BM5702 MAKİNE ÖĞRENMESİNE GİRİŞ

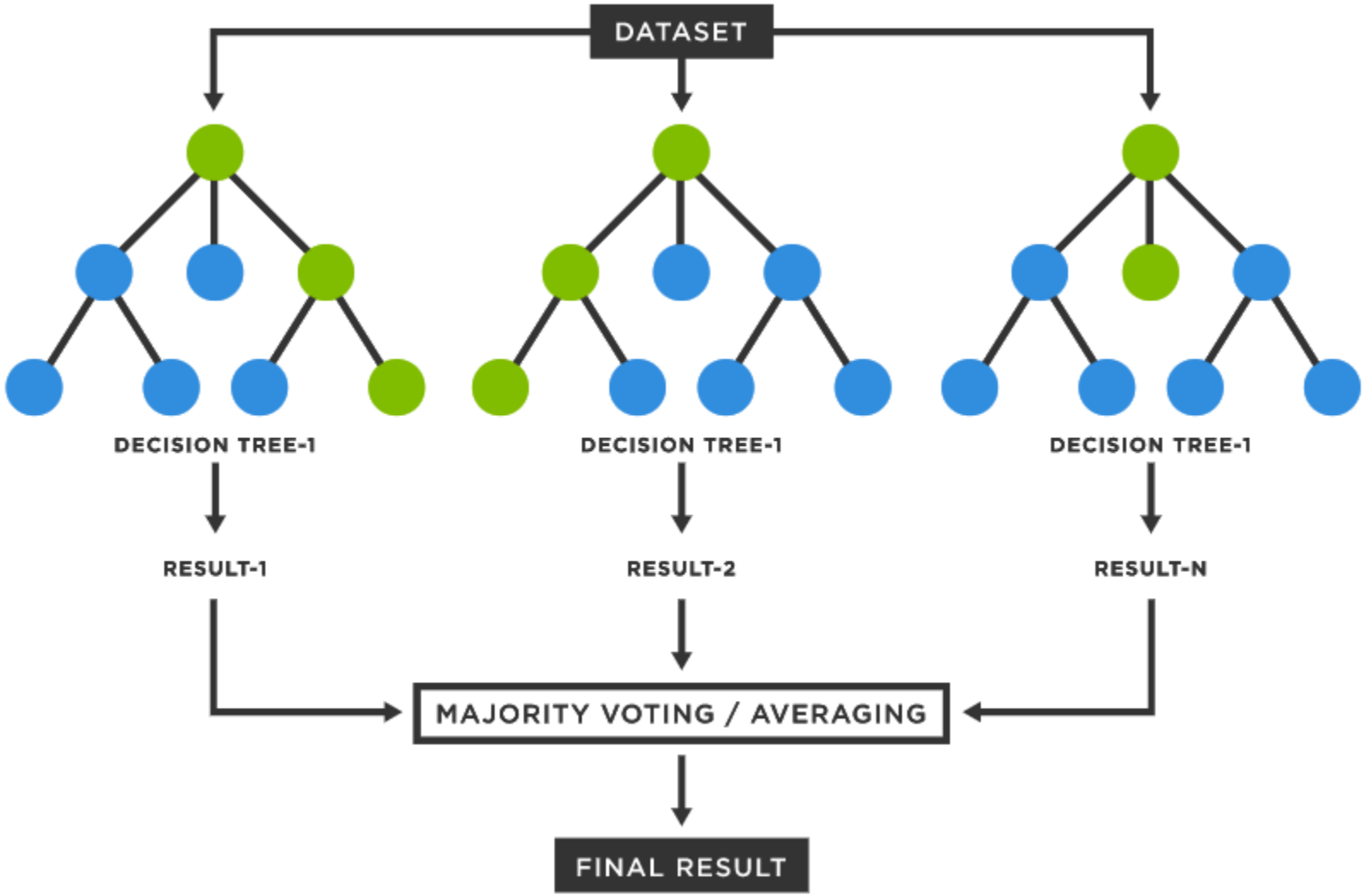
Hafta 10

Doç. Dr. Murtaza CİCİOĞLU

Ensembles of Decision Trees

- Ensembles are methods that combine multiple machine learning models to create more powerful models.
- There are many models in the machine learning literature that belong to this category, but there are two ensemble models that have proven to be effective on a wide range of datasets for classification and regression, both of which use decision trees as their building blocks:
- random forests and gradient boosted decision trees.

Random forests



Random forests

- As we just observed, a main drawback of decision trees is that they tend to overfit the training data. Random forests are one way to address this problem.
- The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.
- If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results.
- randomness → each tree is different
- There are two ways in which the trees in a random forest are randomized:
  - by selecting the data points used to build a tree
  - by selecting the features in each split test.

