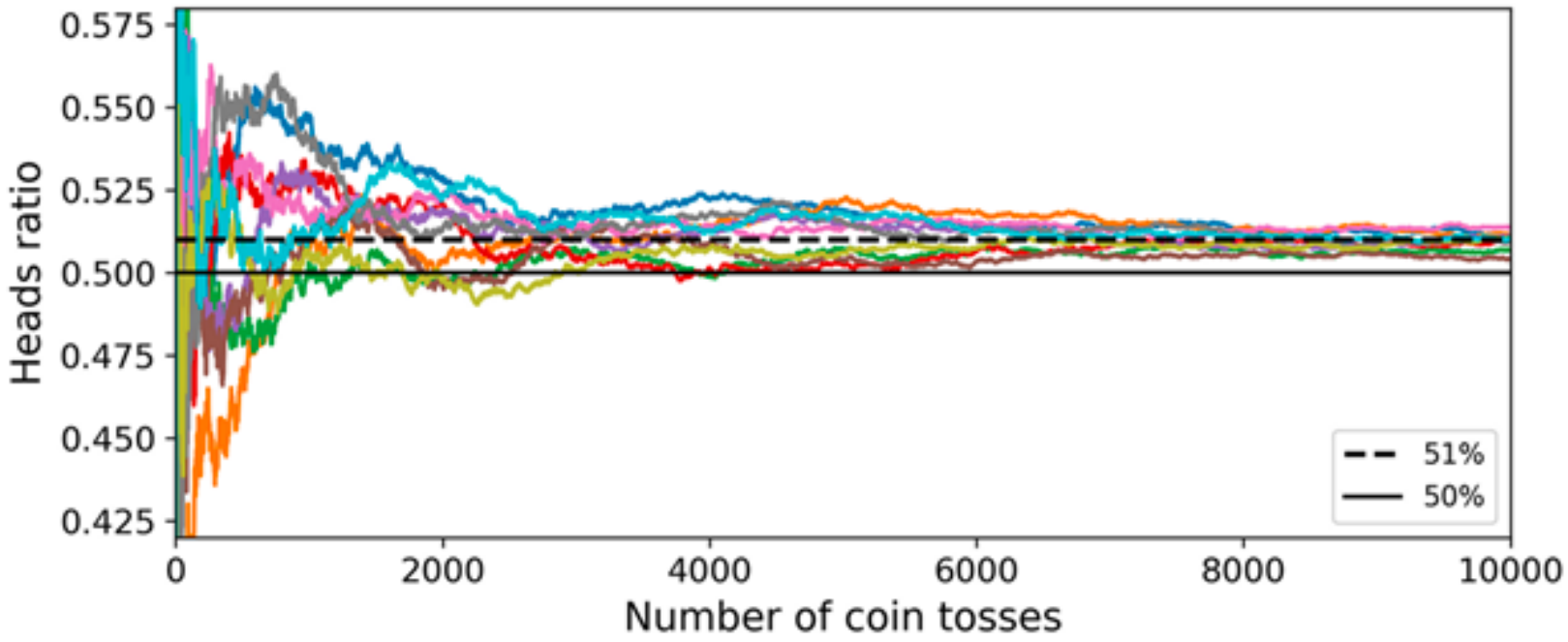
Ensemble learning

- **Example (Voting Classifiers)**: Suppose you have trained a few classifiers, a very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes



Why dose ensemble learning work?

- **Key idea**: Even if each classifier is a weak learner (meaning it does only slightly better than random guessing), the ensemble can still be a strong learner (achieving high accuracy)

