哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Big data technology and Practice

李春山

2020年9月22日

内容提要

Chapter 9 : ML - Decision Trees

Chapter 10: Ensemble Learning and Random Forests

Chapter 9 : ML – Decision Trees

- Decision Trees are very powerful ML algorithms
- Can perform both classification and regression tasks, and even multi-output tasks,
- Capable of fitting complex datasets.
- Fundamental components of Random Forests (to be discussed in next Chapter), which are among the most powerful Machine Learning algorithms available today.

Chapter 9 : ML – Decision Trees

- We discuss this topic by using Scikit-Learn and associated samples in this lecture.

- In Lab class, a number of examples of Spark API based Decision trees and Random Forest will be introduced.

1. Training and Visualizing a Decision Tree

- **To understand Decision Trees, let's just build one and take a look at how it makes predictions.**

- **1). Dataset**
- This is a famous dataset that contains the sepal and petal length and width of 150 iris flowers of three different species: Iris-Setosa, Iris-Versicolor, and Iris-Virginica

`sklearn.datasets.` **`load_iris`** (*return_X_y=False*)

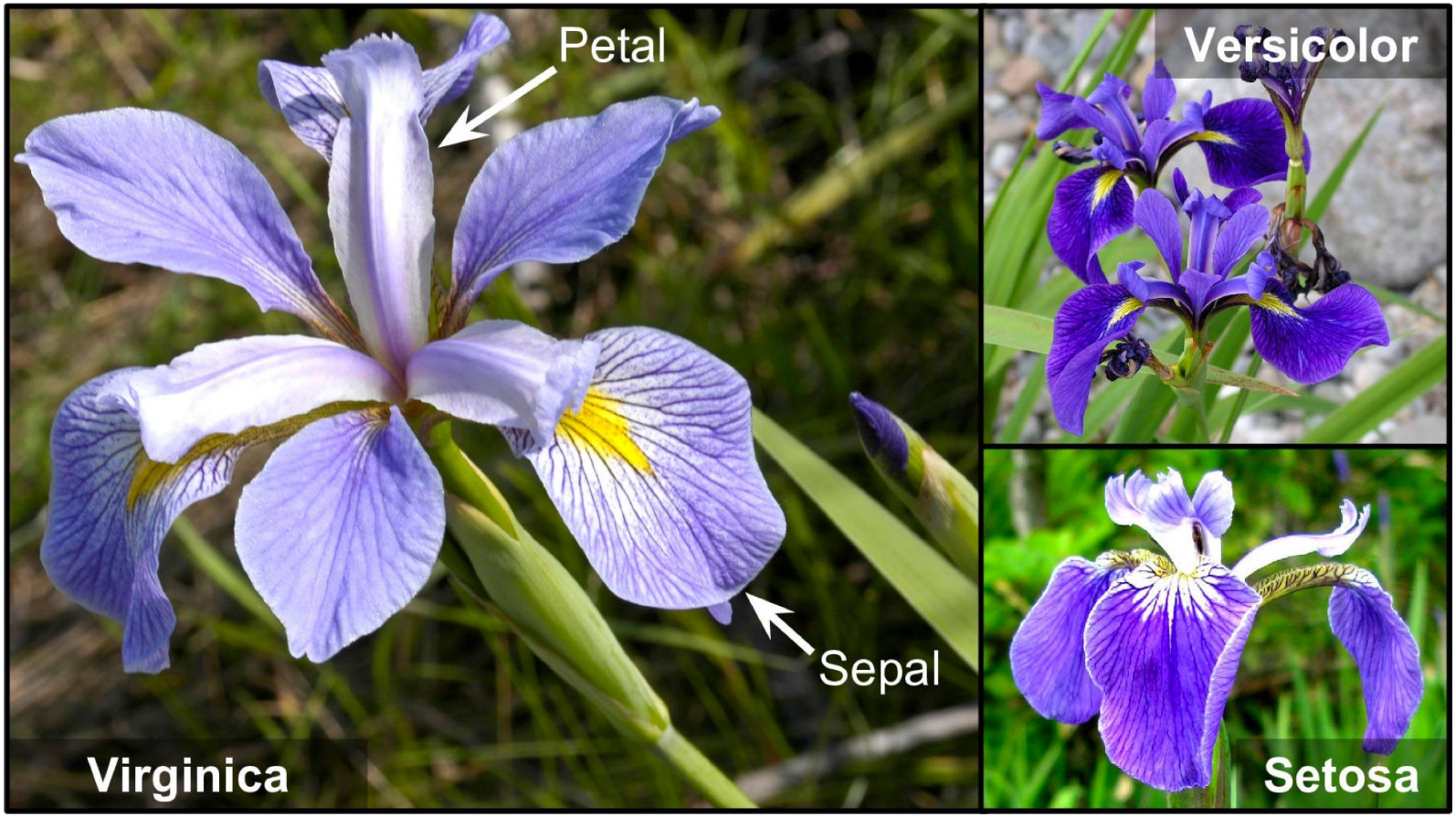
Load and return the iris dataset (classification).

The iris dataset is a classic and very easy multi-class classification dataset.

Classes	3
Samples per class	50
Samples total	150
Dimensionality	4
Features	real, positive

Dataset

- Flowers of three iris plant species



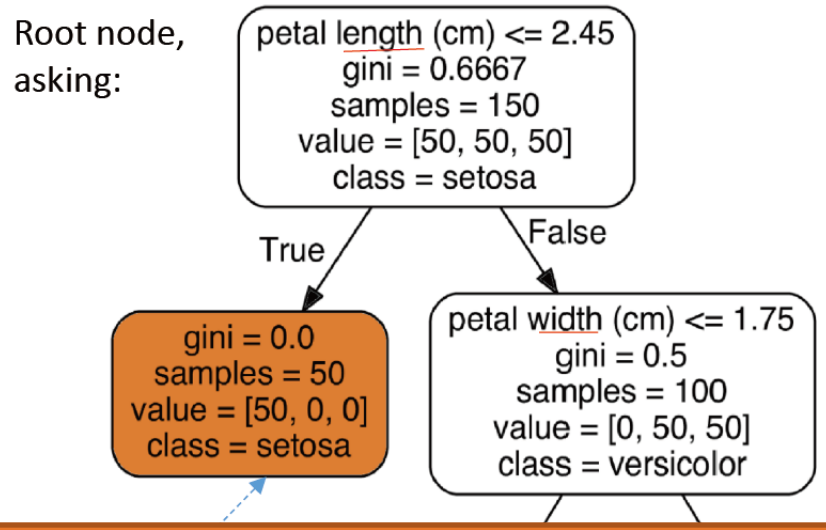
Dataset

- Suppose that we have built or trained a “Decision Tree” in next slide and look at how to use it for prediction or classification.

- The details of the Classification And Regression Tree (CART) algorithm for building the tree will be presented in later section.

2). Make Predictions

- Your first decision tree looks like:
- Let's see how the tree represented here makes predictions.
- Suppose you find an iris flower and you want to classify it. Starting at root node:



