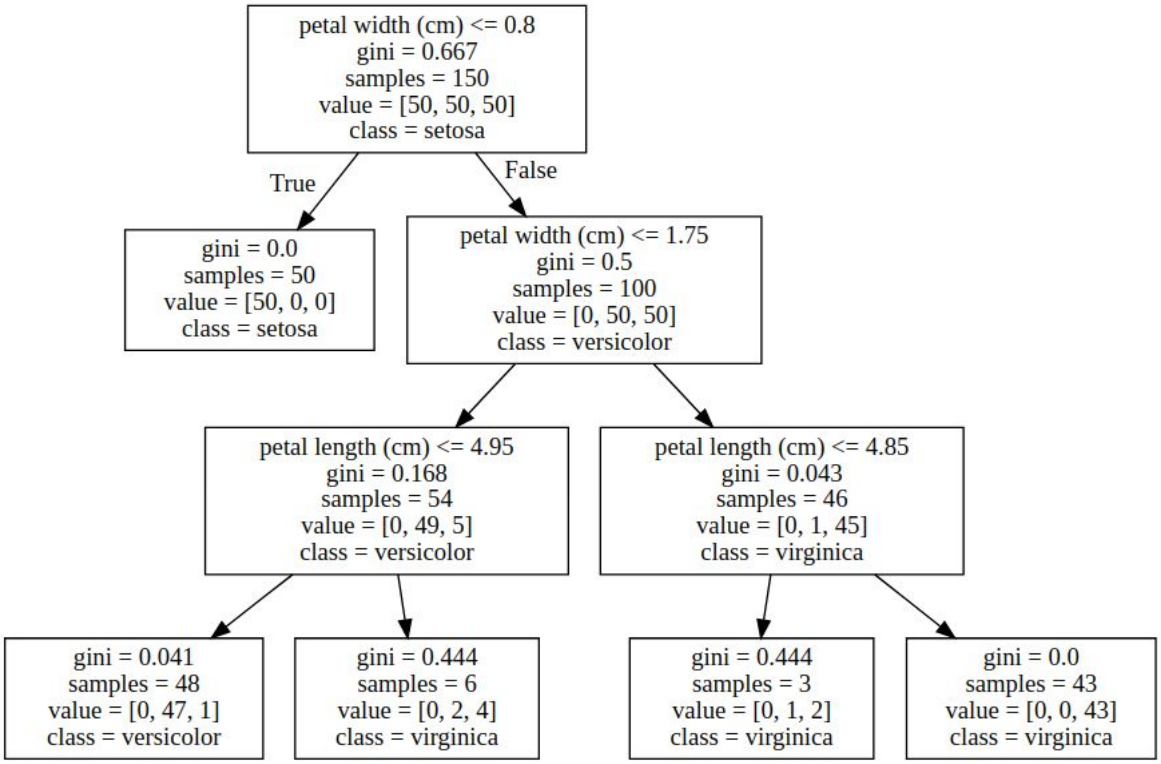
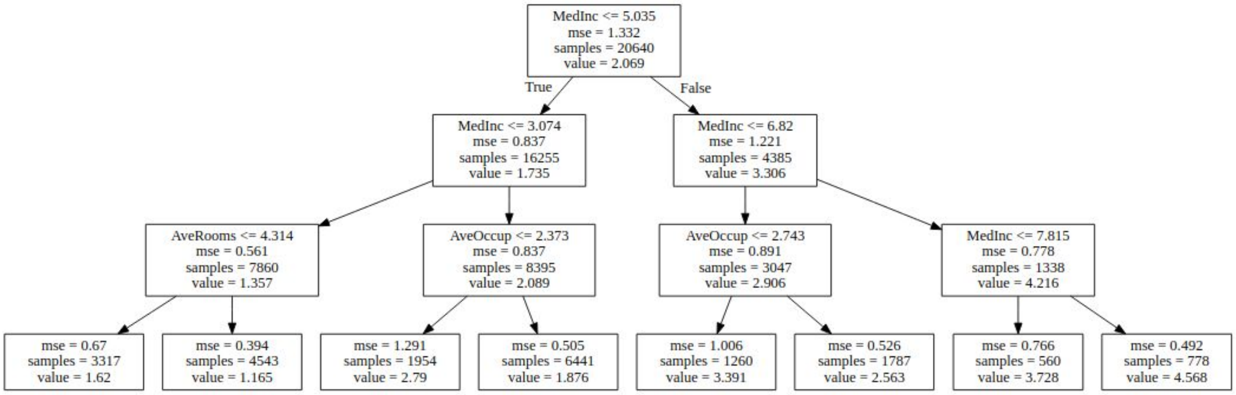
Week 7 - Tree-based model