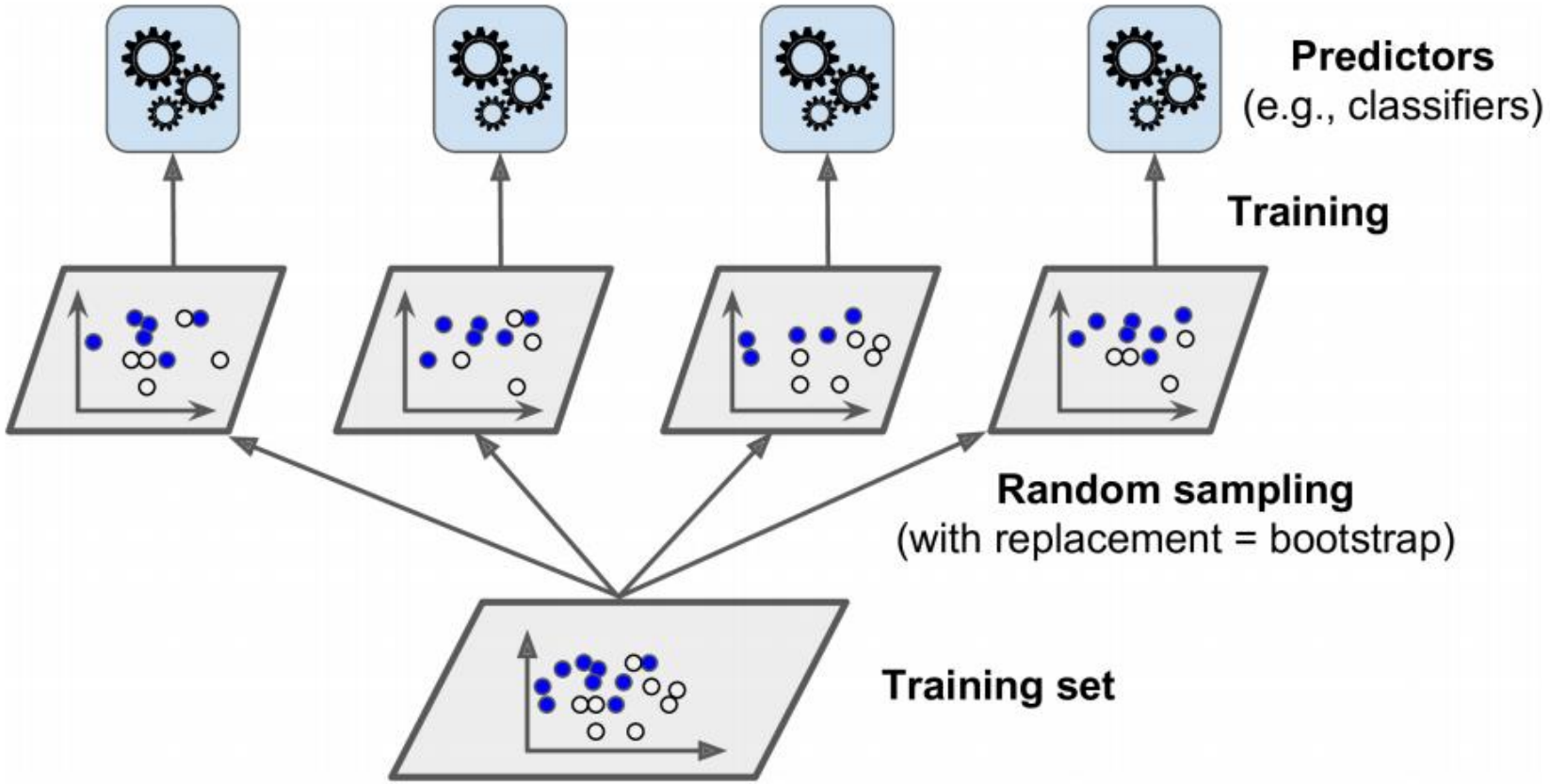


Toss a slightly biased coin that has a 51% chance of coming up heads

- **The law of large numbers**: e.g. as you keep tossing the coin, the ratio of heads gets closer and closer to the probability of heads (51%)

Bagging and Pasting

- To get a diverse set of classifiers, one way is to use the same training algorithm for every predictor, but to train them on different random subsets of training data
- Bagging: sampling with replacement
- Pasting: sampling without replacement