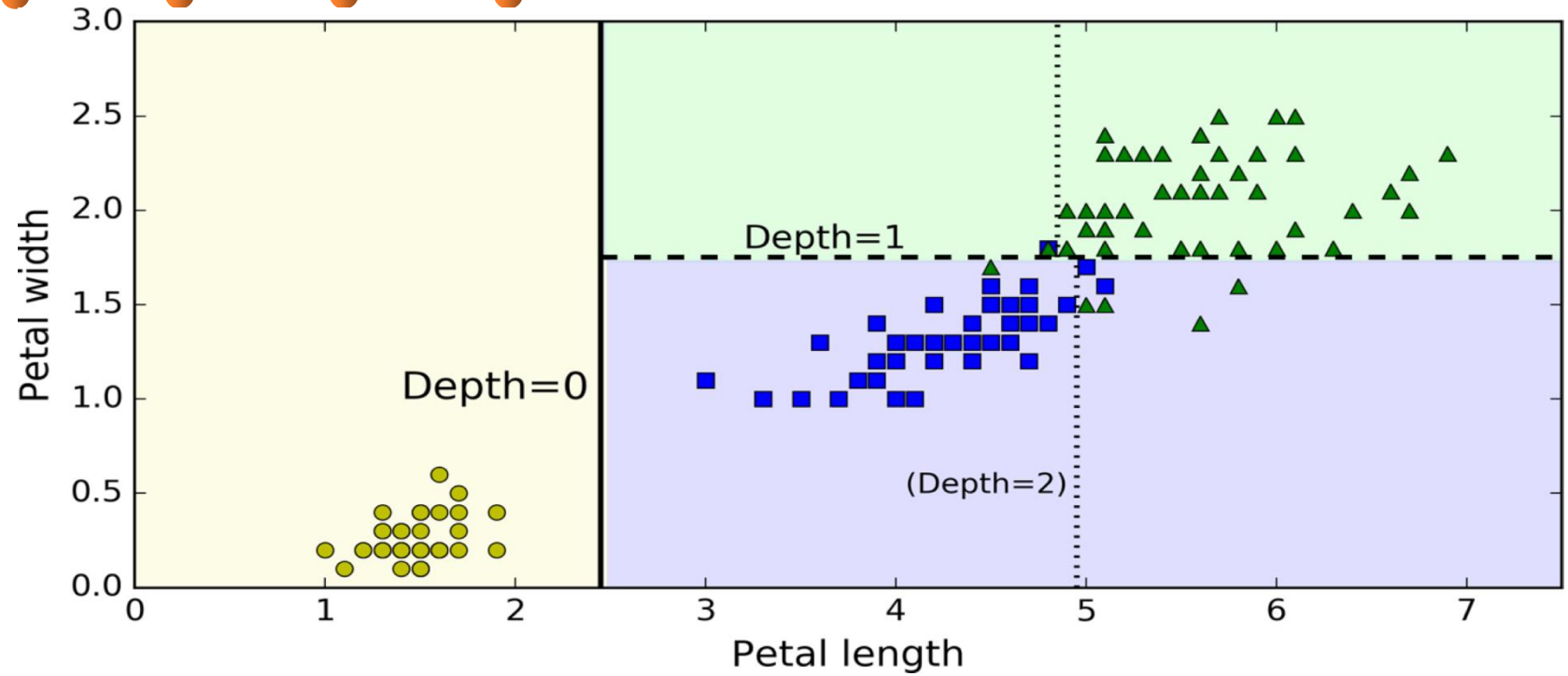
The diagram shows how the training algorithm computes the gini score Gi of the ith node.
For example, the depth-2 left node has a gini score equal to 1 – (0/54)2 – (49/54)2 – (5/54)2 ≈ 0.168.

Gini measures its impurity. A node is "pure" if gini = 0

$G_i = 1 -$

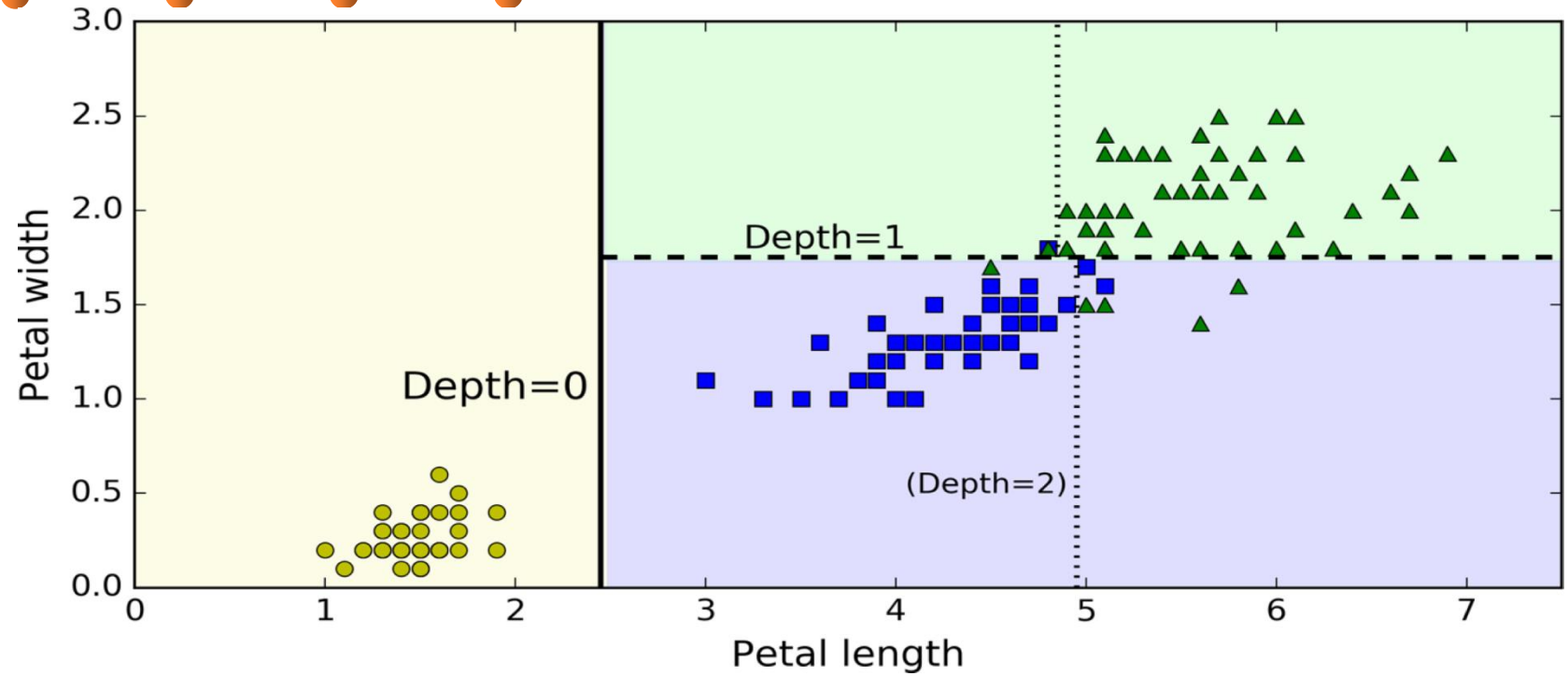
- $p_{i,k}$ is th

2). Make Predictions



- It shows this Decision Tree's decision boundaries. The thick vertical line represents the decision boundary of the root node (depth 0): petal length = 2.45 cm. Since the left area is pure (only Iris-Setosa), it cannot be split any further.

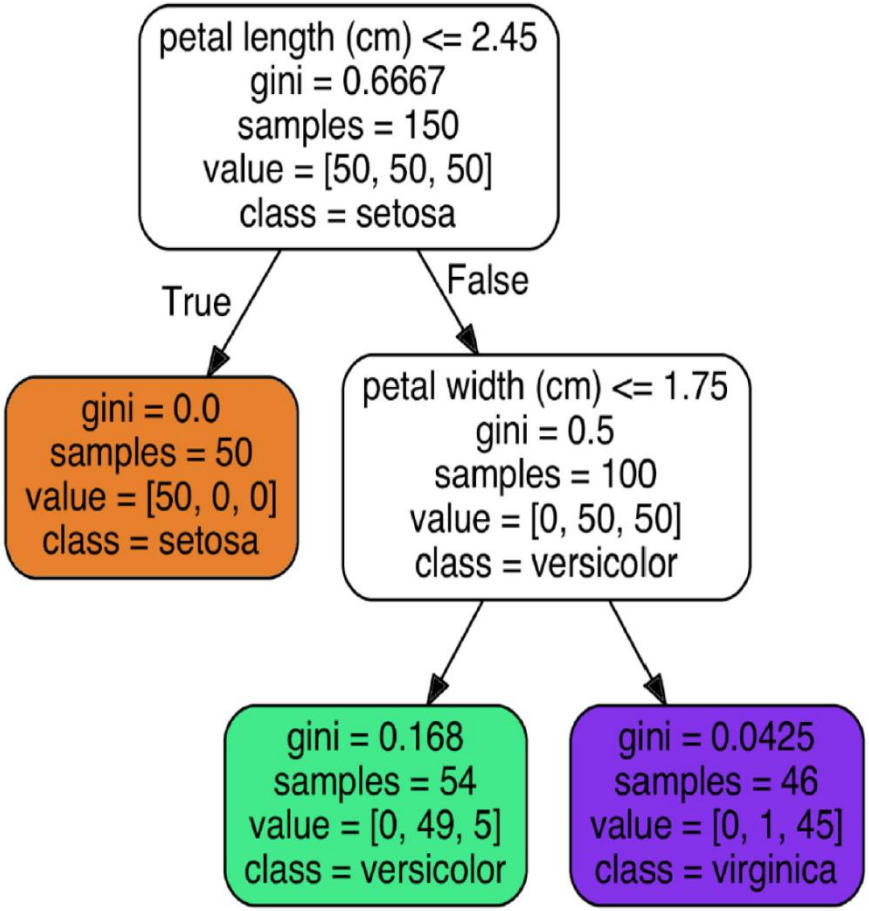
2). Make Predictions



- The right area is impure, so the depth-1 right node splits it at petal width = 1.75 cm (represented by the dashed line). Since max_depth was set to 2, the Decision Tree stops right there.
- However, if you set max_depth to 3, then the two depth-2 nodes would each add another decision boundary (represented by the dotted lines).

