Supervised Learning – Part 4

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Learning algorithms covered in this course

- **Supervised Learning** (Classification/Regression)
  - K-Nearest Neighbors
  - Linear Models (Logistic/Linear Regression)
  - **Decision Trees**
  - **Random Forests**
  - **Gradient Boosting Machines**
  - Support Vector Machines
  - Neural Networks

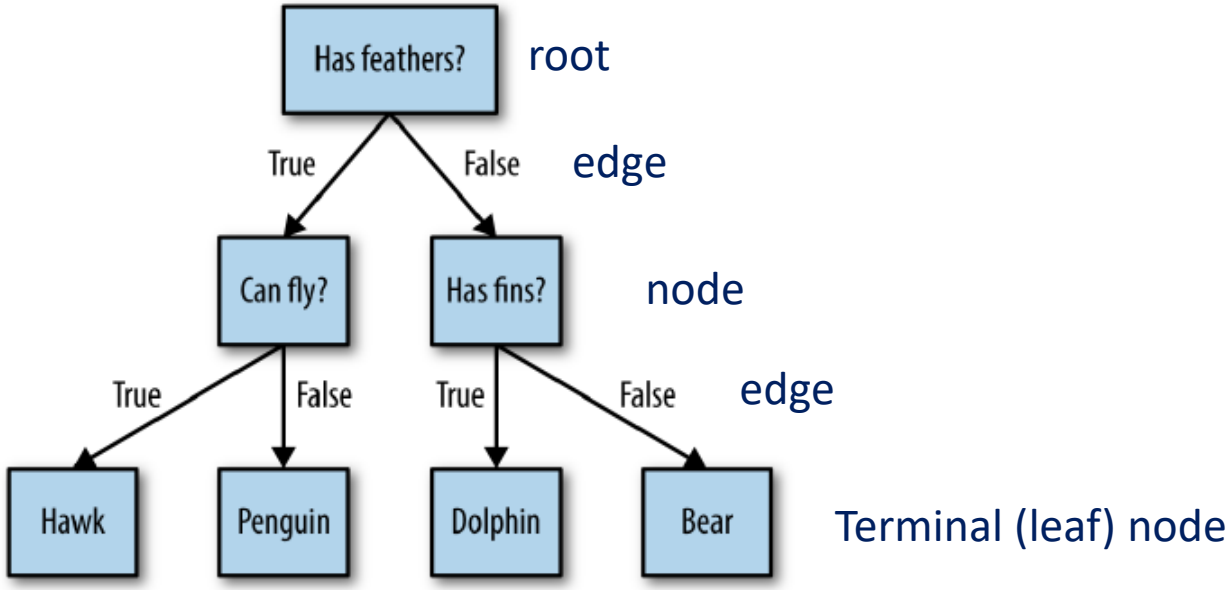
** Many algorithms have a classification and a regression variant, and we will describe both.*

** We will review the most popular machine learning algorithms, explain how they learn from data and how they make predictions, and examine the strengths and weaknesses of each algorithm.*

Decision Trees

Decision Trees

- **A decision tree is a hierarchy of if/else questions leading to a decision.**
  - Each node either represents a question or a terminal node (also called a leaf) that contains the answer.
  - The edges connect the answers to a question with the next question you would ask.



Decision Trees

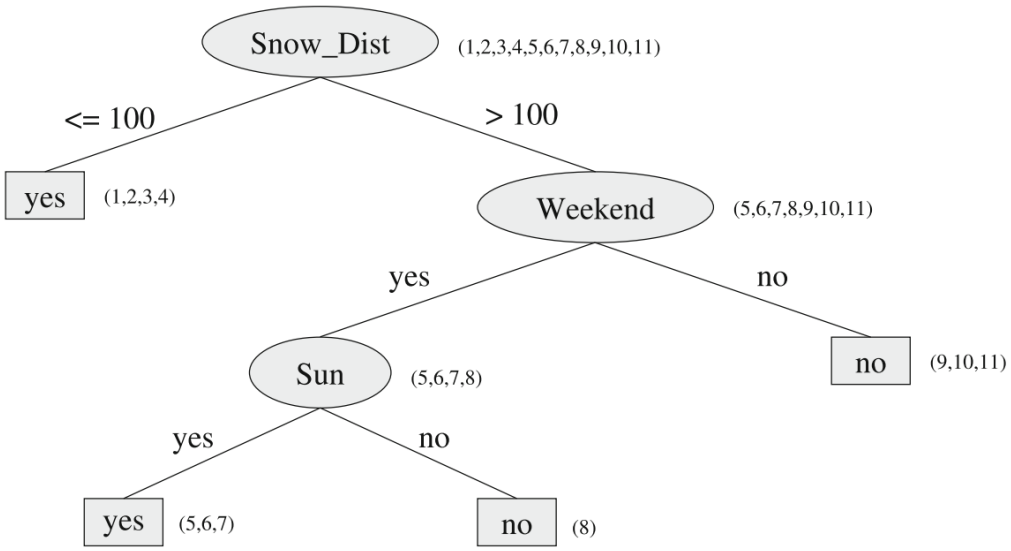
- **Example – Skiing Classification**
  - **Features:** "*Snow_Dist*", "*Weekend*", and "*Sun*".
  - **Target**: *"Skiing" = yes or no*

Dataset

Day	*Snow_Dist*	*Weekend*	*Sun*	*Skiing*
1	≤100	yes	yes	yes
2	≤100	yes	yes	yes
3	≤100	yes	no	yes
4	≤100	no	yes	yes
5	>100	yes	yes	yes
6	>100	yes	yes	yes
7	>100	yes	yes	no
8	>100	yes	no	no
9	>100	no	yes	no
10	>100	no	yes	no
11	>100	no	no	no

Ski (goal variable)	yes, no	Should I drive to the nearest ski resort with enough snow?
Sun (feature)	yes, no	Is there sunshine today?
Snow_Dist (feature)	≤ 100, > 100	Distance to the nearest ski resort with good snow conditions (over/under 100 km)
Weekend (feature)	yes, no	Is it the weekend today?

Decision Tree



Decision Trees

- **Instead of building the tree by hand, we can learn them from data**
  → the output is a set of if/else rules represented by a tree diagram
  - Each inner node represents a feature.
  - Each edge stands for the condition of a feature value.
  - At each leaf node, a prediction is given.

- **The extracted knowledge can be easily understood, interpreted, and controlled by humans in the form of a readable decision tree**

- **Decision trees are widely used for classification and regression tasks**

Decision Trees

- **There are finitely many different decision trees.**
  - **Optimal algorithm** for the construction of a tree is to simply generate all possible trees and choose the best one.
  - The obvious disadvantage of this algorithm is its unacceptably high computation time, as soon as the number of features becomes somewhat larger.

- **Decision Tree Learning**
  - We use **heuristic algorithms** with greedy strategy.
  - Because greedy strategy is used for construction of the tree, the trees are in general sub-optimal.

Decision Trees

- Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features and y_i is the corresponding target label.

- **General Procedure –** Repeatedly split a node into two parts so as to minimize the impurity of outcome within the new parts.
  - The training dataset D constitutes the root node
  - Repeat the following process
    - Try all possible splits in all nodes and features to find the **"best split"**
    - Split the node

Decision Trees

- **How to determine the best split (for classification)**
  - Nodes with low degree of impurity are preferred

C0: 5 C1: 5	C0: 9 C1: 1
High degree of impurity	Low degree of impurity

  - **Measures of node impurity**

    * $p(j|t)$ is the fraction of class j at node t

$$\text{GINI}(t) = 1 - \sum_j [p(j|t)]^2$$

$$\text{Entropy}(t) = -\sum_j p(j|t) \log p(j|t)$$

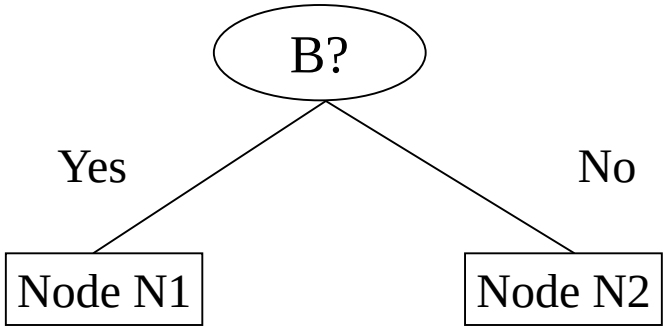
Decision Trees

- **How to determine the best split (for classification)**
  1. **Compute impurity measure (P) before splitting**
  2. **Compute impurity measure (M) after splitting**
     - Compute impurity measure of each child node
     - M is the weighted impurity of children
  3. **Choose the feature that produces the highest gain**
     or equivalently, lowest impurity measure after splitting (M)
     - **Gain = P - M**