1. Decision tree - 30m

- Both used for classification & regression
- Is a series of questions with yes/no answers, where the resulting tree structure contains all the combinations of responses. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
- Training algorithm: at each feature, construct a threshold and choose this via the greatest drop weighted error metric **- delta before/after!!!** (and use this threshold to divide into two sets - reduce Gini impurity or MSE - variance). The greater the node purity, the lower the Gini metric.
- This algorithm is called a “greedy” algorithm that can cause locally optimal choice (optimal at current round only). Although greedy algorithms do not always converge to global but better time complexity (faster computation).

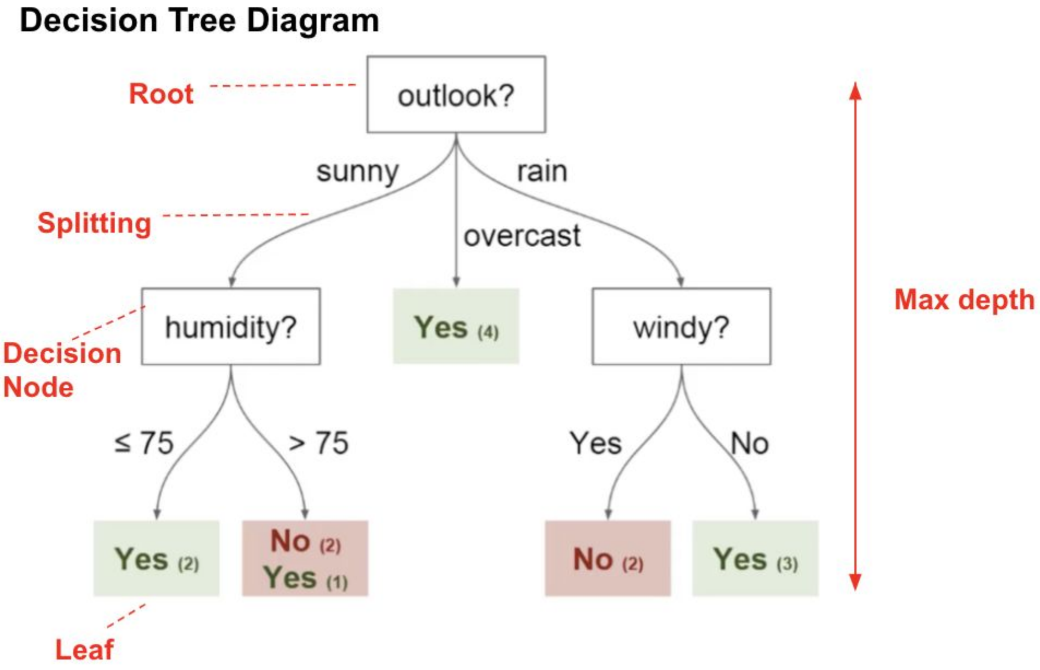
  For classification, the partitions are chosen to separate the different classes while in regression, the partitions are picked to reduce the variance of sample labels.





https://github.com/manhitv/worldquant-university-unit-II/blob/master/ML_Tree_Based_Models.ipynb (example for classification - Gini, most common values and regression - Mse, average values)

- Hyperparameters: Leaf - end, Split - higher floor!
    + Max_depth: key param
    + Max_features: number of features to consider best split
    + Min_sample_split: min samples considered to split internal node
    + Min_sample_leaf: min samples required for a leaf.