Random forests

- number of trees: n_estimators parameter of RandomForestRegressor or RandomForestClassifier
- These trees will be built completely independently from each other, and the algorithm will make different random choices for each tree to make sure the trees are distinct.
- bootstrap sample
- n_samples data points, we repeatedly draw an example randomly with replacement (meaning the same sample can be picked multiple times), n_samples times.
- This will create a data- set that is as big as the original dataset, but some data points will be missing from it (approximately one third), and some will be repeated.

Random forests

- To illustrate, let’s say we want to create a bootstrap sample of the list ['a', 'b', 'c', 'd'].
- A possible bootstrap sample would be ['b', 'd', 'd', 'c'].
- Another possible sample would be ['d', 'a', 'd', 'a'].
- Next, a decision tree is built based on this newly created dataset.
- Instead of looking for the best test for each node, 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. The number of features that are selected is controlled by the max_features parameter.
- This selection of a subset of features is repeated separately in each node, so that each node in a tree can make a decision using a different subset of the features.

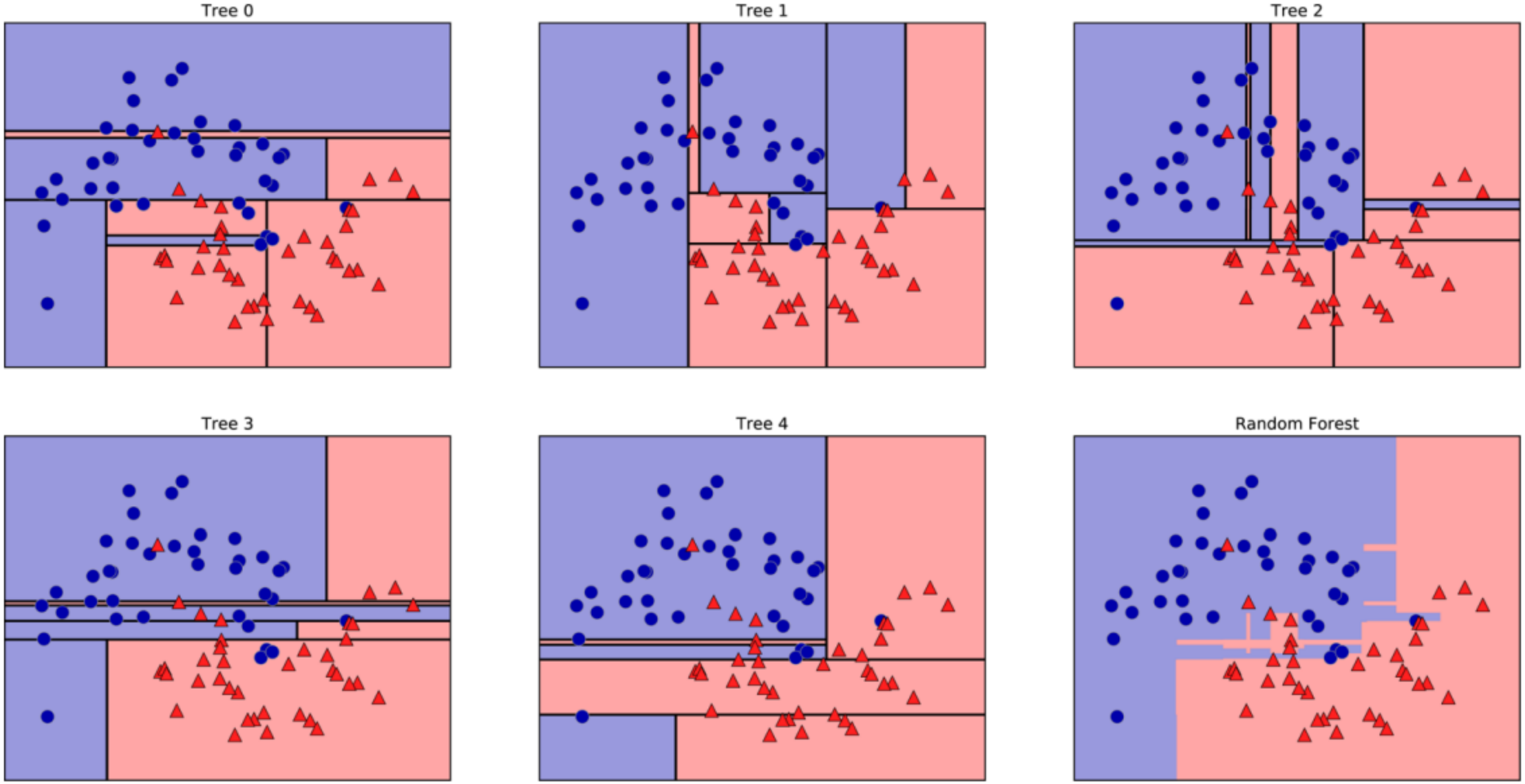
Random forests

- A critical parameter in this process is max_features.
- If we set max_features to n_features, that means that each split can look at all features in the dataset, and no randomness will be injected in the feature selection (the randomness due to the bootstrapping remains, though).
- If we set max_features to 1, that means that the splits have no choice at all on which feature to test, and can only search over different thresholds for the feature that was selected randomly.
- Therefore, 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.

Random forests

- To make a prediction using the random forest, the algorithm first makes a prediction for every tree in the forest.
- For regression, we can average these results to get our final prediction.
- For classification, a "soft voting" strategy is used. This means each algorithm makes a "soft" prediction, providing a probability for each possible output label. The probabilities predicted by all the trees are averaged, and the class with the highest probability is predicted.
- The trees that are built as part of the random forest are stored in the estimator_ attribute.

Random forests



Strengths, weaknesses, and parameters

- Random forests for regression and classifica- tion are currently among the most widely used machine learning methods. They are very powerful, often work well without heavy tuning of the parameters, and don’t require scaling of the data.
- It is basically impossible to interpret tens or hundreds of trees in detail, and trees in random forests tend to be deeper than decision trees (because of the use of feature subsets).
- While building random forests on large datasets might be somewhat time consuming, it can be parallelized across multiple CPU cores within a computer easily.
- You can set n_jobs=-1 to use all the cores in your computer
- If you want to have reproducible results, it is important to fix the random_state.

Strengths, weaknesses, and parameters

- Random forests don't tend to perform well on very high dimensional, sparse data, such as text data. For this kind of data, linear models might be more appropriate.
- Random forests usually work well even on very large datasets, and training can easily be parallelized over many CPU cores within a powerful computer. However, random forests require more memory and are slower to train and to predict than linear models. If time and memory are important in an application, it might make sense to use a linear model instead.
- The important parameters to adjust are n_estimators, max_features, and possibly pre-pruning options like max_depth.