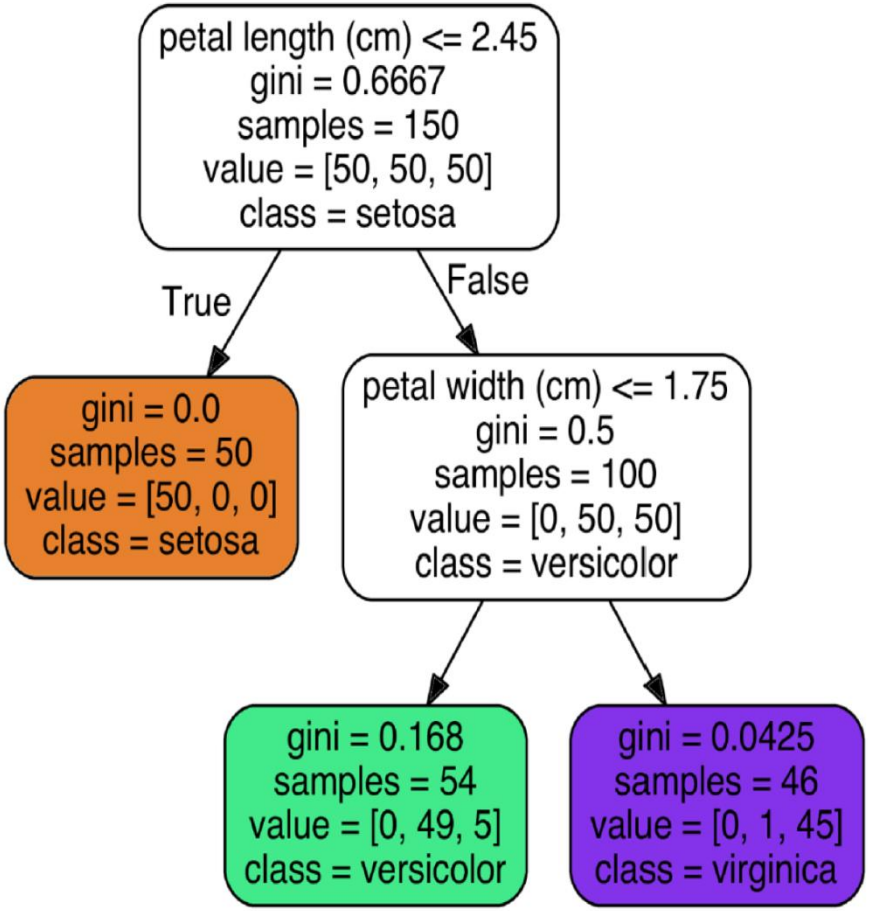
3). Estimating Class Probabilities

- A Decision Tree can also estimate the probability that an instance belongs to a particular class k:
- It traverses the tree to find the leaf node for this instance, and then it returns the ratio of training instances of class k in this node.
- For example, for a flower whose petals are 5 cm long and 1.5 cm wide. The corresponding leaf node is the depth-2 left node.



3). Estimating Class Probabilities

- The 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).
- And of course if you ask it to predict the class, it should output Iris-Versicolor (class 1) since it has the highest probability.



4). The Algorithm for Building Decision Trees

- Scikit-Learn uses the Classification And Regression Tree (CART) algorithm to build or train Decision Trees (also called "growing" trees).
- The idea of 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 cost function that the algorithm tries to minimize (next slide)

CART cost function for classification

m_{left} (m_{right}) is the number of samples on left (right) subset.

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

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

- Once it has successfully split the training set in two, it splits the subsets using the same logic, then the sub-subsets and so on, recursively.
- It stops recurring once it reaches the maximum depth (defined by the max_depth hyperparameter), or if it cannot find a split that will reduce impurity.
- Once a Classification And Regression Tree (CART) is trained (or built), we can use it for classification.

2. Regression

- Decision Trees are also capable of performing regression tasks.
- In Lab class, Let's build a regression tree using Scikit-Learn's DecisionTreeRegressor class.

内容提要

Chapter 9 : ML - Decision Trees

Chapter 10: Ensemble Learning and Random Forests

1. About Ensemble Learning

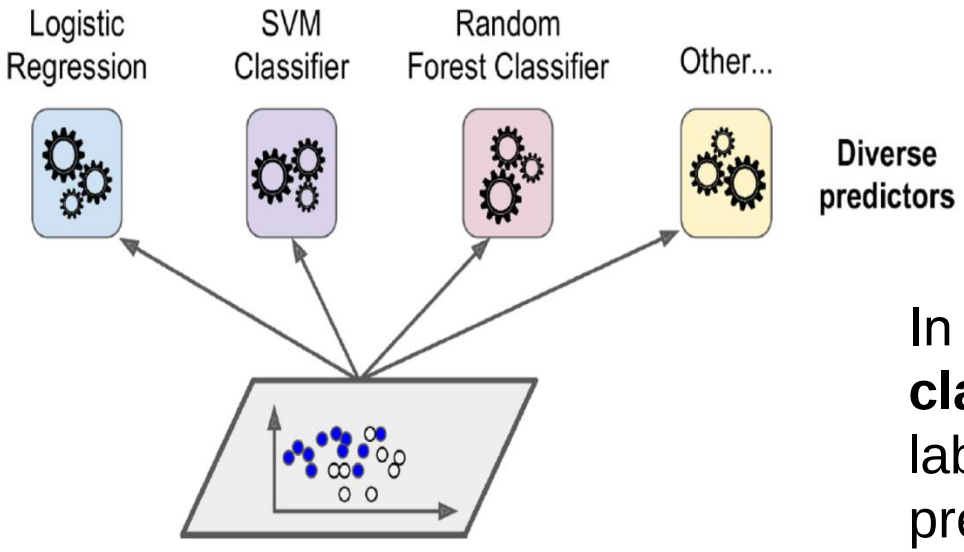
- **Suppose you ask a complex question to thousands of random people, then aggregate their answers. In many cases you will find that this aggregated answer is better than an expert's answer. This is called the wisdom of the crowd.**

- Similarly, the predictions of a group of predictors (such as classifiers or regressors) often get better predictions than with the best individual predictor.

- A group of predictors is called an ensemble . So, this technique is called Ensemble Learning , and an Ensemble Learning algorithm is called an Ensemble method.

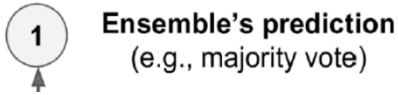
2. Voting Classifiers

- As one of the ensemble methods, it gets a diverse set of classifiers to use very different training algorithms, called “Voting Classifiers”.
- Suppose you have trained a few classifiers as below, each one achieving about 80% accuracy.



In “hard voting”, (or **majority-vote classifier)** we predict the final class label as the class label that has been predicted most frequently by the classification models.