Decision Trees

- **How to determine the best split (for classification)**

Gini(Parent)
$= 1 - (7/12)^2 - (5/12)^2$
$= 0.486$



Gini(N1)
$= 1 - (5/6)^2 - (1/6)^2$
$= 0.278$

Gini(N2)
$= 1 - (2/6)^2 - (4/6)^2$
$= 0.444$

Weighted Gini of N1 and N2
$= 6/12 * 0.278 + 6/12 * 0.444$
$= 0.361$

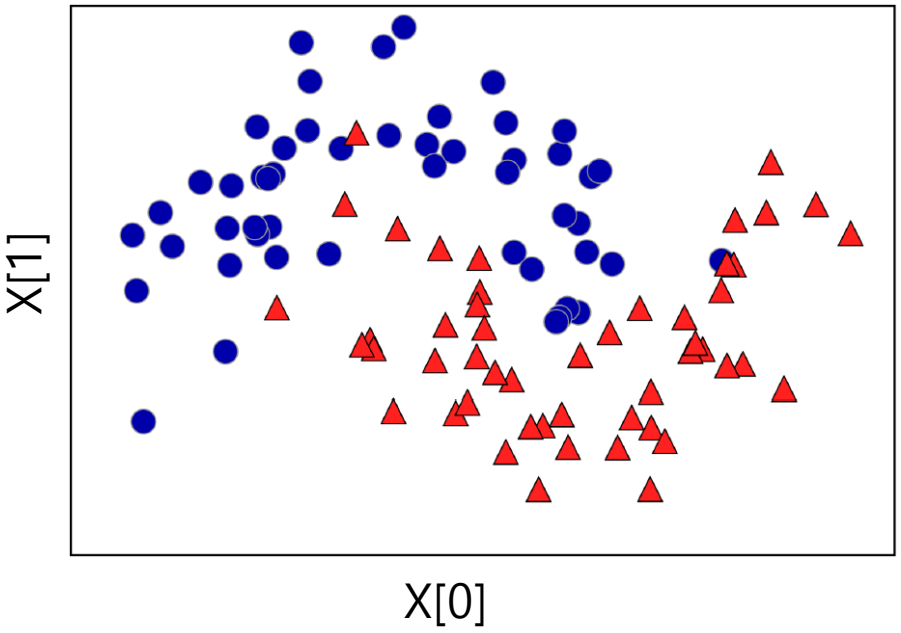
	Parent
C1	7
C2	5
Gini = 0.486	

	N1	N2
C1	5	2
C2	1	4
Gini=0.361		

Gain = 0.486 – 0.361 = 0.125

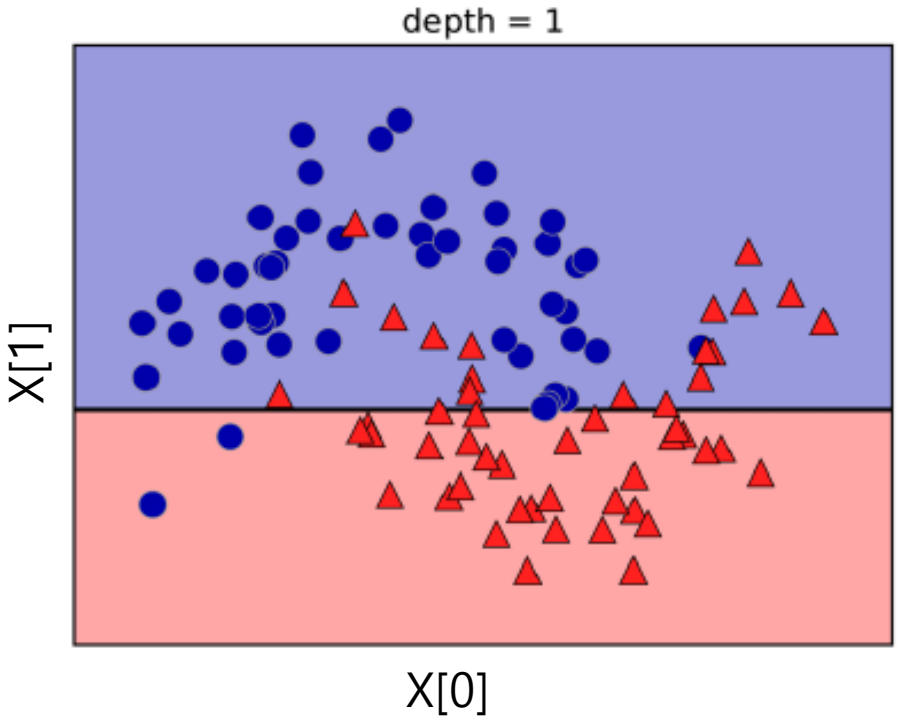
Decision Trees

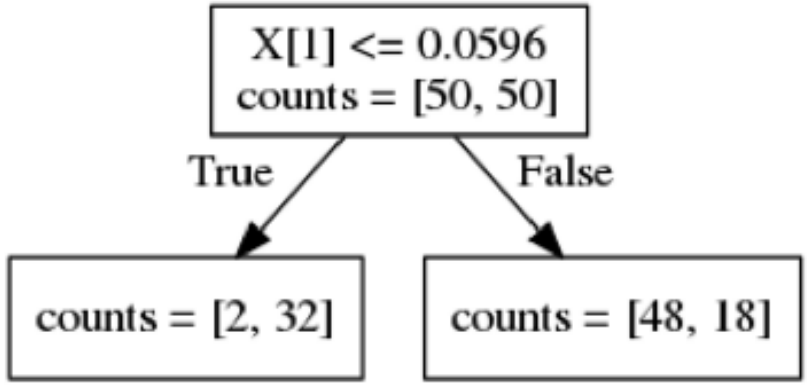
- **Example of building a decision tree (for classification)**
  - The *two-moons* dataset consists of two half-moon shapes, with each class consisting of 50 data points.



Decision Trees

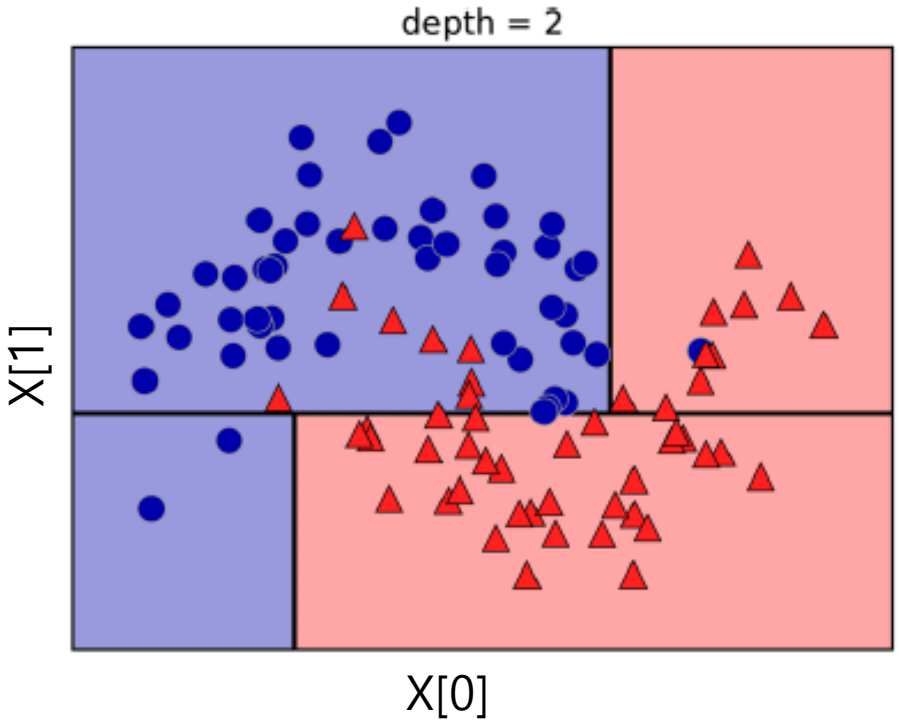
- **Example of building a decision tree (for classification)**
  - The algorithm searches over all possible tests and finds the one that is most informative about the target.

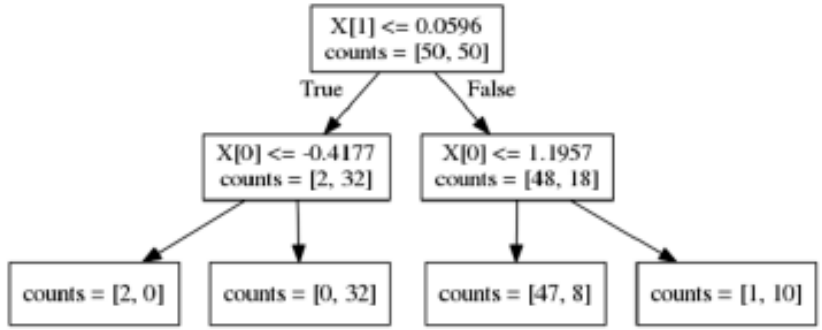




Decision Trees

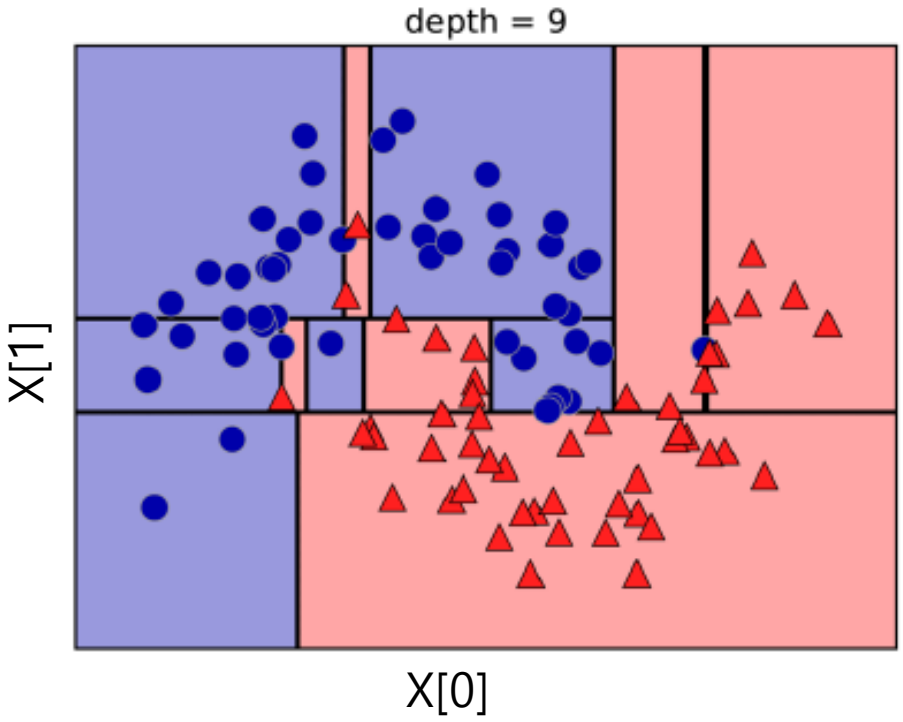
- **Example of building a decision tree (for classification)**
  - We can build a more accurate model by repeating the process of looking for the most informative next split for the left and the right region.