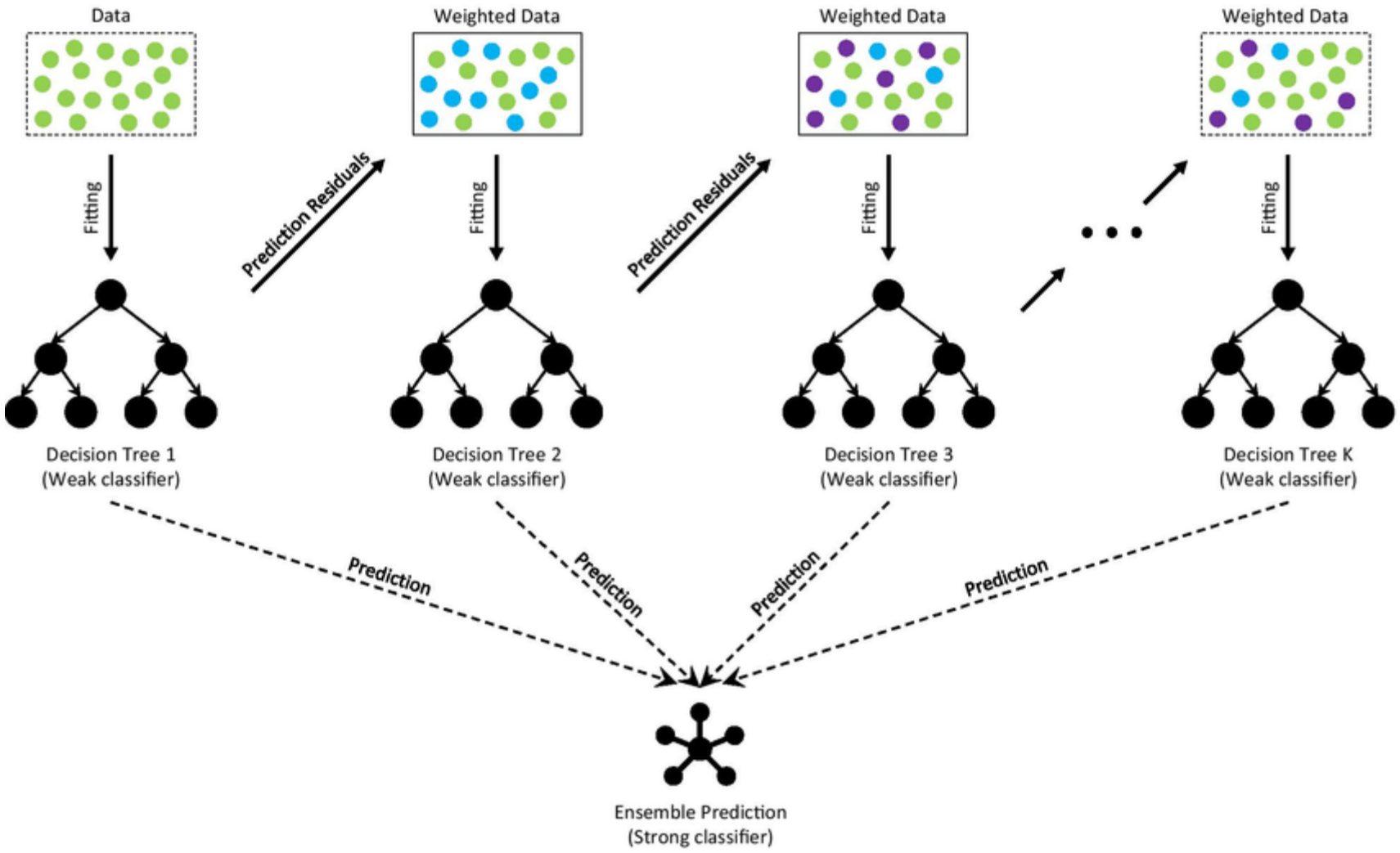
Strengths, weaknesses, and parameters

- n_estimators, larger is always better. Averaging more trees will yield a more robust ensemble by reducing overfitting. However, there are diminishing returns, and more trees need more memory and more time to train. A common rule of thumb is to build “as many as you have time/memory for.”
- max_features determines how random each tree is, and a smaller max_features reduces overfitting. In general, it’s a good rule of thumb to use the default values: max_features=sqrt(n_features) for classification
- max_features=n_features for regression.
- Adding max_features or max_leaf_nodes might sometimes improve performance.
- It can also drastically reduce space and time requirements for training and prediction.

Gradient boosted regression trees (gradient boosting machines)

- The gradient boosted regression tree is another ensemble method that combines multiple decision trees to create a more powerful model.
- Despite the “regression” in the name, these models can be used for regression and classification.
- In contrast to the random forest approach, gradient boosting works by building trees in a serial manner, where each tree tries to correct the mistakes of the previous one.
- By default, there is no randomization in gradient boosted regression trees; instead, strong pre-pruning is used.
- Gradient boosted trees often use very shallow trees, of depth one to five, which makes the model smaller in terms of memory and makes predictions faster.

Gradient boosted regression trees (gradient boosting machines)



Gradient boosted regression trees

- The main idea behind gradient boosting is to combine many simple models (in this context known as weak learners), like shallow trees. Each tree can only provide good predictions on part of the data, and so more and more trees are added to iteratively improve performance.