2. Voting Classifiers



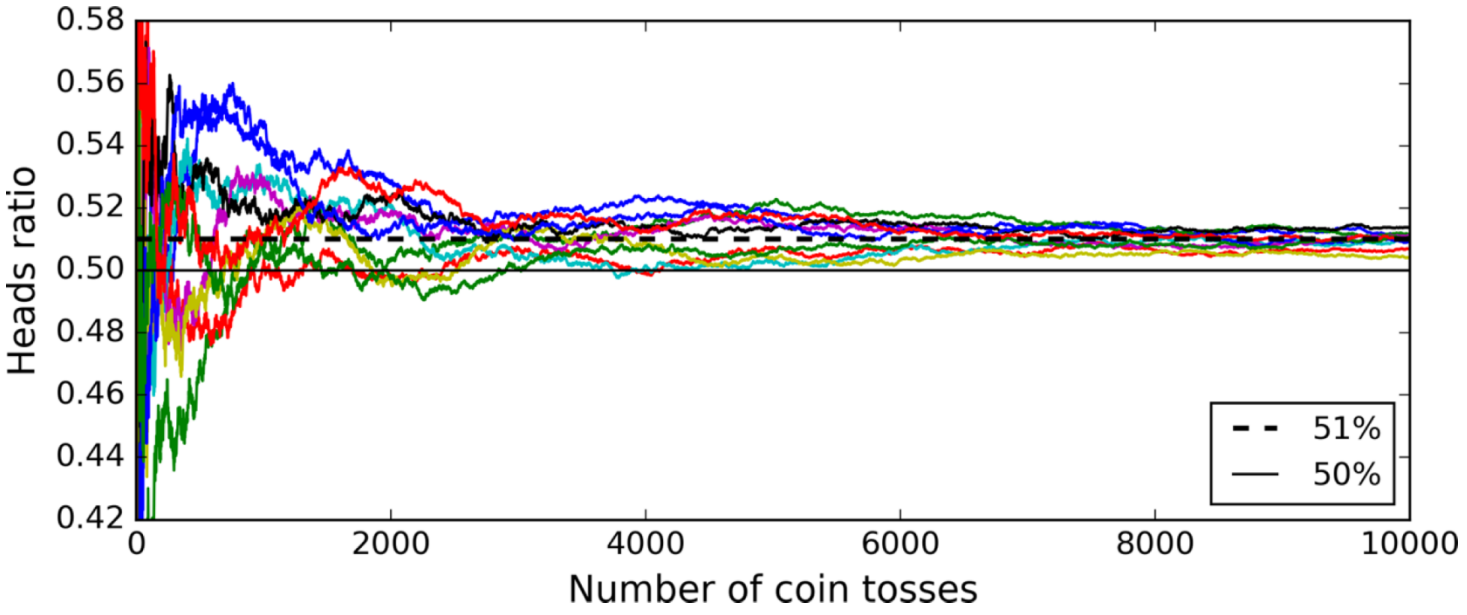

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

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)
```

Let's look at each classifier's accuracy on the test set:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
>>>     clf.fit(X_train, y_train)
>>>     y_pred = clf.predict(X_test)
>>>     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
LogisticRegression 0.864
RandomForestClassifier 0.872
SVC 0.888
VotingClassifier 0.896
```

2. Voting Classifiers

- Somewhat surprisingly, this voting classifier often achieves a higher accuracy than the best classifier in the ensemble.
- In fact, 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), provided there are a sufficient number of weak learners and they are sufficiently diverse.



"The law of large numbers"

2. Voting Classifiers

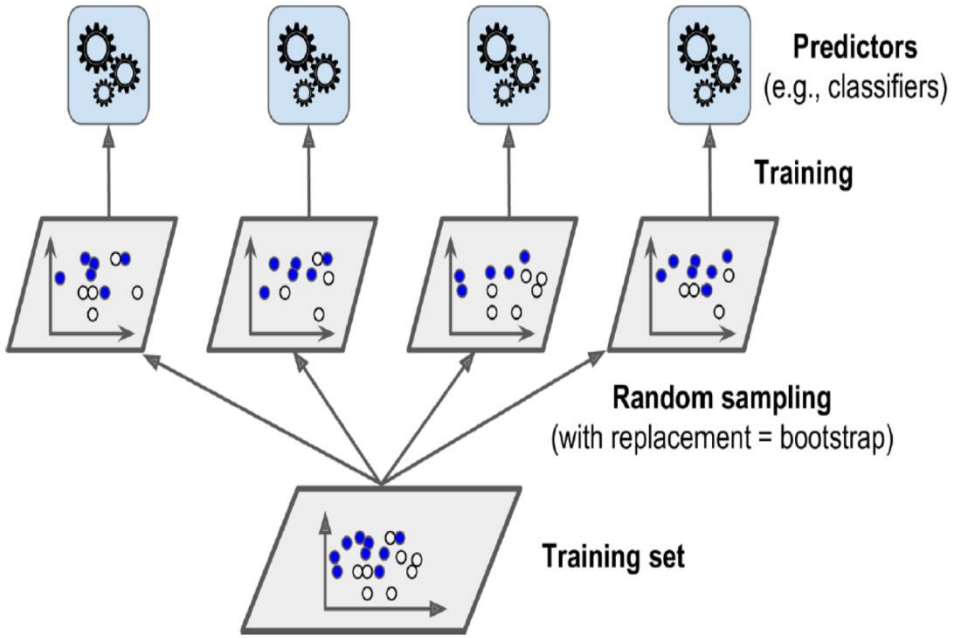
- If all classifiers are able to estimate class probabilities (i.e., they have a predict_proba() method), then you can tell Scikit-Learn to predict the class with the highest class probability, averaged over all the individual
- This is called soft voting.
- It often achieves higher performance than hard voting because it gives more weight to highly confident votes.
- In coding, all you need to do is replace voting="hard" with voting="soft“ and ensure that all classifiers can estimate class probabilities.
- See their details: https://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/

3. Bagging and Pasting

- Another approach is to use the same training algorithm for every predictor, but to train them on different random subsets of the training set.
- When sampling is performed with replacement, this method is called bagging.
- When sampling is performed without replacement, it is called pasting.
- In other words, both bagging and pasting allow training instances to be sampled several times across multiple predictors, but only bagging allows training instances to be sampled several times for the same predictor.

3. Bagging and Pasting

- Once all predictors are trained, the ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors.
- Scikit-Learn offers a simple API for both bagging and pasting with the BaggingClassifier class or BaggingRegressor for regression.



4. Random Forests

- **The fundamental idea is to combine many decision trees into a single model. Individually, a prediction made by a decision tree may not be accurate, but combined together, the predictions will be closer to the mark on average.**
- It’s an ensemble learning method for classification, regression and other tasks.
- They operate by constructing a multitude of decision trees at training time and outputting
- Random forests correct for decision trees' habit of overfitting to their training set.

Random Forest Algorithm:

1. Randomly select "k" instances (or data input points) from total "m" instances. Where k < m

2. Among the "k" instance, calculate the node "d" using the best split point.

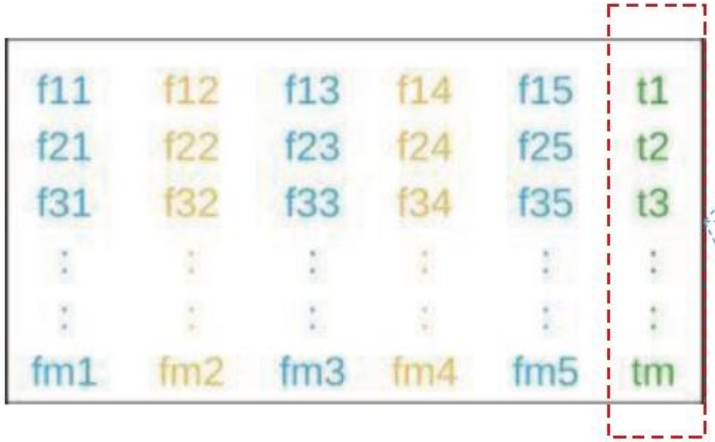
3. Split the node into child nodes using the best split.

4. Repeat 1 to 3 steps until a certain number of nodes has been reached.

5. Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees, which forms a "forest".