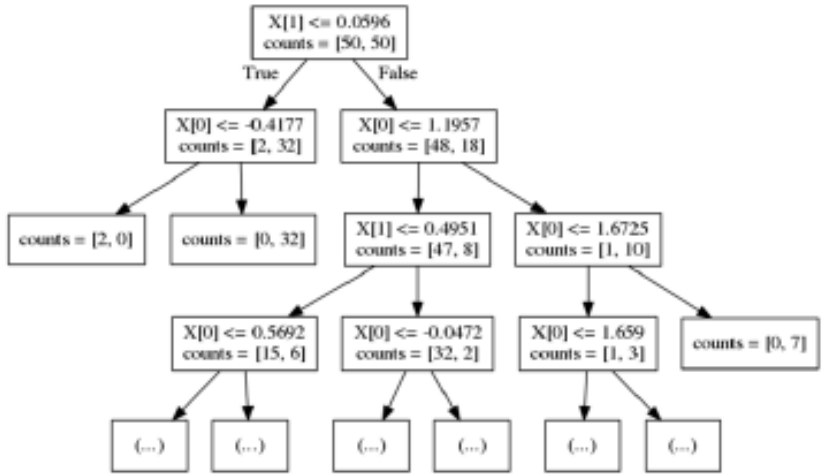




Decision Trees

- **Example of building a decision tree (for classification)**
  - The recursive partitioning of the data is repeated until each region in the partition (each leaf in the tree) become homogeneous.





Decision Trees

- **To make a prediction for a new data point, we traverse the tree to find the leaf the data point falls into.**

- **For Classification,**
  - The output for the data point is the majority class of the training points in this leaf.

- **For Regression**
  - The output for the data point is the mean target of the training points in this leaf.

Decision Trees

- **Building a tree typically leads to a model that is very complex and highly overfit to the training data.**
  - Continuing until all leaves are pure means that a tree is 100% accurate on the training set, but fails to generalize on new data

- **Two common strategies to prevent overfitting**
  - **Pre-pruning:** stopping the creation of the tree early (*e.g.*, below)
    - *max_depth*: limiting the maximum depth of the tree
    - *min_sample_leaf*: requiring a minimum number of points in a node to keep splitting it
    - *max_leaf_nodes*: limiting the maximum number of leaves
  - **Post-pruning:** building the tree but then removing or collapsing nodes that contain little information

scikit-learn Practice: *DecisionTreeClassifier*

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
class_weight=None, ccp_alpha=0.0, monotonic_cst=None)
```

A decision tree classifier.

**** It supports pre-pruning but not post-pruning.***

criterion	*{"gini", "entropy", "log_loss"}, default="gini"* The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_loss" and "entropy" both for the Shannon information gain, see Mathematical formulation.
max_depth	*int, default=None* The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
min_samples_leaf	*int or float, default=1* The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression. If int, then consider min_samples_leaf as the minimum number. If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
max_leaf_nodes	*int, default=None* Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

scikit-learn Practice: *DecisionTreeClassifier*

- **Example (*breast_cancer* dataset)**

[1]:
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
```

[2]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=0)
```

[3]:
```
from sklearn.metrics import accuracy_score
y_train_hat = clf.predict(X_train)
print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))
y_test_hat = clf.predict(X_test)
print('test accuracy: %.5f'%accuracy_score(y_test, y_test_hat))
```

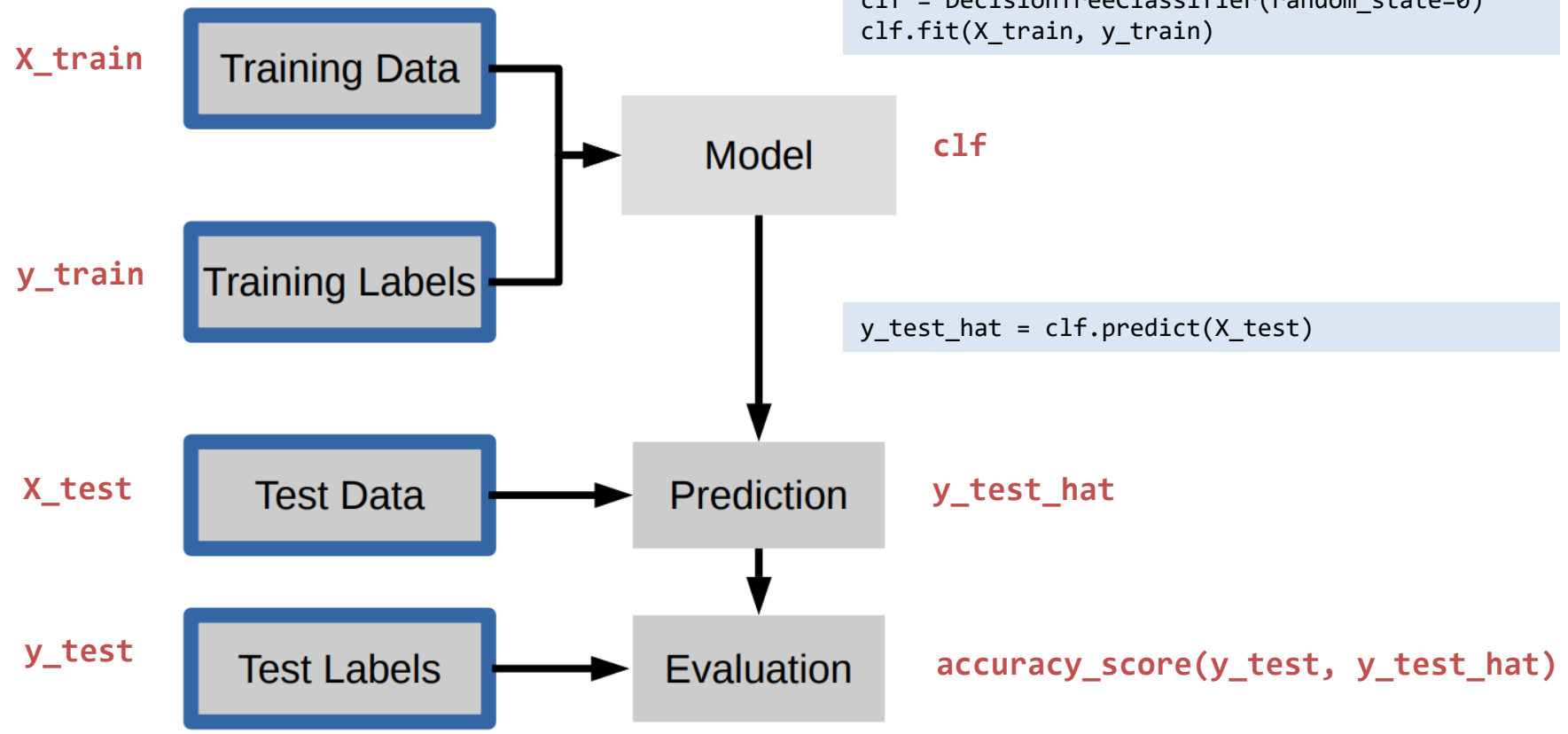
```
train accuracy: 1.00000
test accuracy: 0.93706
```

scikit-learn Practice: *DecisionTreeClassifier*

- **Example (*breast_cancer* dataset)**

```
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, stratify=cancer.target,
random_state=42)
```

```
clf = DecisionTreeClassifier(random_state=0)
clf.fit(X_train, y_train)
```



scikit-learn Practice: *DecisionTreeClassifier*

- **Example (*breast_cancer* dataset): varying the hyperparameter *min_samples_leaf***

[1]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
```

[2]:
```
training_accuracy = []
test_accuracy = []

m_settings = [1, 2, 5, 7, 10, 20]
for m in m_settings:
    # build the model
    clf = DecisionTreeClassifier(min_samples_leaf=m, random_state=0)
    clf.fit(X_train, y_train)

    # accuracy on the training set
    y_train_hat = clf.predict(X_train)
    training_accuracy.append(accuracy_score(y_train, y_train_hat))

    # accuracy on the test set (generalization)
    y_test_hat = clf.predict(X_test)
    test_accuracy.append(accuracy_score(y_test, y_test_hat))
```