- Gradient boosted trees are frequently the winning entries in machine learning competitions, and are widely used in industry. They are generally a bit more sensitive to parameter settings than random forests, but can provide better accuracy if the parameters are set correctly.
- 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.
- Adding more trees to the ensemble, which can be accomplished by increasing n_estimators, also increases the model complexity, as the model has more chances to correct mistakes on the training set

Gradient boosted regression trees

- As both gradient boosting and random forests perform well on similar kinds of data, a common approach is to first try random forests, which work quite robustly.
- If random forests work well but prediction time is at a premium, or it is important to squeeze out the last percentage of accuracy from the machine learning model, moving to gradient boosting often helps.
- If you want to apply gradient boosting to a large-scale problem, it might be worth looking into the xgboost package and its Python interface, which at the time of writing is faster (and sometimes easier to tune) than the scikit-learn implementation of gradient boosting on many datasets.

Strengths, weaknesses, and parameters

- most powerful and widely used models for supervised learning
- require careful tuning of the parameters and may take a long time to train
- tree-based models, it also often does not work well on high-dimensional sparse data
- n_estimators, and the learning_rate, which controls the degree to which each tree is allowed to correct the mistakes of the previous trees. These two parameters are highly interconnected, as a lower learning_rate means that more trees are needed to build a model of similar complexity.
- In contrast to random forests, where a higher n_estimators value is always better, increasing n_estimators in gradient boosting leads to a more complex model, which may lead to overfitting.
- A common practice is to fit n_estimators depending on the time and memory budget, and then search over different learning_rates.