Training data set

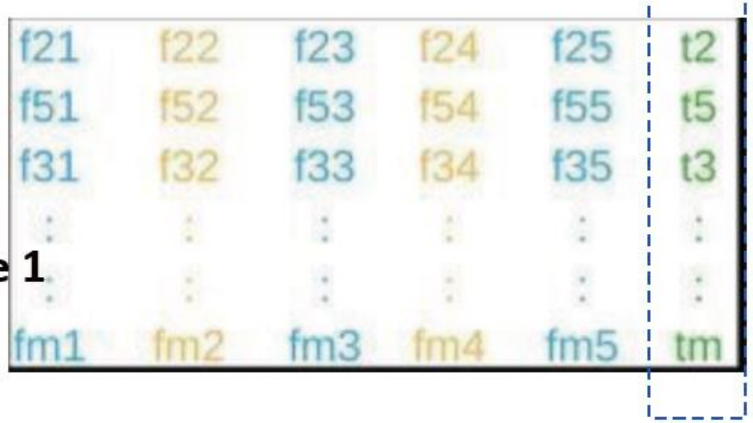
- **m** instances



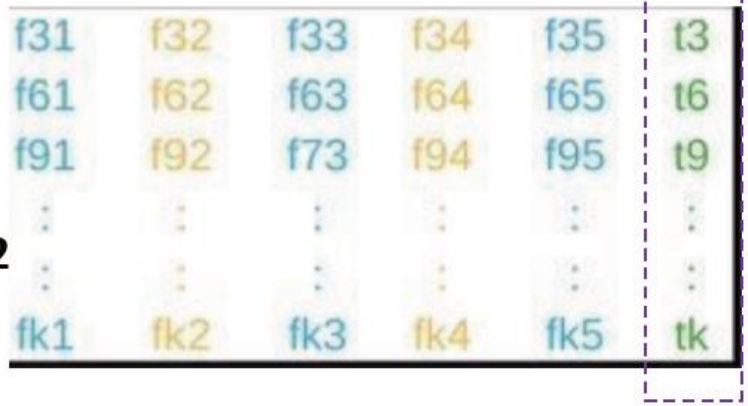
Usually, $k \approx \frac{2}{3} m$

Subset of the dataset, k < m

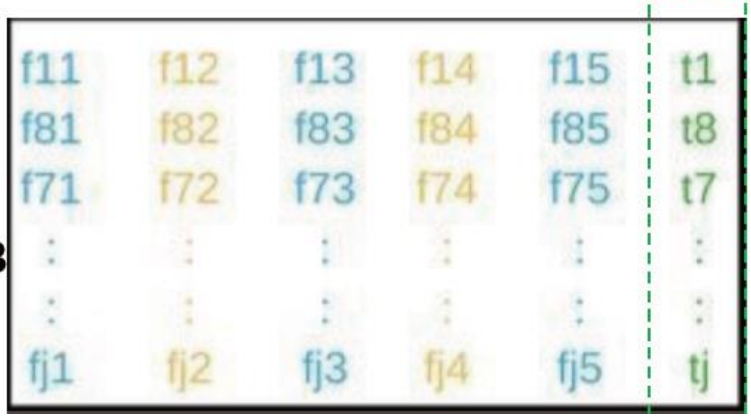
Sample 1



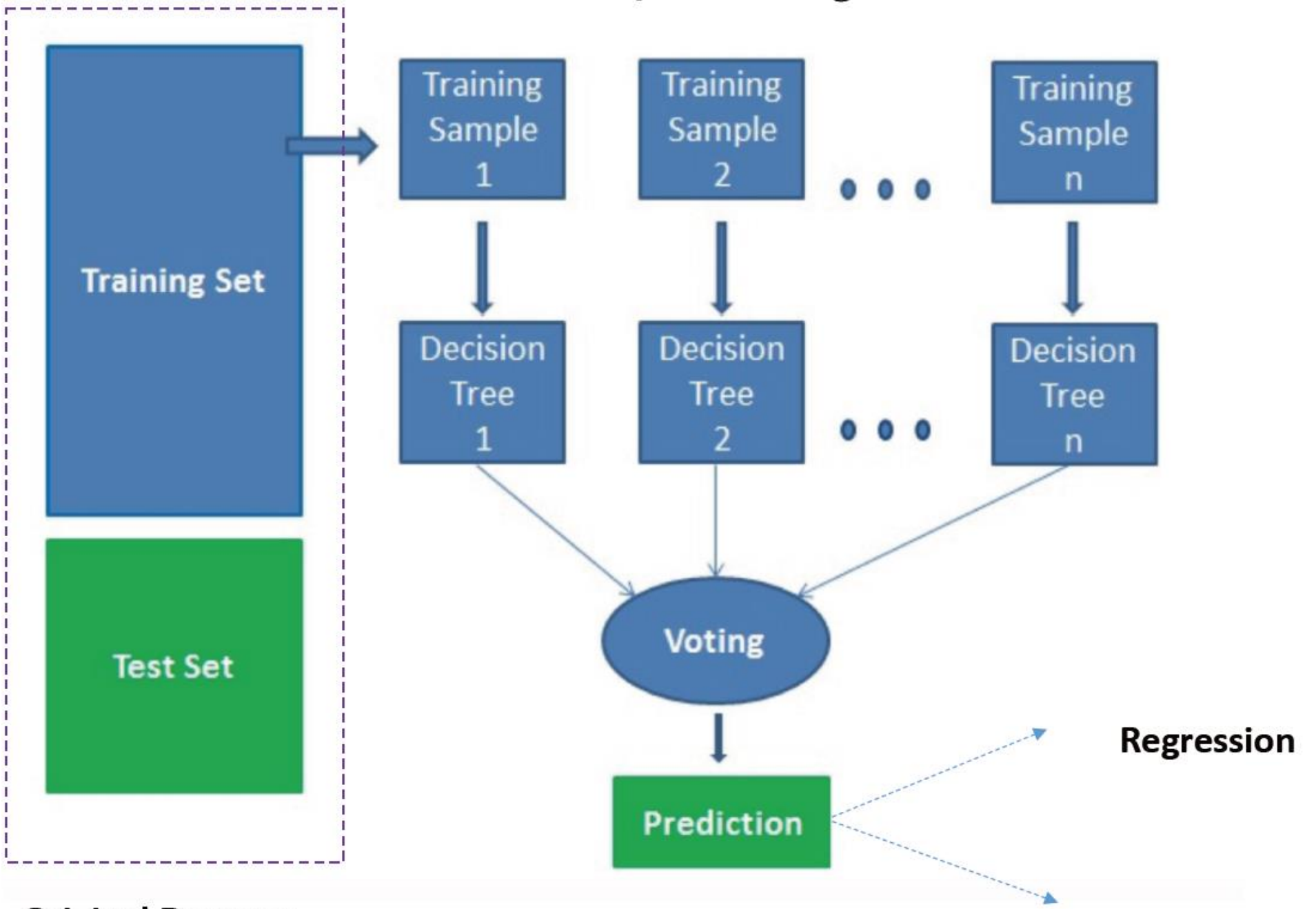
Sample 2



Sample 3



- Usually, $createing\ 50 \sim 100\ trees$



Random Forest Algorithm:

- Instead of building a BaggingClassifier and passing it a DecisionTreeClassifier, you can use the RandomForestClassifier class, which is more convenient and optimized for Decision Trees (similarly, there is a RandomForestRegressor class for regression tasks).
- The following code trains a Random Forest classifier with 500 trees (each limited to maximum 16 nodes), using all available CPU cores:

```
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)
```

- *Random Forests are very handy to get a quick understanding of what features actually matter, in particular if you need to perform feature selection.