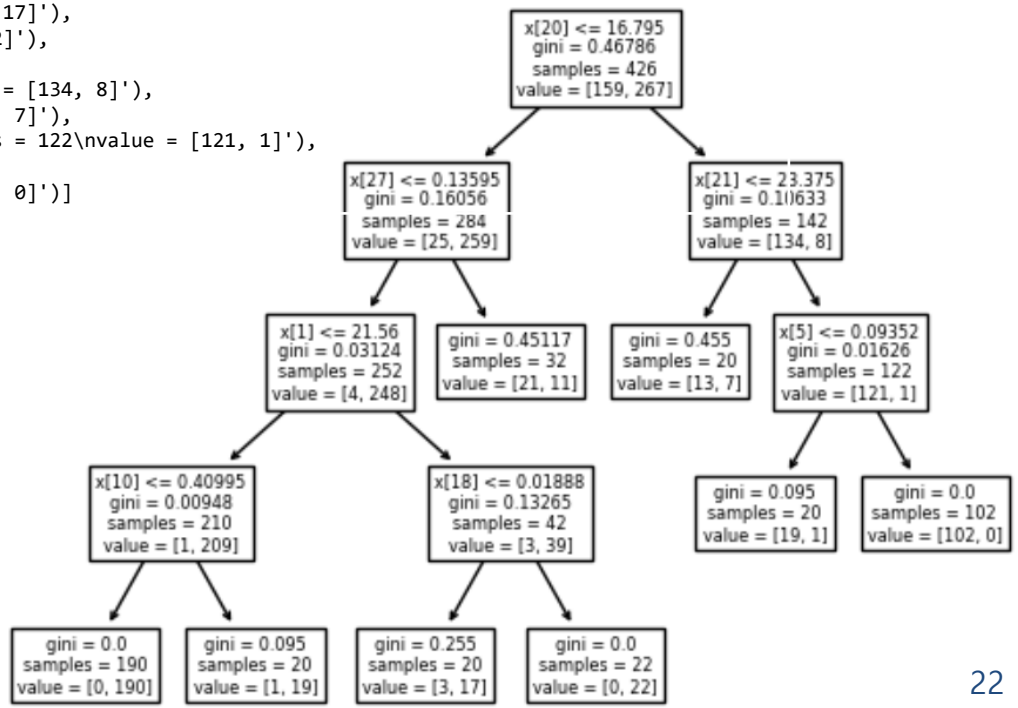
	min_samples_leaf	training accuracy	test accuracy
0	1	1.00000	0.93706
1	2	0.99061	0.93706
2	5	0.97653	0.93706
3	7	0.96244	0.95105
4	10	0.96244	0.95105
5	20	0.94601	0.91608

Visualizing and Analyzing Decision Trees

- **The visualization of the tree provides a great in-depth view of how the algorithm makes predictions**

[3]:
```
from sklearn import tree
tree.plot_tree(clf, precision=5)
```

```
[Text(0.5833333333333334, 0.9, 'x[20] <= 16.795\ngini = 0.46786\nsamples = 426\nvalue = [159, 267]'),
Text(0.4166666666666667, 0.7, 'x[27] <= 0.13595\ngini = 0.16056\nsamples = 284\nvalue = [25, 259]'),
Text(0.3333333333333333, 0.5, 'x[1] <= 21.56\ngini = 0.03124\nsamples = 252\nvalue = [4, 248]'),
Text(0.16666666666666666, 0.3, 'x[10] <= 0.40995\ngini = 0.00948\nsamples = 210\nvalue = [1, 209]'),
Text(0.08333333333333333, 0.1, 'gini = 0.0\nsamples = 190\nvalue = [0, 190]'),
Text(0.25, 0.1, 'gini = 0.095\nsamples = 20\nvalue = [1, 19]'),
Text(0.5, 0.3, 'x[18] <= 0.01888\ngini = 0.13265\nsamples = 42\nvalue = [3, 39]'),
Text(0.4166666666666667, 0.1, 'gini = 0.255\nsamples = 20\nvalue = [3, 17]'),
Text(0.5833333333333334, 0.1, 'gini = 0.0\nsamples = 22\nvalue = [0, 22]'),
Text(0.5, 0.5, 'gini = 0.45117\nsamples = 32\nvalue = [21, 11]'),
Text(0.75, 0.7, 'x[21] <= 23.375\ngini = 0.10633\nsamples = 142\nvalue = [134, 8]'),
Text(0.6666666666666666, 0.5, 'gini = 0.455\nsamples = 20\nvalue = [13, 7]'),
Text(0.8333333333333334, 0.5, 'x[5] <= 0.09352\ngini = 0.01626\nsamples = 122\nvalue = [121, 1]'),
Text(0.75, 0.3, 'gini = 0.095\nsamples = 20\nvalue = [19, 1]'),
Text(0.9166666666666666, 0.3, 'gini = 0.0\nsamples = 102\nvalue = [102, 0]')]
```



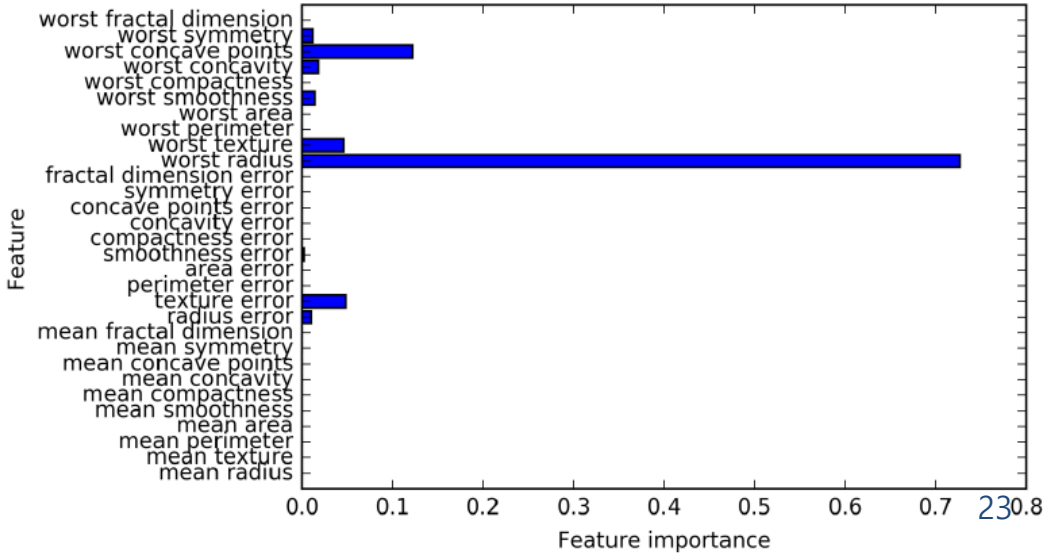
Feature Importance in Decision Trees

- ***feature importance*** **summarizes the workings of a tree by rating how important each feature is for the decision the tree makes.**
  - The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature.
  - It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target.”

Attributes	
feature_importances_	*ndarray of shape (n_features,)* Return the feature importances.

```
[4]: clf.feature_importances_
```

```
array([0.00000e+00, 1.86437e-03, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       5.01021e-04, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       5.42188e-04, 0.00000e+00, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 0.00000e+00, 2.82508e-03, 0.00000e+00,
       8.30651e-01, 2.40602e-02, 0.00000e+00, 0.00000e+00, 0.00000e+00,
       0.00000e+00, 0.00000e+00, 1.39556e-01, 0.00000e+00, 0.00000e+00])
```