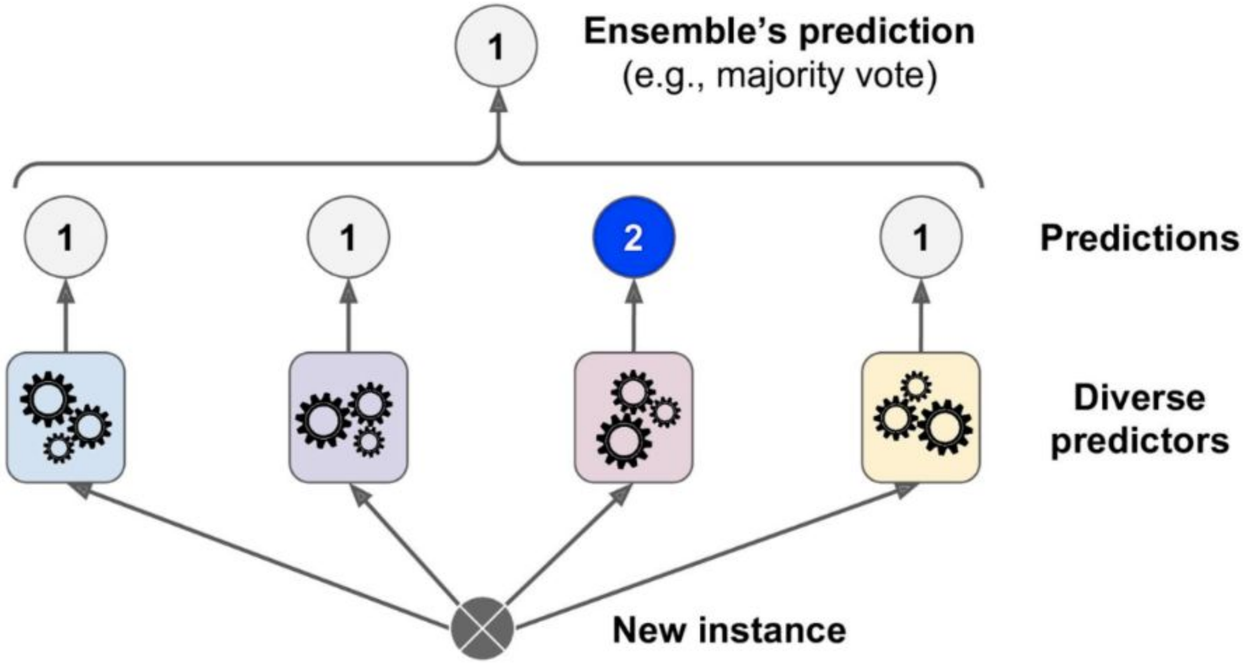
- Time complexity: training time (np*O(log(n)) (log(n) - number of level / np - number of possible nodes) and time for making prediction (O(log(n)) - make log(n) decisions.
- Tree based models are popular because they mimic human decision making processes, work well for a large class of problems, naturally handle multiclassification, and handle a mix of categorical and numerical data.
- Pros and cons:

+ Advantages: easy to interpret and visualize, requires fewer data preprocessing from the user, can be used for feature engineering such as predicting missing values, no assumptions about distribution.
  The transparency of a model is often called its *explicability*. Models with low explicability are often referred to as "*black boxes*" and are difficult to derive insight over the process they are modeling. (explainable ML field)
+ Disadvantages: sensitive to noisy data, unstable (sensitive to small variance of data, this can be reduced by bagging and boosting algorithms), biased with imbalance dataset (need to balance out before creating the tree).

*Question: For decision trees, there is no need to scale your data. Why is this?

2. Ensemble models - Tree-based algorithms

- ML models that use more than one predictor to arrive at a prediction. A group of predictors form an ensemble. In general, ensemble models perform better than using a single predictor.
- VotingClassifier: is this Ensemble models?