5. Boosting

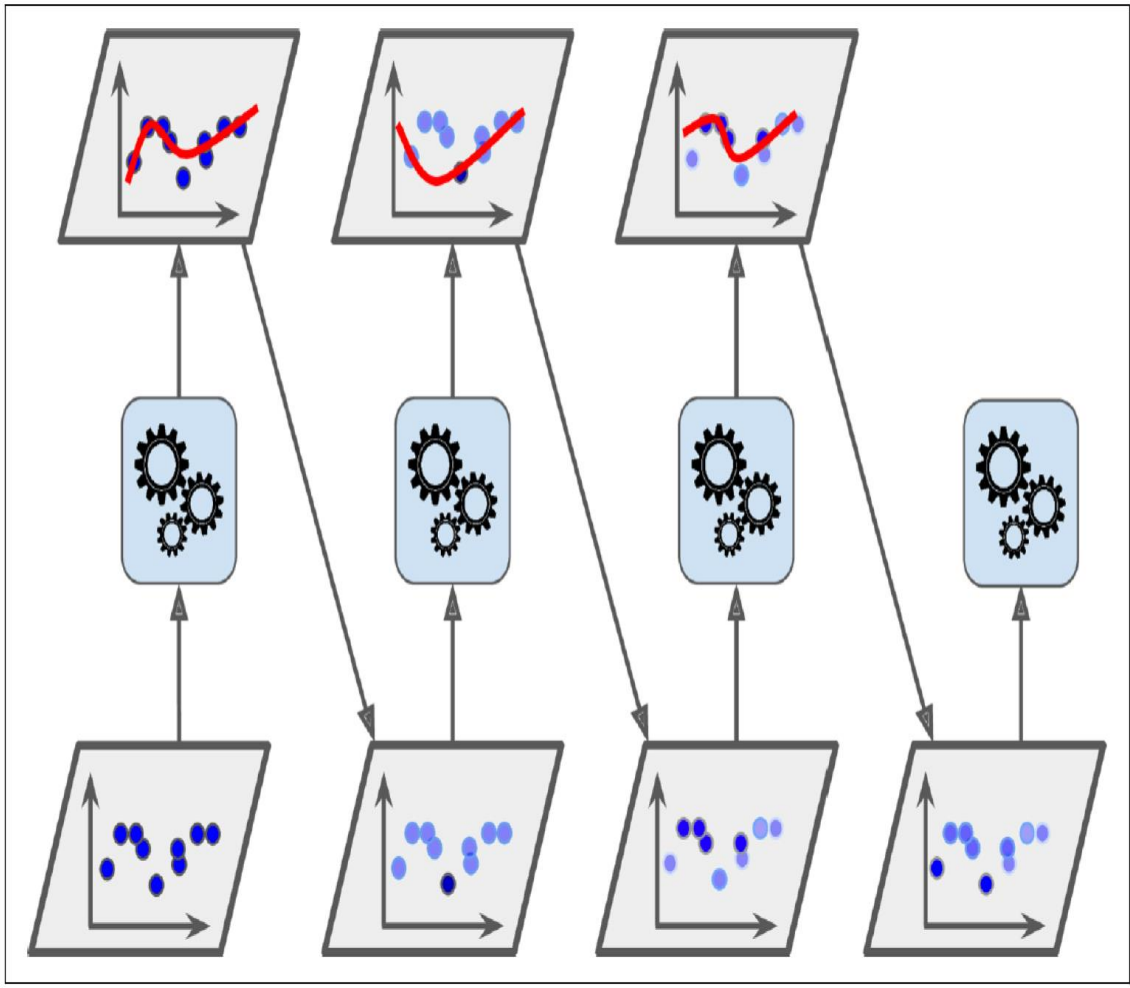
- Boosting refers to any Ensemble method that can combine several weak learners into a strong learner.
- The general idea is to train predictors sequentially, each trying to correct its predecessor.
- There are many boosting methods available, but by far the most popular are AdaBoost (short for Adaptive Boosting) and Gradient Boosting.

1). AdaBoosting

- A way for a new predictor to correct its predecessor is to pay a bit more attention to the training instances that the predecessor underfitted.
- This results in new predictors focusing more and more on the hard cases. This is the technique used by AdaBoost.

AdaBoosting

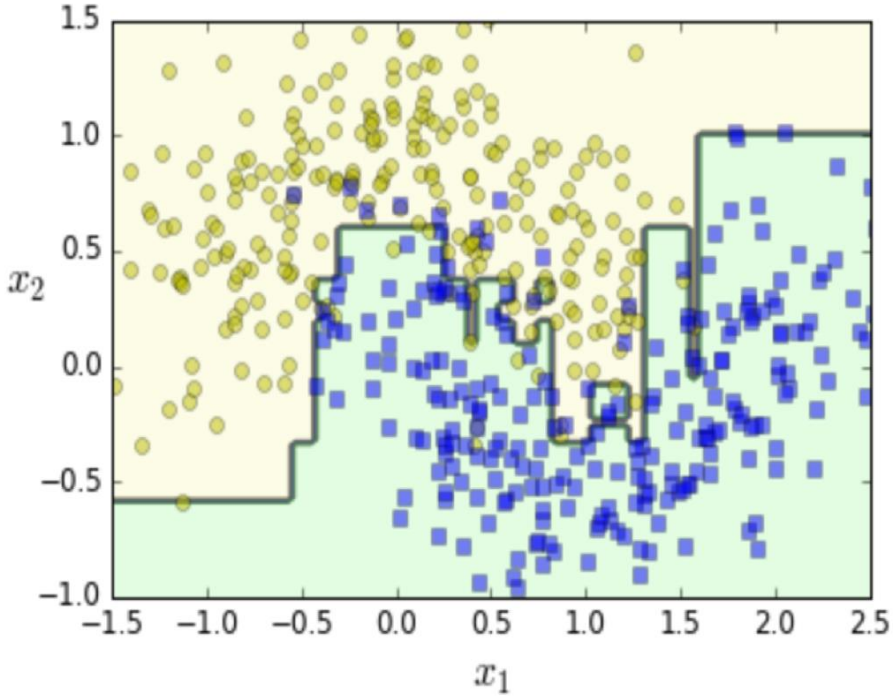
- For example, to build an AdaBoost classifier, a first base classifier (such as a DecisionTree) is trained and used to make predictions on the training set.
- The relative weight of misclassified training instances is then increased.

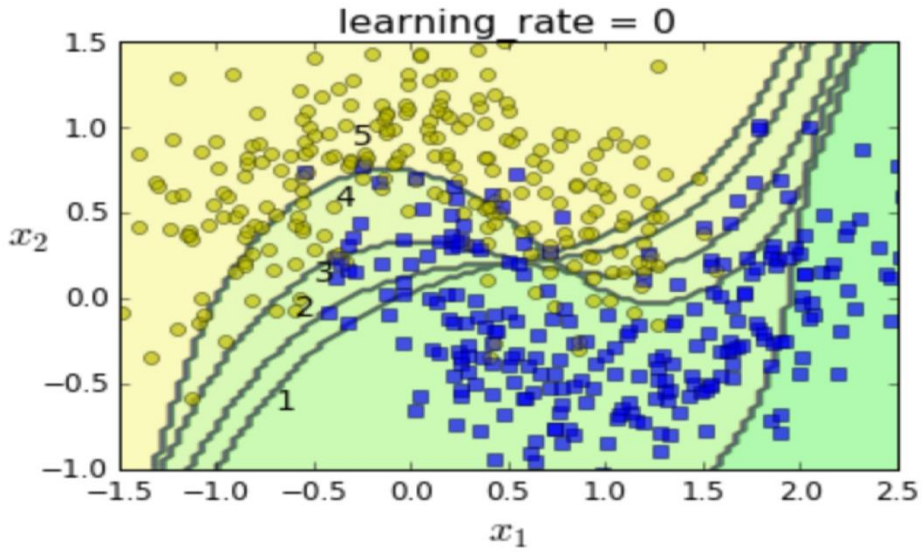


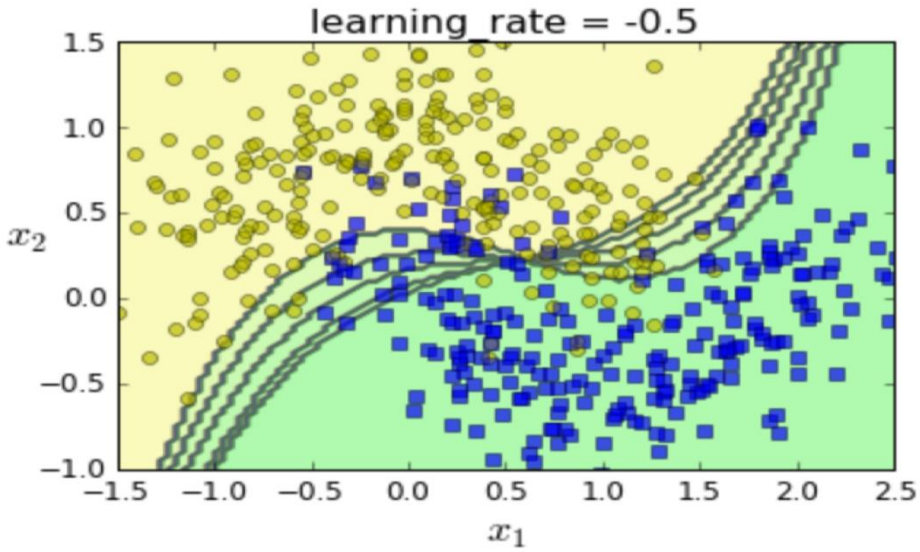
A second classifier is trained using the updated weights and again it makes predictions on the training set, weights are updated, and so on..