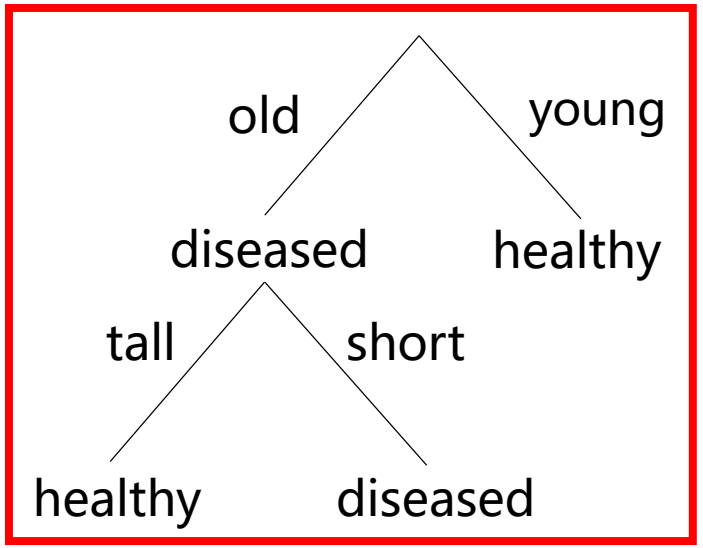


Random Forest

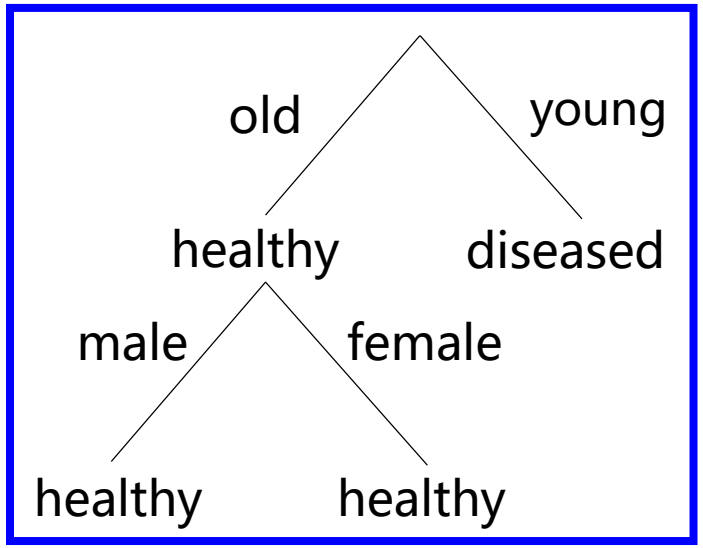
Intuition of Random Forests

- A **Random Forest** is an ensemble of Decision Trees, generally trained via the bagging method
- Example (Health Diagnostic)
  - New sample: old, retired, male, short
  - Tree predictions will be: diseased, healthy, diseased
  - Conclusion (Majority vote): diseased