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

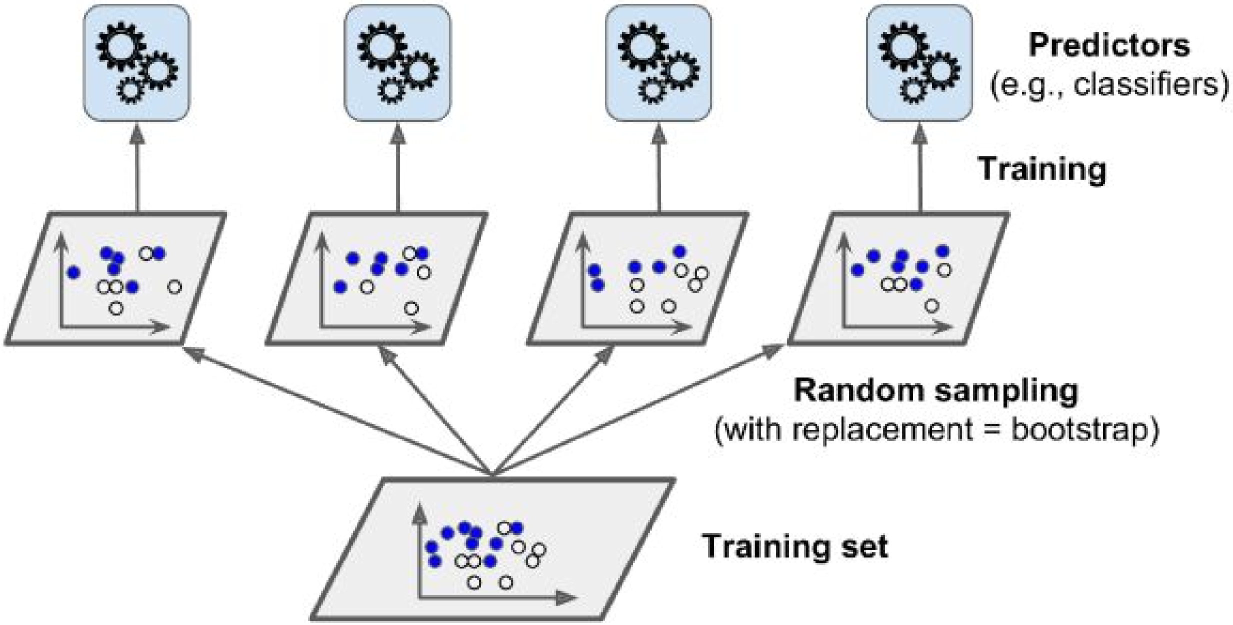
If all classifiers are able to estimate class probabilities (i.e., they all have a predict_proba() method), then you can tell Scikit-Learn to predict the class with the highest class probability, averaged over all the individual classifiers.

https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier

Ensemble methods work best when the predictors are as independent from one another as possible. One way to get diverse classifiers is to train them using very different algorithms. This increases the chance that they will make very different types of errors, improving the ensemble's accuracy.

- There are 3 types of ensemble models: bagging, boosting and blending

- Bagging: e.g. random forest



After getting many predictors → aggregate function by statistical mode (average for regression, most frequent value for classification).

Predictors can all be trained in parallel, via different CPU cores or even different servers. Similarly, predictions can be made in parallel. This is one of the reasons bagging and pasting are such popular methods: they scale very well.

Reference at sklearn (rarely used): sklearn.ensemble.BaggingClassifier — scikit-learn 0.23.2 documentation

Bootstrapping is a general statistical technique to generate "new" data sets with a single set by random sampling with *replacement*.

`n_estimators` - The number of trees in the forest - increase performance till it's maximum.