scikit-learn Practice: *DecisionTreeRegressor*

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

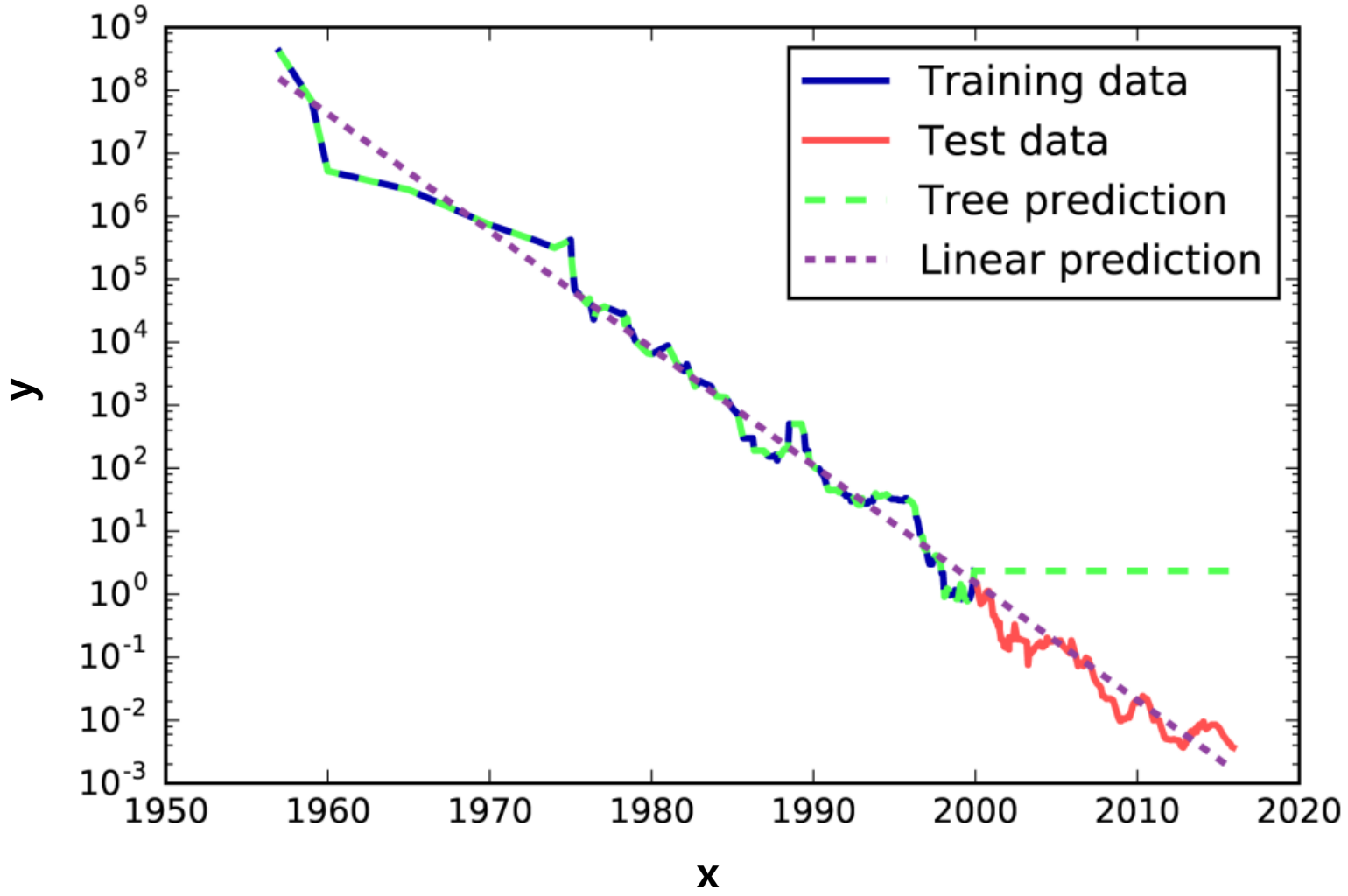
```
class sklearn.tree.DecisionTreeRegressor(*, criterion='squared_error', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0,
ccp_alpha=0.0, monotonic_cst=None)
```

A decision tree regressor.

**** It supports pre-pruning but not post-pruning.***

criterion	*{"squared_error", "friedman_mse", "absolute_error", "poisson"}, default="squared_error"* The function to measure the quality of a split. Supported criteria are "squared_error" for the mean squared error, which is equal to variance reduction as feature selection criterion and minimizes the L2 loss using the mean of each terminal node, "friedman_mse", which uses mean squared error with Friedman's improvement score for potential splits, "absolute_error" for the mean absolute error, which minimizes the L1 loss using the median of each terminal node, and "poisson" which uses reduction in the half mean Poisson deviance to find splits.

scikit-learn Practice: *DecisionTreeRegressor*



Discussion

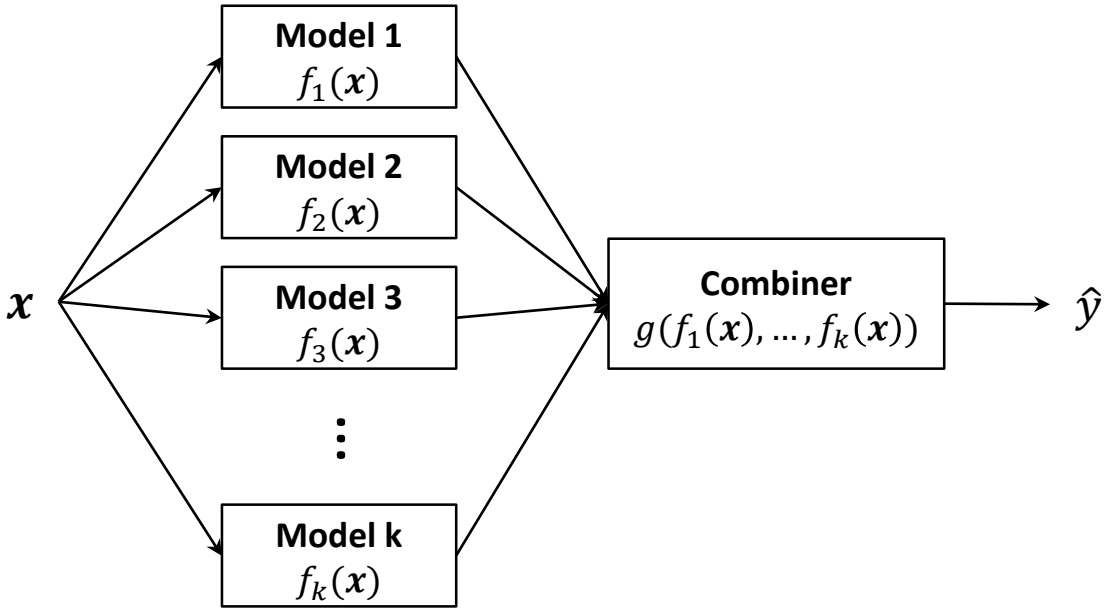
- **The main hyperparameters of decision trees**
  - *criterion*: the function to measure the quality of a split (impurity measure)
  - Picking one of the pre-pruning strategies—setting either *max_depth, max_leaf_nodes*, or *min_samples_leaf*—is sufficient to prevent overfitting.

  * Typically chosen to achieve the highest performance on validation data
- **Strengths**
  - Decision trees tend to work well when you have a mix of continuous and categorical features.
  - The algorithm is completely invariant to the scales of features (no data scaling is needed)
  - Feature selection & reduction is automatic.
  - It is robust to outliers.
  - The resulting model can easily be visualized and understood.
- **Weaknesses**
  - Even with the use of pre-pruning, they tend to overfit and provide poor generalization performance.
    → the ensemble methods are usually used in place of a single decision tree.
  - It does not take into account interactions between features.
  - Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the optimal tree.

Random Forests

Ensembles of Decision Trees

- Decision trees tend to overfit the training data
  → Ensembles are one way to address this problem.
- **Ensemble methods** combine multiple machine learning models to create more powerful models.



Random Forests

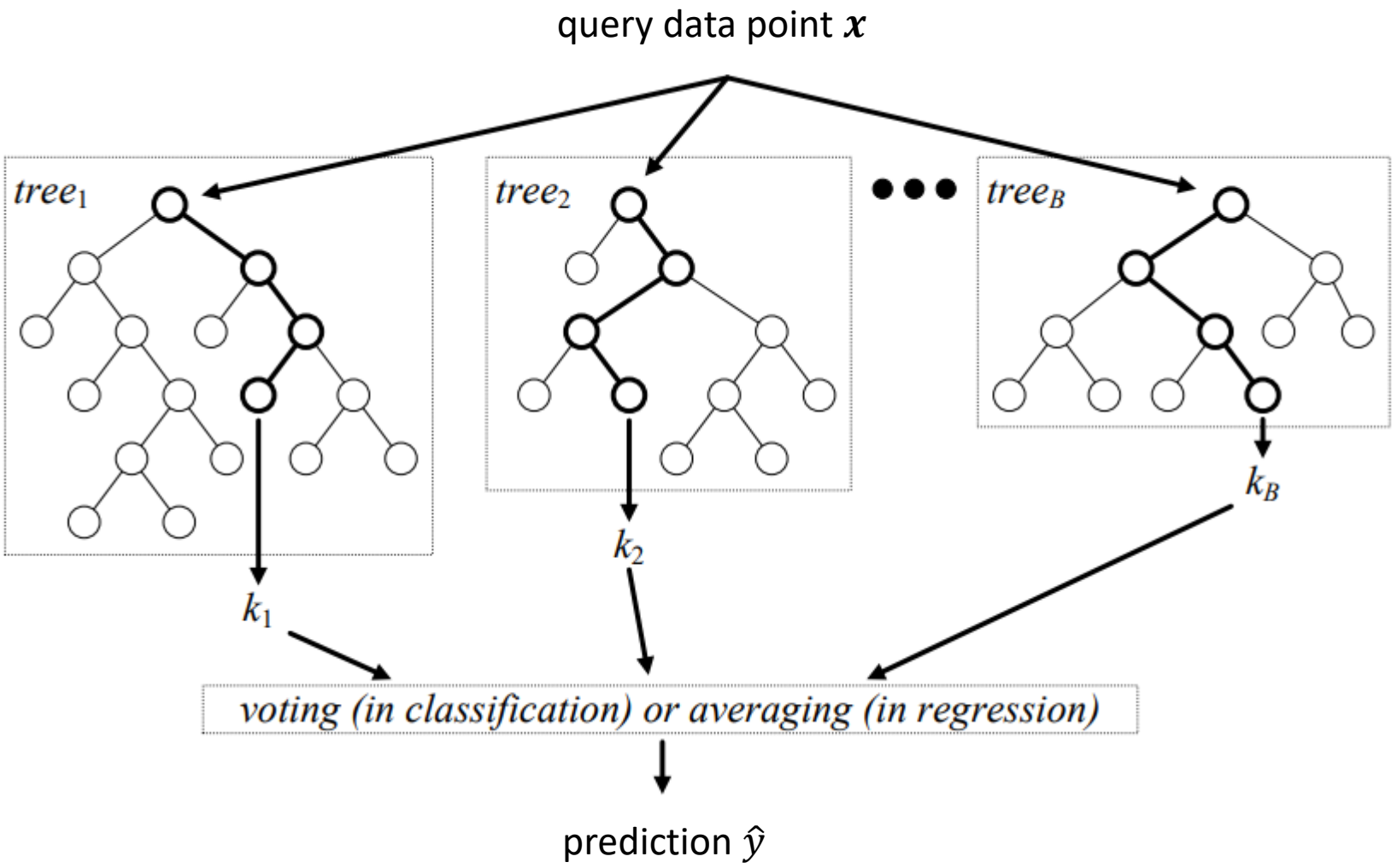
- **A random forest is an ensemble of decision trees, where each tree is slightly different from the others.**
  - Each tree might do a relatively good job of predicting, but will likely overfit on part of the data in different ways.
  - If we build many trees, we can reduce the amount of overfitting by averaging their results while retaining the predictive power of the trees.
    - Each tree should do an acceptable job of predicting the target.
    - Each tree should be different from the other trees.
  - Random forests get their name from injecting randomness into the tree building to ensure each tree is different.