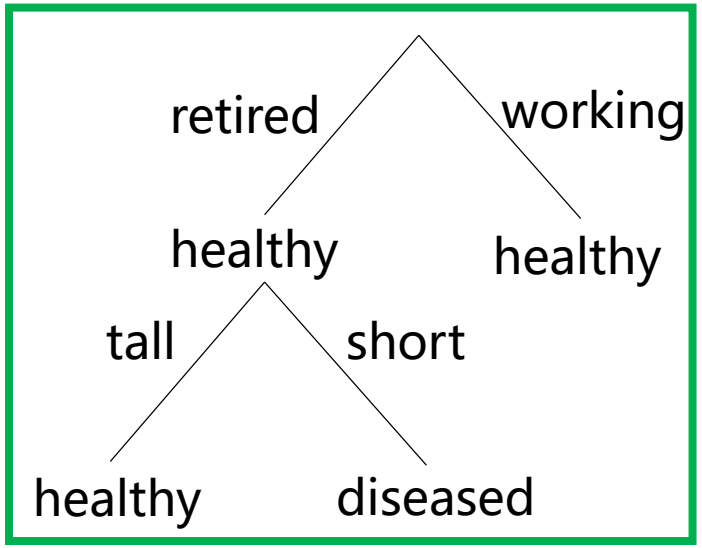
Tree 1



Tree 2

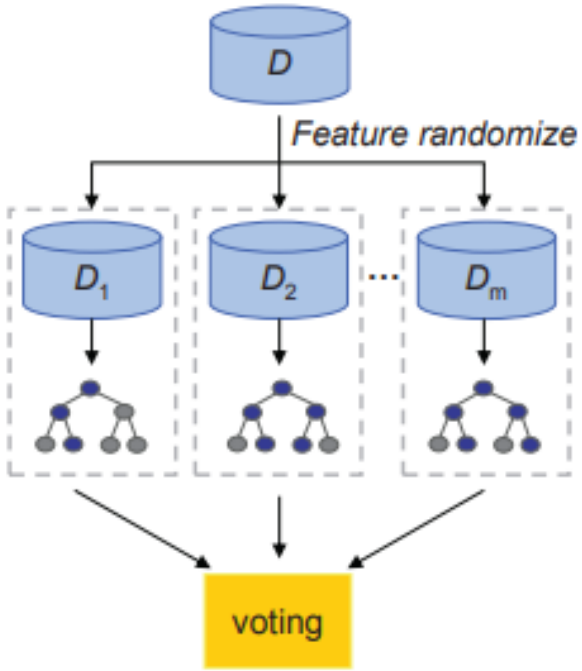


Tree 3



The Random Forests Algorithm


```
for i ← 1 to B by 1 do
    Draw a bootstrap sample with size N from the training data;
    while node size != minimum node size do
        randomly select a subset of m predictor variables from total p;
        for j ← 1 to m do
            if j-th predictor optimizes splitting criterion then
                split internal node into two child nodes;
                break;
            end
        end
    end
end
return the ensemble tree of all B sub-trees grown in the for loop;
```



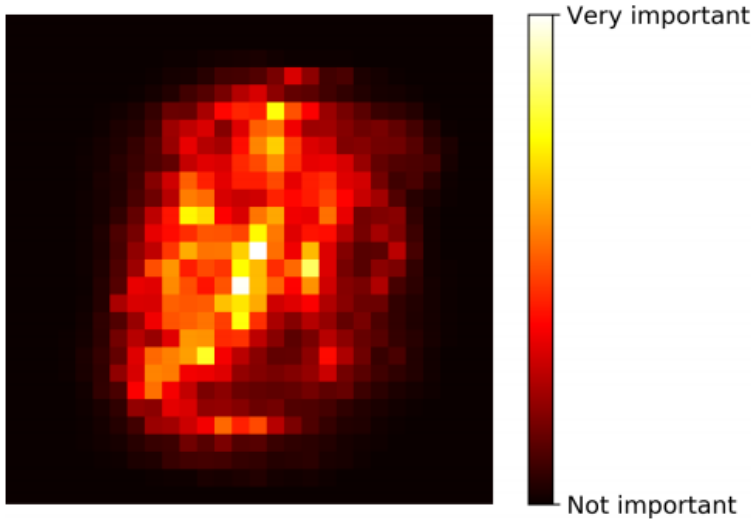
- Prediction
  - Classification Problems

  $\hat{f}(x)$ = majority vote of all predicted classes over B trees
  - Regression Problems