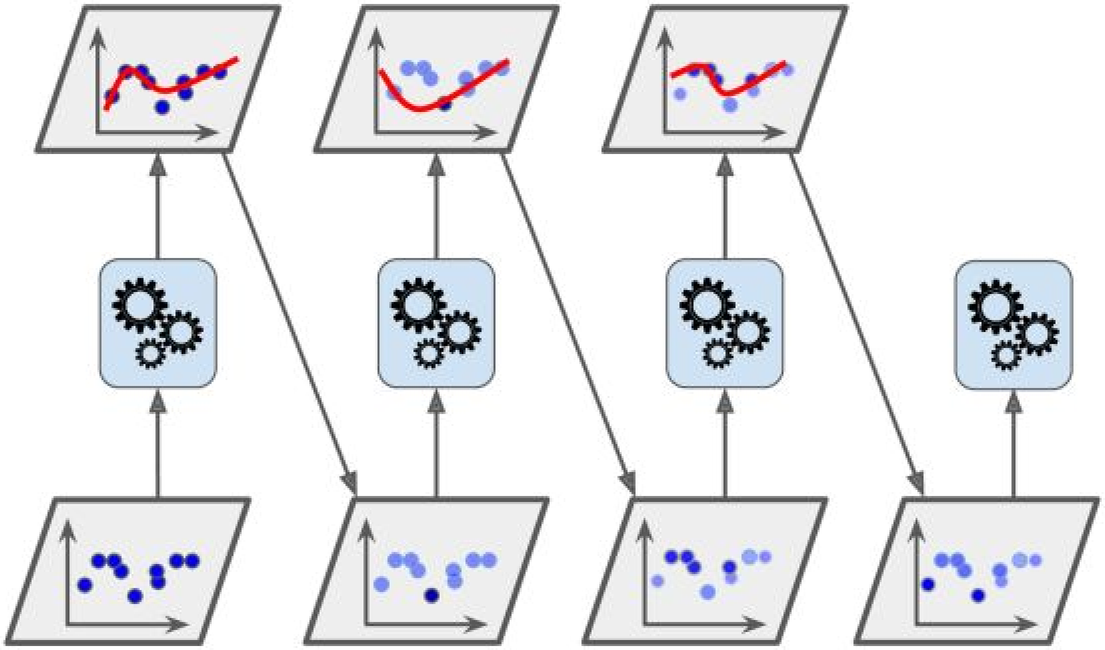
Random Forest: a random subset of the features are selected to determine which one to use for a node split.

Extremely randomized trees: instead of considering the optimal split point *for each* selected feature, a candidate for the split for each feature is chosen at *random*. From these randomly chosen values, the best is chosen to perform the split. (might use more trees, run to compare two classifiers)

- Boosting: combine multiple "weak learner" (boost) to form a strong learner.

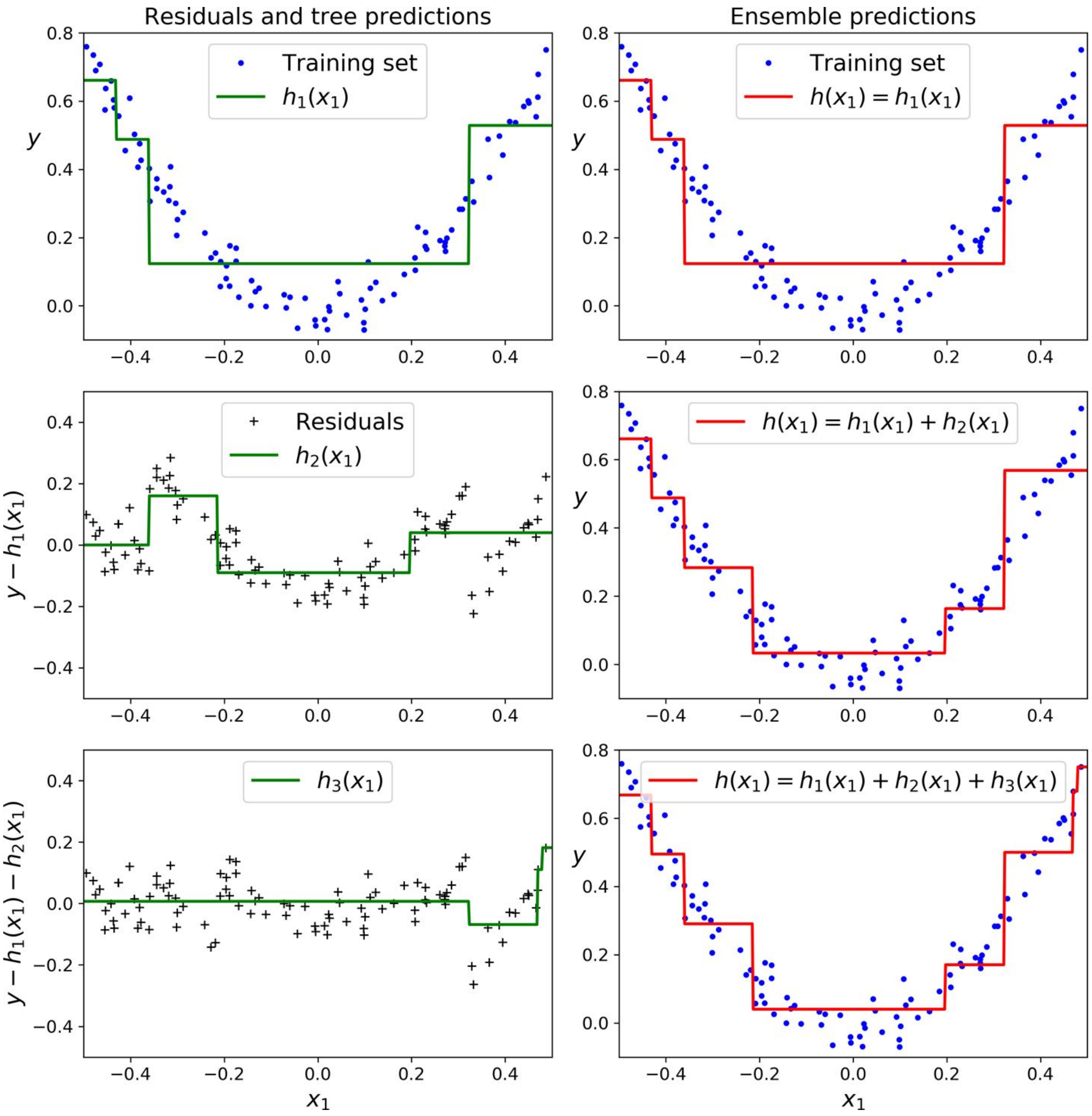
The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.