AdaBoosting

- The first base classifier (such as a DecisionTree) is trained and used to make predictions on the training set. The result shows:
- After five corrections, the decision boundaries of consecutive predictors:

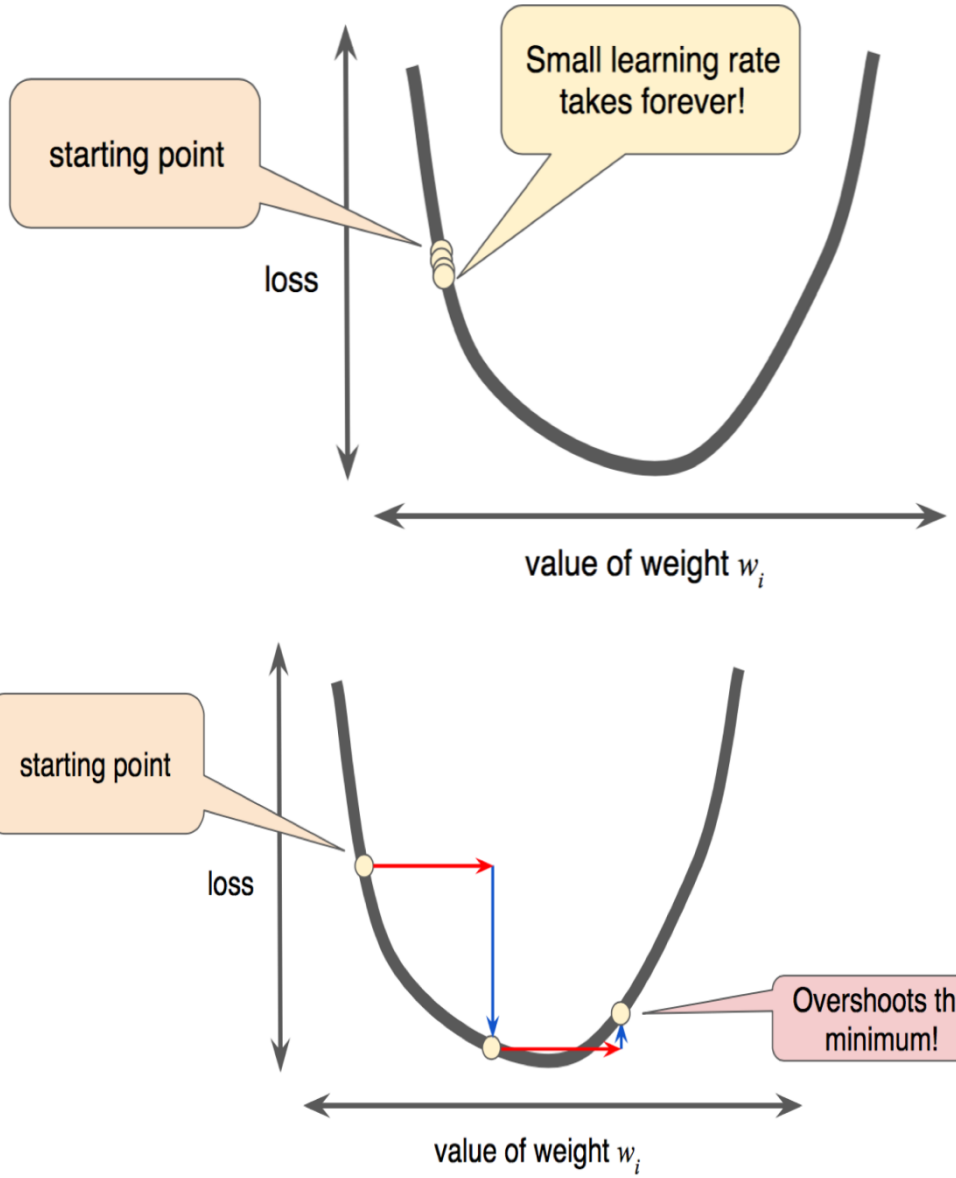






Learning Rate

- Gradient descent algorithms multiply the gradient by a scalar known as the learning rate (also sometimes called step size) to determine the next point.
- For example, if the gradient magnitude is 2.5 and the learning rate is 0.01, then the gradient descent algorithm will pick the next point 0.025 away from the previous point.

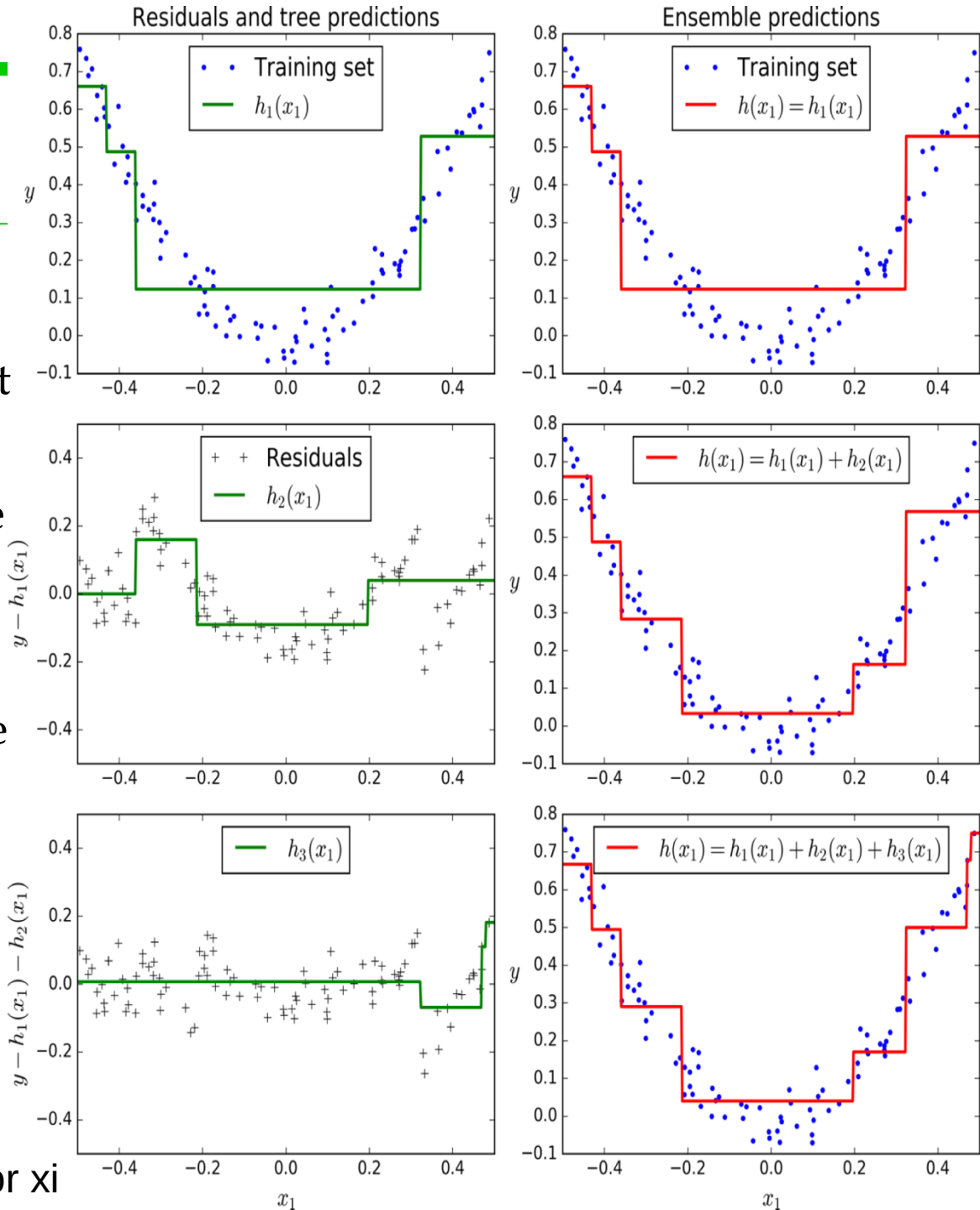


2). Gradient Boosting

- It is a very popular Boosting algorithm, just like AdaBoost.
- **However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit the new predictor to the residual errors made by the previous predictor.**
- A simple regression example, using DecisionTrees as the base predictors (of course it also works great with regression tasks).
- **This is called Gradient Tree Boosting, or Gradient Boosted Regression Trees (GBRT).**
- Let's learn its coding in the Lab class.