  $\hat{f}(x)$ = sum of all sub-tree predictions divided over B trees

Properties of Random Forests

- Combines two randomization strategies
  - Select random subset of the dataset to learn decision tree, e.g. select n = 100 random trees
  - Select random subset of features, e.g. select $\sqrt{m}$ features
- Random forests are used to estimate the importance of features (by measuring how much the tree nodes that use that feature reduce impurity on average)
- Random forests in Scikit-Learn




```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Tree numbers

Why is Random Forests popular?

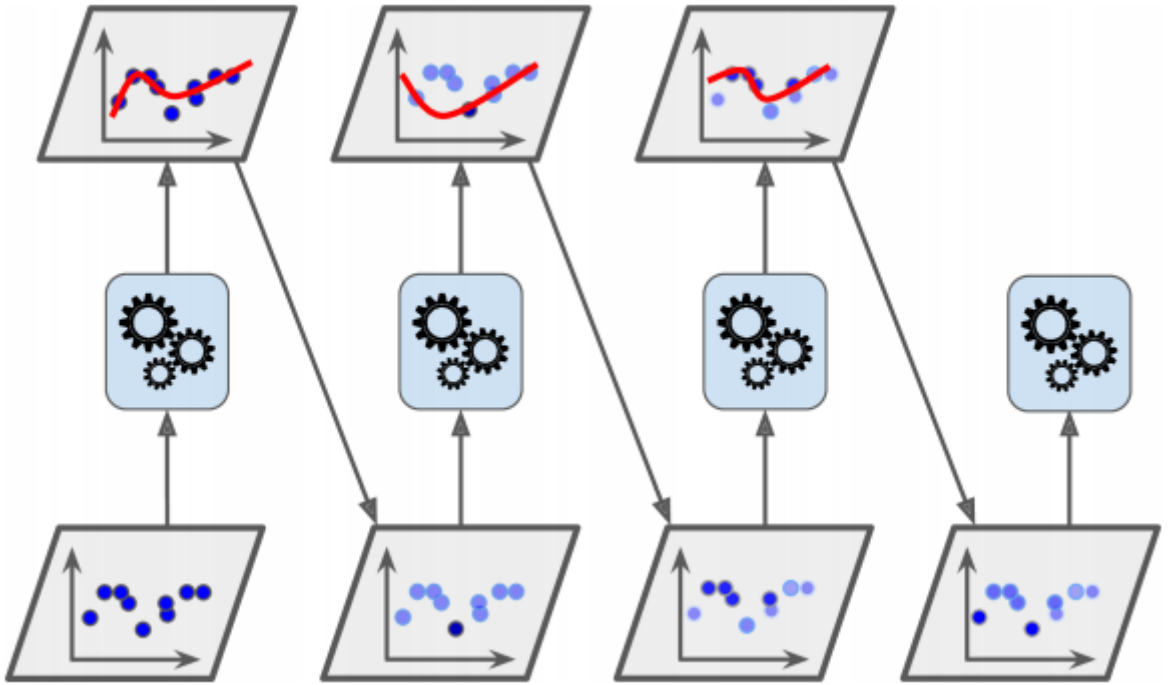
- No overfitting
  - Use of multiple trees reduce the risk of overfitting
  - Training time is less
- High accuracy
  - Runs efficiently on large database
  - For large data, it produces highly accurate predictions
- Estimates missing data
  - Random Forests can maintain accuracy when a large proportion of data is missing
- Easy to turn parameters
- Provide a reliable feature importance estimate
  - measure feature importance by calculating the 'mean decrease accuracy'

Random Forest is widely used in various fields of biological and biomedical research

Boosting and Stacking