Random Forests

- **Two ways in which the trees in a random forest are randomized**
  - **by selecting the data points used to build a tree (bootstrap sampling)**
    - *bootstrap*: It leads to each tree in the random forest being built on a slightly different dataset.
      - We repeatedly draw data points randomly with replacement (meaning the same data point can be picked multiple times)
      - From a list ['a', 'b', 'c', 'd'], possible examples of bootstrap samples are ['b', 'd', 'd', 'c'] and ['d', 'a', 'd', 'a'].
    - *max_samples*: The size of each bootstrap sample.
  - **by selecting the features in each split test.**
    - *max_features*: In each node, the algorithm randomly selects a subset of the features, and it looks for the best possible test involving one of these features
      → each node in a tree can make a decision using a different subset of the features.
      - A high *max_features* means that the trees in the random forest will be quite similar, and they will be able to fit the data easily, using the most distinctive features.
      - A low *max_features* means that the trees in the random forest will be quite different, and that each tree might need to be very deep in order to fit the data well.

** Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors recommend p/3 (rounded down) as the default. In practice the best value will depend on the problem.*

Random Forests



Random Forests

- **To make a prediction for a new data point, we first make a prediction for every tree in the forest.**

- **For Classification,**
  - Each tree makes a “soft” prediction, providing a probability for each possible output label.
  - The probabilities predicted by all the trees are averaged.
  - The output for the data point is the class with the highest average probability.

- **For Regression**
  - The output for the data point is the mean prediction of the trees in the forest.

scikit-learn Practice: *RandomForestClassifier*

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True,
oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)
```

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Trees in the forest use the best split strategy, i.e. equivalent to passing splitter="best" to the underlying DecisionTreeClassifier. The sub-sample size is controlled with the max_samples hyperparameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

n_estimators	*int, default=100* The number of trees in the forest.
max_features	*{"sqrt", "log2", None}, int or float, default="sqrt"* The number of features to consider when looking for the best split: • If int, then consider max_features features at each split. • If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split. • If "sqrt", then max_features=sqrt(n_features). • If "log2", then max_features=log2(n_features). • If None, then max_features=n_features.

scikit-learn Practice: *RandomForestClassifier*

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

bootstrap	*bool, default=True* Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.
max_samples	*int or float, default=None* If bootstrap is True, the number of samples to draw from X to train each base estimator. • If None (default), then draw X.shape[0] samples. • If int, then draw max_samples samples. • If float, then draw max(round(n_samples * max_samples), 1) samples. Thus, max_samples should be in the interval (0.0, 1.0].

scikit-learn Practice: *RandomForestClassifier*

- **Example (*breast_cancer* dataset): varying the hyperparameter *n_estimators***

[1]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, stratify=cancer.target, random_state=42)
```

[2]:
```
training_accuracy = []
test_accuracy = []

n_settings = [1, 2, 5, 10, 20, 50, 100]
for n in n_settings:
    # build the model
    clf = RandomForestClassifier(n_estimators=n)
    clf.fit(X_train, y_train)

    # accuracy on the training set
    y_train_hat = clf.predict(X_train)
    training_accuracy.append(accuracy_score(y_train, y_train_hat))

    # accuracy on the test set (generalization)
    y_test_hat = clf.predict(X_test)
    test_accuracy.append(accuracy_score(y_test, y_test_hat))
```

	n_estimators	training accuracy	test accuracy
0	1	0.98357	0.93706
1	2	0.97887	0.91608
2	5	0.98826	0.93706
3	10	0.99765	0.95105
4	20	1.00000	0.95105
5	50	1.00000	0.95804
6	100	1.00000	0.95804

How Many Trees in a Random Forest?

International Workshop on Machine Learning and Data Mining in Pattern Recognition
MLDM 2012: Machine Learning and Data Mining in Pattern Recognition pp 154-168 | Cite as

How Many Trees in a Random Forest?

Authors | Authors and affiliations

Thais Mayumi Oshiro, Pedro Santoro Perez, José Augusto Baranauskas

Conference paper

80 Citations | 2 Readers | 5.5k Downloads

Part of the Lecture Notes in Computer Science book series (LNCS, volume 7376)

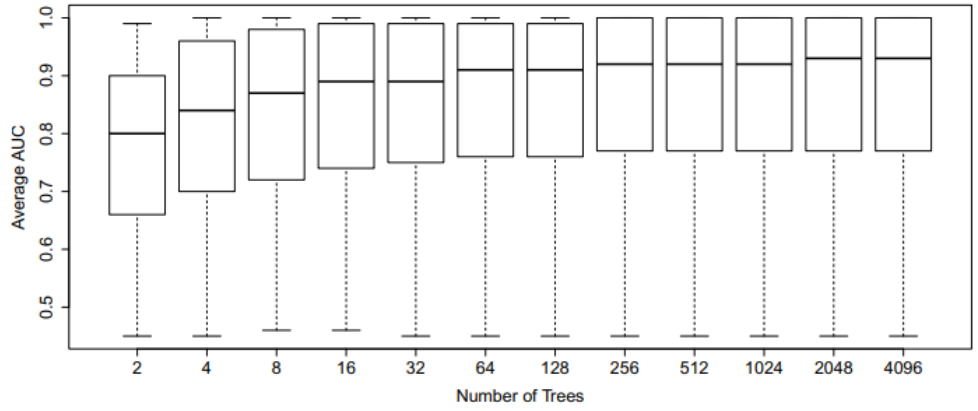


Fig. 1. AUC in all datasets

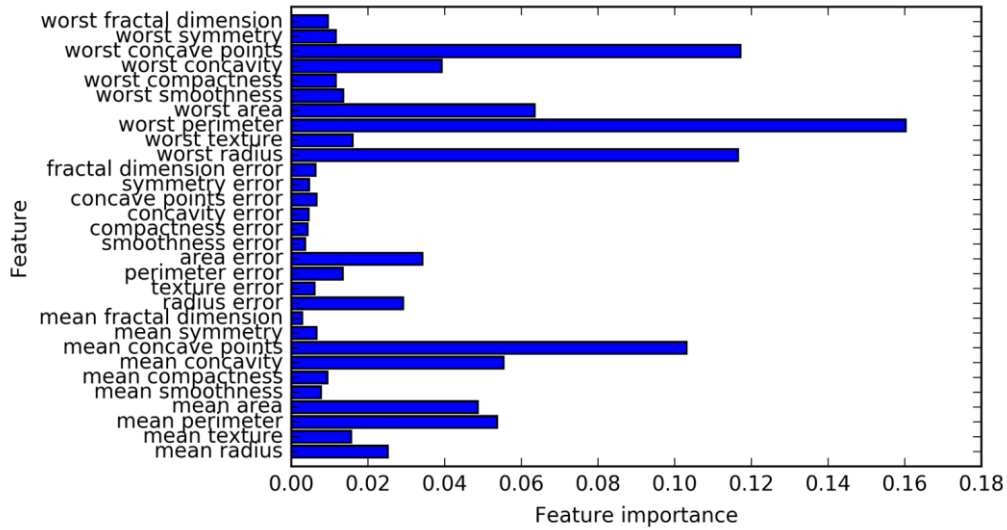
… … … The analysis of 29 datasets shows that from 128 trees there is no more significant difference between the forests using 256, 512, 1024, 2048 and 4096 trees. The mean and the median AUC values do not present major changes from 64 trees. Therefore, ***it is possible to suggest, based on the experiments, a range between 64 and 128 trees in a forest****. With these numbers of trees it is possible to obtain a good balance between AUC, processing time, and memory usage. … … …*

Feature Importance in Random Forest

- **Similarly to the decision tree, the random forest provides feature importances**
  - Computed by aggregating the feature importances over the trees in the forest.
  - Typically, the feature importances provided by the random forest are more reliable than the ones provided by a single tree.

```
[3]: clf.feature_importances_
```

```
array([0.04033, 0.0113 , 0.02692, 0.05785, 0.01001, 0.0192 , 0.04596,
       0.08665, 0.00524, 0.00268, 0.00956, 0.00659, 0.00839, 0.02436,
       0.00687, 0.0026 , 0.00927, 0.00568, 0.00429, 0.00481, 0.05634,
       0.01483, 0.09196, 0.20529, 0.01559, 0.00908, 0.02657, 0.17535,
       0.01021, 0.00624])
```



scikit-learn Practice: *RandomForestRegressor*

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

```
class sklearn.ensemble.RandomForestRegressor(n_estimators=100, *,
criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None,
verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None, monotonic_cst=None)
```

A random forest regressor.