Gradient Boosting

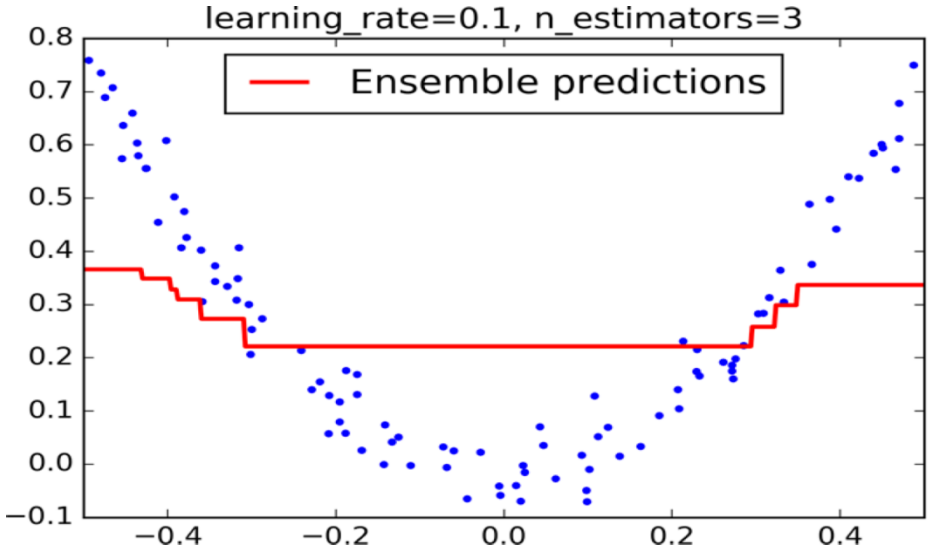
- the ensemble has one tree, so its predictions are exactly the same as the first tree's predictions
- a new tree is trained on the residual errors of the first tree. On the right you can see that the ensemble's predictions are equal to the sum of the predictions of the first two trees.
- Similarly, in the third row another tree is trained on the residual errors of the second tree.

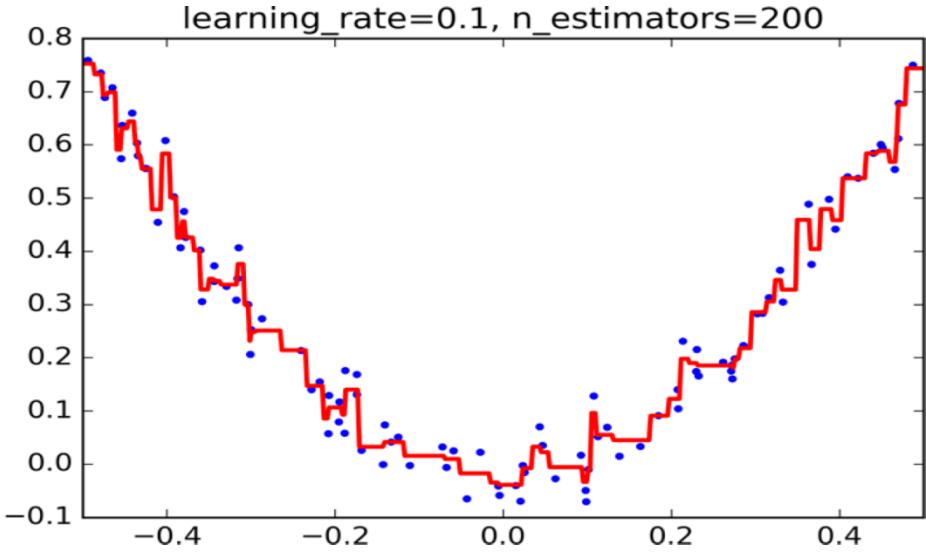


hj(xi) represents a prediction for xi

Gradient Boosting

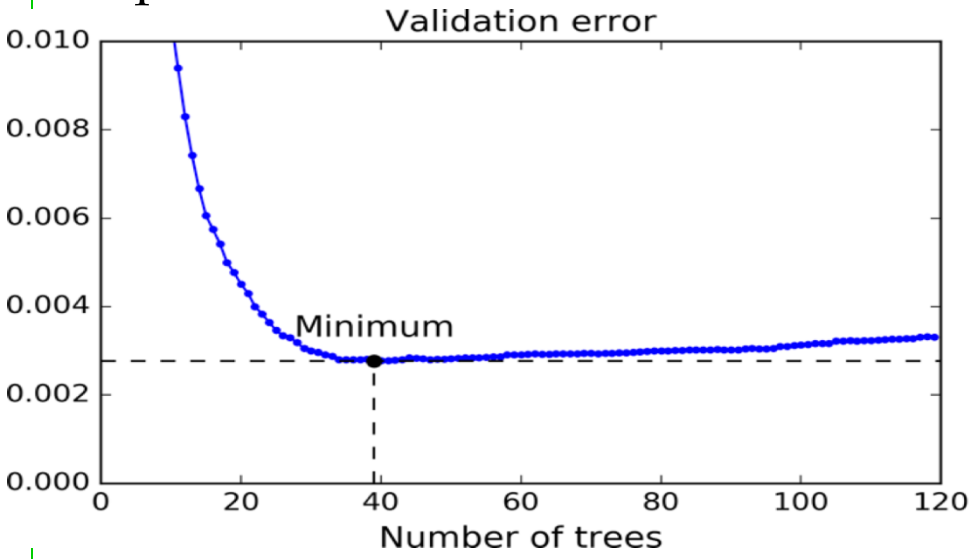
- The learning_rate scales the contribution of each tree. If you set it to a low value, such as 0.1, you will need more trees in the ensemble to fit the training set, but the predictions will generalize better.
- This is a regularization technique called shrinkage. The figure as below shows two GBRT ensembles trained with a low learning rate: the one on the left does not have enough trees to fit the training set, while the one on the right has too many trees and overfits the training set.

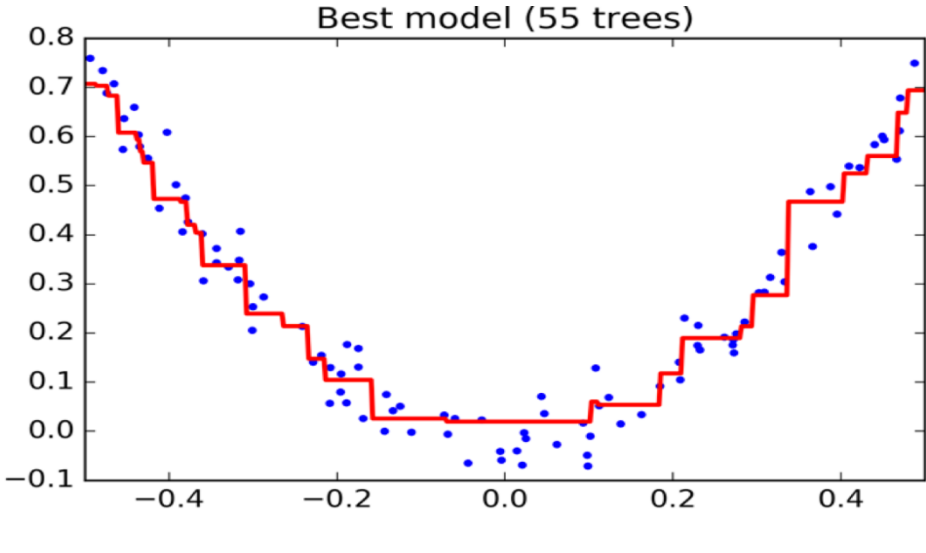




Gradient Boosting

- To find the optimal number of trees, you can use early stopping.
- It is to use the staged_predict() method: it returns an iterator over the predictions made by the ensemble at each stage of training (with one tree, two trees, etc.).
- The following example trains a GBRT ensemble with 120 trees, then measures the validation error at each stage of training to find the optimal number of trees, and finally trains another GBRT ensemble using the optimal number of trees:





Gradient Boosting

- It is also possible to implement early stopping by actually stopping training early (instead of training a large number of trees first and then looking back to find the optimal number).
- You can do so by setting warm_start=True, which makes Scikit-Learn keep existing trees when the fit() method is called, allowing incremental training.

Math reference:

- Chapter 8 of Introduction to Statistical Learning, By Gareth James, et al.

Let us have a look at coding samples:

- Ensemble_learning_and_random_forests.ipynb, located in …/Labs/Lab4/

Samples for Decision Trees and Random Forest with Scitk-Learn and Spark

- Samples:
  - 1) decision_trees.ipynb and
  - 2) Ensemble_learning_and_random_forests
  - 3) Testing_Three_Tree Methods

- **We will have a practice with them in the lab class.**

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

结束

2020年9月22日