Boosting

- The general idea of most **boosting** methods is to train predictors sequentially, each trying to correct its predecessor.
- **Example (Adaboost)**: Correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted (by increasing The relative weight of misclassified training instances)



Adaboost

- Drawback: it cannot be parallelized, since each predictor can only be trained after the previous predictor has been trained and evaluated
- Each instance weight w(i) is initially set to 1/m. A first predictor is trained and its weighted error rate r_j is computed on the training set

$$r_j = \frac{\sum_{i=1}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}} \quad \text{where } \hat{y}_j^{(i)} \text{ is the } j^{\text{th}} \text{ predictor's prediction for the } i^{\text{th}} \text{ instance.}$$

- The predictor's weight α_j is then computed

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

- Next the instance weights are updated

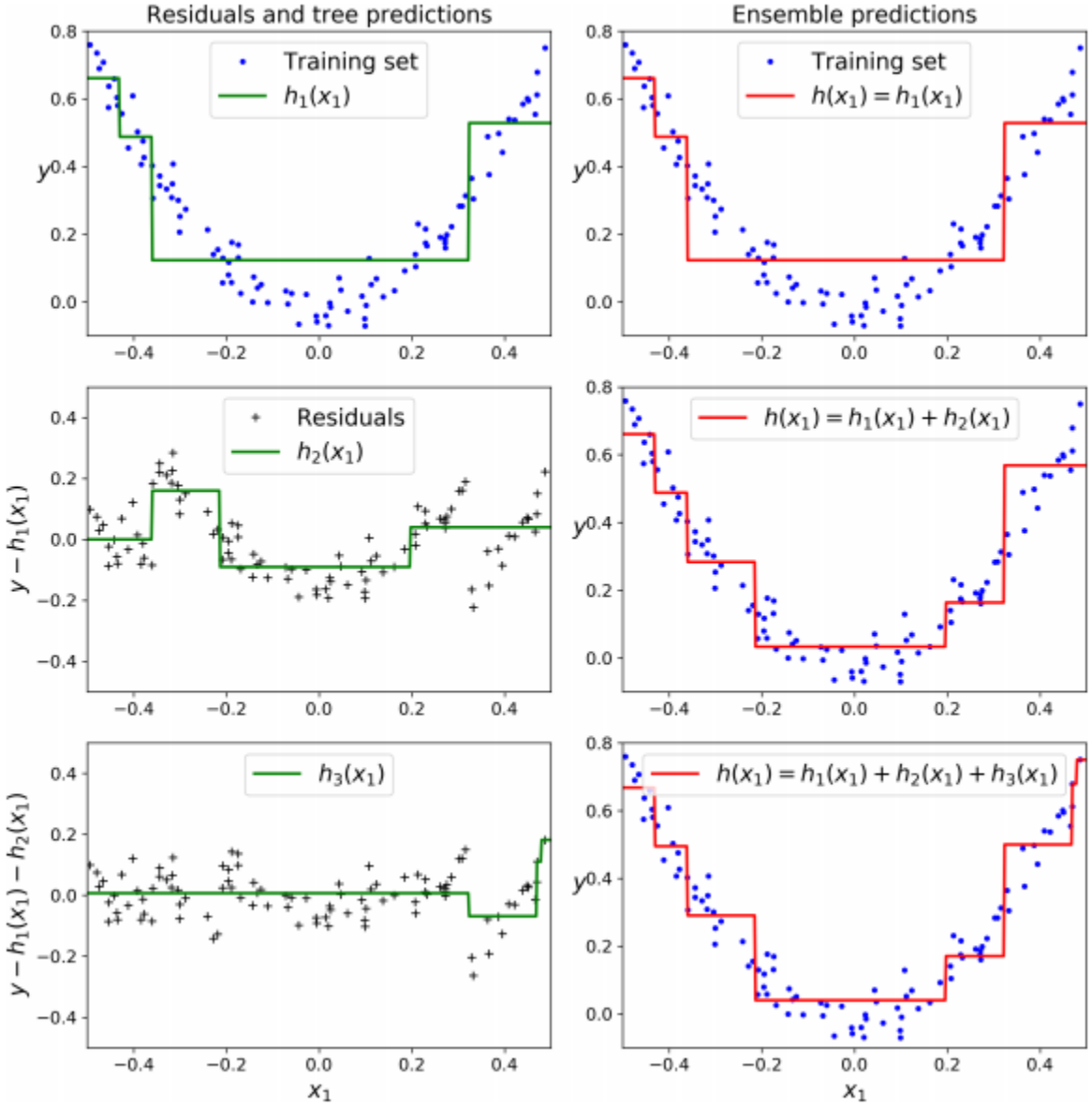
$$\text{for } i = 1, 2, \cdots, m$$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \hat{y}_j^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases}$$

The misclassified instances are boosted