A random forest is a meta estimator that fits a number of decision tree regressors on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Trees in the forest use the best split strategy, i.e. equivalent to passing splitter="best" to the underlying DecisionTreeRegressor. The sub-sample size is controlled with the max_samples hyperparameter if bootstrap=True (default), otherwise the whole dataset is used to build each tree.

max_features	*{"sqrt", "log2", None}, int or float, default=1.0* The number of features to consider when looking for the best split: • If int, then consider max_features features at each split. • If float, then max_features is a fraction and max(1, int(max_features * n_features_in_)) features are considered at each split. • If "sqrt", then max_features=sqrt(n_features). • If "log2", then max_features=log2(n_features). • If None, then max_features=n_features.

Discussion

- **The main hyperparameters of random forests**
  - *n_estimators*: larger is always better
  - *bootstrap, max_samples*: whether bootstrap samples are used
  - *max_features*: how random each tree is, the default values in scikit-learn are sqrt(n_features) for classification and n_features for regression
  - Plus, the main hyperparameters of decision trees

  * Typically chosen to achieve the highest performance on validation data
- **Strengths**
  - Random forests are very powerful, often work well without heavy tuning of the hyperparameters
  - Random forests share most of the benefits of decision trees
- **Weaknesses**
  - It is basically impossible to interpret all the trees in detail.
  - Trees in random forests tend to be deeper than decision trees.
  - Random forests don't tend to perform well on very high dimensional, sparse data.
  - Building random forests on large datasets might be time consuming
    → but, it can be parallelized across multiple CPU (use *n_jobs*)

Discussion

Journal of Machine Learning Research 15 (2014) 3133-3181 Submitted 11/13; Revised 4/14; Published 10/14

Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

Abstract

We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

Discussion

Journal of Machine Learning Research 17 (2016) 1-5 Submitted 7/15; Revised 3/16; Published 7/16

Are Random Forests Truly the Best Classifiers?

Michael Wainberg M.WAINBERG@UTORONTO.CA
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON M5S 3G4, Canada;
Deep Genomics, Toronto, ON M5G 1L7, Canada

Babak Alipanahi BABAK@PSI.TORONTO.EDU
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON M5S 3G4, Canada

Brendan J. Frey FREY@PSI.TORONTO.EDU
Department of Electrical and Computer Engineering
University of Toronto, Toronto, ON M5S 3G4, Canada;
Deep Genomics, Toronto, ON M5G 1L7, Canada

Editor: Nando de Freitas

Abstract

The JMLR study *Do we need hundreds of classifiers to solve real world classification problems?* benchmarks 179 classifiers in 17 families on 121 data sets from the UCI repository and claims that "the random forest is clearly the best family of classifier". In this response, we show that the study's results are biased by the lack of a held-out test set and the exclusion of trials with errors. Further, the study's own statistical tests indicate that random forests do not have significantly higher percent accuracy than support vector machines and neural networks, calling into question the conclusion that random forests are the best classifiers.

Gradient Boosting Machines

Gradient Boosting Machines

- **A gradient boosting machine is another ensemble method that combines multiple decision trees to create a more powerful model.**
  - (In contrast to the random forest) gradient boosting works by **building trees in a serial manner**, where **each tree tries to correct the mistakes of the previous one**.
    - Each tree can only provide good predictions on part of the data, and so more and more trees are added to iteratively improve performance.
    - Trees are sequentially added to an ensemble, with each one correcting its predecessor by fitting to the residual errors made by the predecessor.
  - (In contrast to the random forest) By default, there is no randomization in gradient boosting machines; instead, strong pre-pruning is used.
  - (In contrast to the random forest) Gradient boosting machines **often use very shallow trees**, of depth one to five, which makes the model smaller in terms of memory and makes predictions faster.
  - Despite the “regression” in the name, it can be used for regression and classification.

Gradient Boosting Machines

- **Training Procedure**
  - Given a (training) dataset $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features and y_i is the corresponding target label.

1. Initialize the model with a constant value # L is the loss function, *e.g.*, squared error $L(y_i, \hat{y}_i) = (\hat{y}_i - y_i)^2$

$$h_0(\boldsymbol{x}) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. For t = 1 to T: # T trees
   1. Compute residuals (negative gradients) $r_i = -\frac{\partial L(y_i, h_{t-1}(\boldsymbol{x}_i))}{\partial h_{t-1}(\boldsymbol{x}_i)}, \forall i$
   2. Fit a **regression tree** f_t to the residuals (*i.e.*, train it using the training set $\{(\boldsymbol{x}_1, r_1), (\boldsymbol{x}_2, r_2), \dots, (\boldsymbol{x}_n, r_n)\}$)
   3. Compute the optimal step size $\gamma_t = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, h_{t-1}(\boldsymbol{x}_i) + \gamma \cdot f_t(\boldsymbol{x}_i))$
   4. Update the model $h_t(\boldsymbol{x}) = h_{t-1}(\boldsymbol{x}) + \lambda \cdot \gamma_t \cdot f_t(\boldsymbol{x})$ # λ is the learning rate
3. Output the final model $h_T(\boldsymbol{x})$

Gradient Boosting Machines

- **Training Procedure** (for regression, squared error loss function)

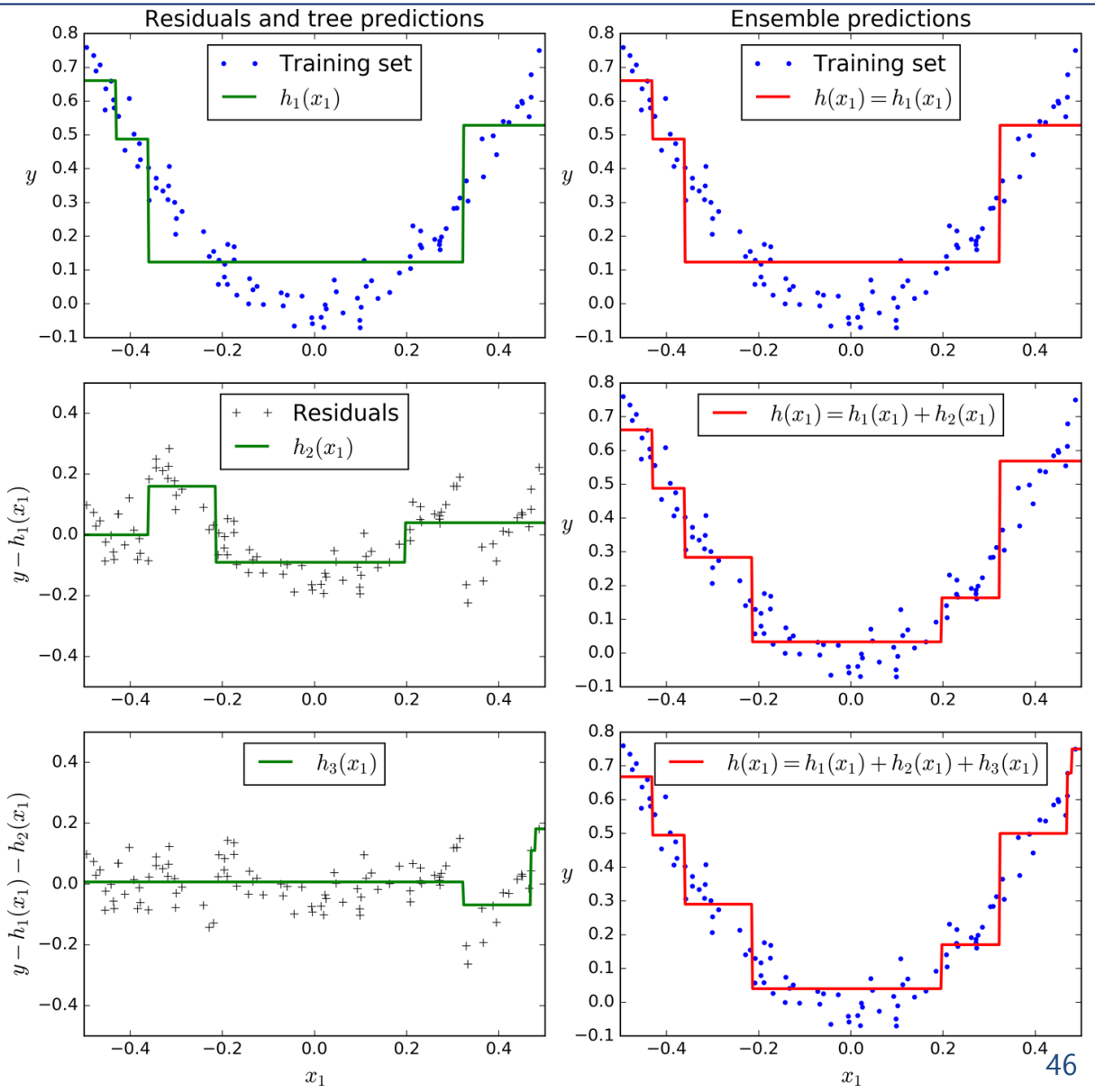
1. Initialize the model with a constant value

$$h_0(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} y_i$$

2. For t = 1 to T: # T trees
   1. Compute residuals (negative gradients) $r_i = y_i - h_{t-1}(\boldsymbol{x}_i), \forall i$
   2. Fit a **regression tree** f_t to the residuals (*i.e.*, train it using the training set $\{(\boldsymbol{x}_1, r_1), (\boldsymbol{x}_2, r_2), \dots, (\boldsymbol{x}_n, r_n)\}$)
   3. Compute the optimal step size $\gamma_t = \frac{\sum_{i=1}^{n} f_t(\boldsymbol{x}_i)(y_i - h_{t-1}(\boldsymbol{x}_i))}{\sum_{i=1}^{n} f_t(\boldsymbol{x}_i)^2}$
   4. Update the model $h_t(\boldsymbol{x}) = h_{t-1}(\boldsymbol{x}) + \lambda \cdot \gamma_t \cdot f_t(\boldsymbol{x})$ # λ is the learning rate
3. Output the final model $h_T(\boldsymbol{x})$