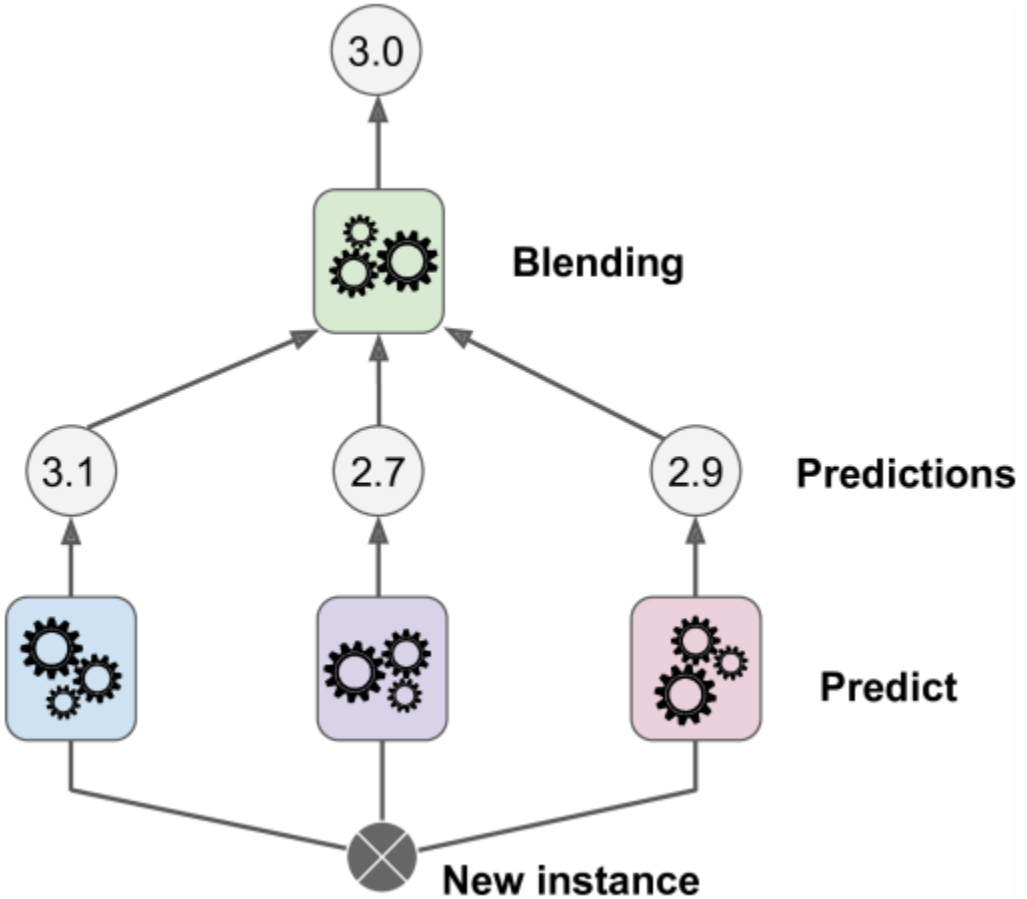
Gradient Boosting

- Instead of tweaking the instance weights at every iteration like AdaBoost does, **Gradient Boosting** tries to fit the new predictor to the residual errors made by the previous predictor
- **Example (Gradient Boosted Regression Trees (GBRT))**



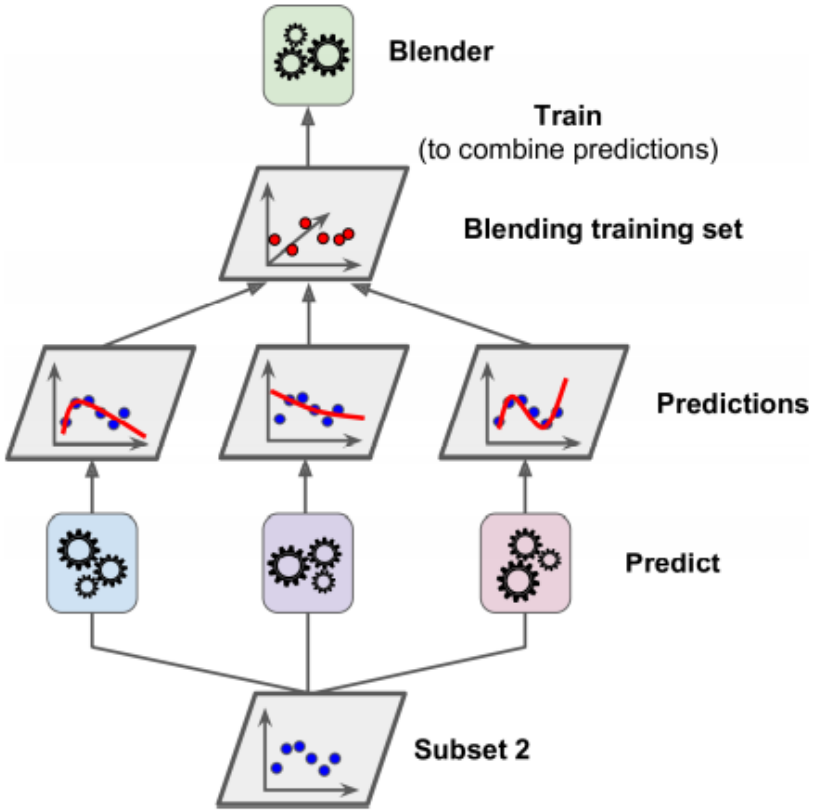
Stacking

- **Stacking** is based on a simple idea: instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, why don't we train a model to perform this aggregation (called a blender)?



Training a blender

- To train the blender, a common approach is to use a hold-out set
  - First, the training set is split in two subsets. The first subset is used to train the predictors in the first layer
  - Next, the first layer predictors are used to make predictions on the second (held-out) set
  - We can create a new training set using these predicted values as input features, and keeping the target values.
  - Finally, the blender is trained on this new training set



Summary

- The concept and architecture of Decision Tree (DT)
- How to visualize a DT
- The algorithm of Classification And Regression Tree (CART)
- The concept and methods of ensemble learning