Gradient Boosting Machines

- **Example of building a gradient boosting machine (for regression)**



Gradient Boosting Machines

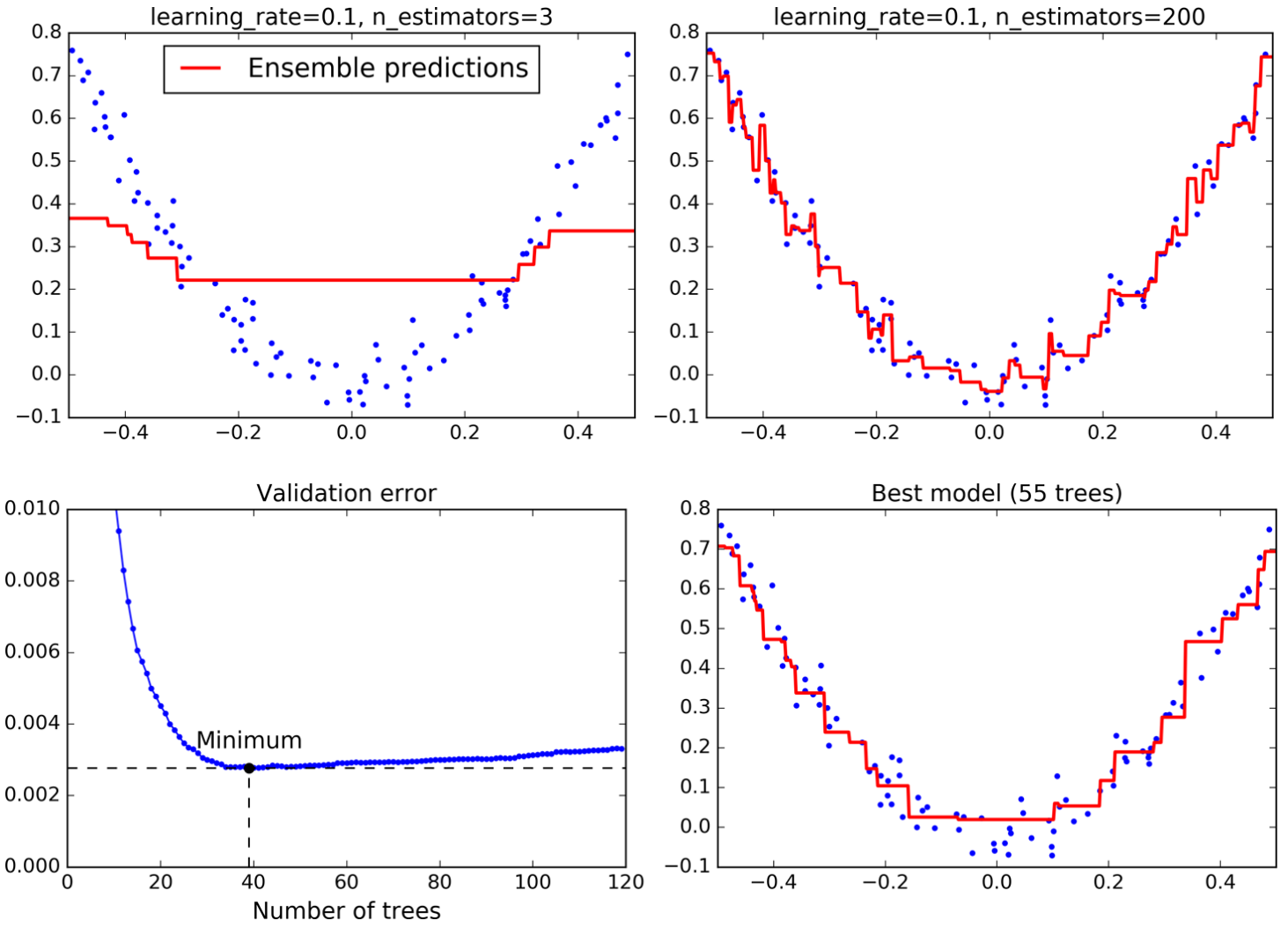
- **Main hyperparameters**
  - *n_estimators*: the number of trees in the ensemble
    - Adding more trees to the ensemble increases the model complexity, as the model has more chances to correct mistakes on the training set.
  - *learning_rate*: controls how strongly each tree tries to correct the mistakes of the previous trees
    - A higher learning rate means each tree can make stronger corrections, allowing for more complex models.
  - **Pre-pruning of trees:** stopping the creation of the tree early (*e.g.*, below). Usually, *max_depth* is set very low to reduce the complexity of each tree.
    - *max_depth*: limiting the maximum depth of the tree
    - *min_sample_leaf*: requiring a minimum number of points in a node to keep splitting it
    - *max_leaf_nodes*: limiting the maximum number of leaves

Gradient Boosting Machines

- **Gradient Boosting Machines are generally more sensitive to hyperparameter settings than Random Forests**
  - They can provide better accuracy if the hyperparameters are properly tuned.
  - A lower *learning_rate* means that more trees are needed to build a model of similar complexity.
    - A common practice is to set *n_estimators* based on time and memory constraints, then search over different *learning_rate* values.
  - To reduce overfitting,
    - We can either apply stronger pre-pruning or lower the learning rate.
    - The use of early stopping is recommended (not set as the default in the scikit-learn implementation).

Gradient Boosting Machines

- **Example: The effect of hyperparameter tuning (for regression)**



scikit-learn Practice: *GradientBoostingRegressor*

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

```
class sklearn.ensemble.GradientBoostingRegressor(*, loss='squared_error',
learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse',
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3,
min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, alpha=0.9,
verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1,
n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

Gradient Boosting for regression.

This estimator builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage a **regression tree** is fit on the negative gradient of the given loss function.

learning_rate	*float, default=0.1* Learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators. Values must be in the range [0.0, inf).
n_estimators	*int, default=100* The number of boosting stages to perform. Gradient boosting is fairly robust to over-fitting so a large number usually results in better performance. Values must be in the range [1, inf).
validation_fraction	*float, default=0.1* The proportion of training data to set aside as validation set for early stopping. Values must be in the range (0.0, 1.0). Only used if n_iter_no_change is set to an integer.
n_iter_no_change	*int, default=None* n_iter_no_change is used to decide if early stopping will be used to terminate training when validation score is not improving. By default it is set to None to disable early stopping. If set to a number, it will set aside validation_fraction size of the training data as validation and terminate training when validation score is not improving in all of the previous n_iter_no_change numbers of iterations. Values must be in the range [1, inf).

scikit-learn Practice: *GradientBoostingClassifier*

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

```
class sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss', learning_rate=0.1,
n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0,
init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None,
warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

Gradient Boosting for classification.

This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage ***n_classes_* regression trees** are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only **a single regression tree** is induced.

'log_loss' refers to binomial and multinomial deviance, the same as used in logistic regression. It is a good choice for classification with probabilistic outputs.

Discussion

- **The main hyperparameters of gradient boosting machines**
  - *n_estimators*: increasing it leads to a more complex model, which may lead to overfitting.
  - *learning_rate*: how strongly each tree tries to correct the mistakes of the previous trees.
  - Plus, the main hyperparameters of decision trees (especially for pre-pruning)

  * Typically chosen to achieve the highest performance on validation data

- **Strengths**
  - Gradient boosting machines are among the most powerful and widely used models for supervised learning on tabular data.
  - Gradient boosting machines share most of the benefits of decision trees.

- **Weaknesses**
  - Gradient boosting machines require careful tuning of the hyperparameters.
  - Gradient boosting machines don't tend to perform well on very high dimensional, sparse data.
  - It may take a long time to train.

Discussion

- **Gradient boosting machines and its variants are frequently the winning entries in machine learning competitions, and are widely used in industry.**
  - **XGBoost (Extreme Gradient Boosting)**
  - **LightGBM (Light Gradient Boosting Machine)**
  - **CatBoost (Categorical Boosting)**
  - **…**

* The key advantages of these variants come from enhancements in regularization, training efficiency, and categorical data handling, which make them more practical and effective than standard GBM implementations.