*Adaboost (adaptive boosting): pay a bit more attention to the training instances that the predecessor underfitted. When training an AdaBoost classifier, the algorithm first trains a base classifier (such as a Decision Tree) and uses it to make predictions on the training set. The algorithm then increases the relative weight of misclassified training instances. Then it trains a second classifier, **using the updated weights**, and again makes predictions on the training set, updates the instance weights, and so on.

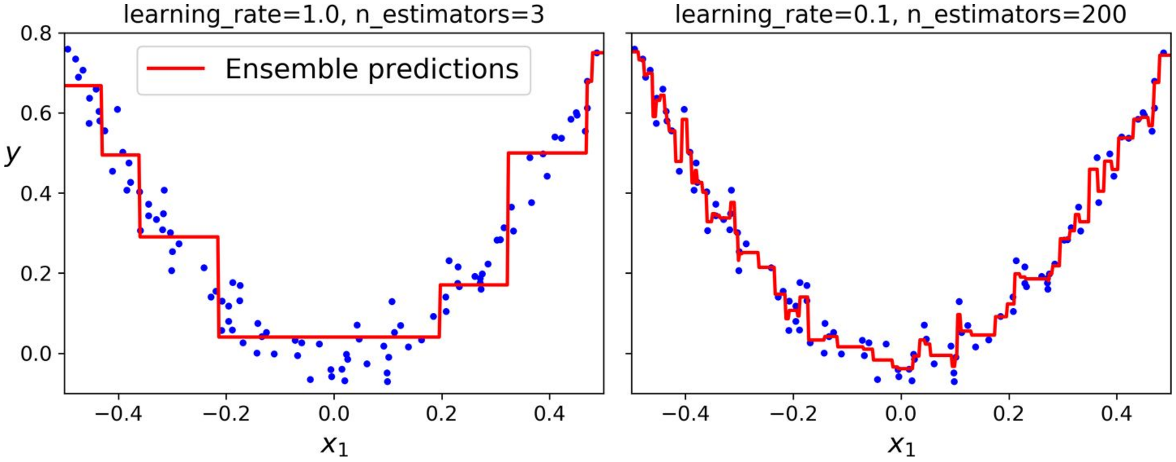


There is one important drawback to this sequential learning technique: it cannot be parallelized (or only partially), since each predictor can only be trained after the previous predictor has been trained and evaluated. It does not scale as well as bagging or pasting.

*Gradient boosting: 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

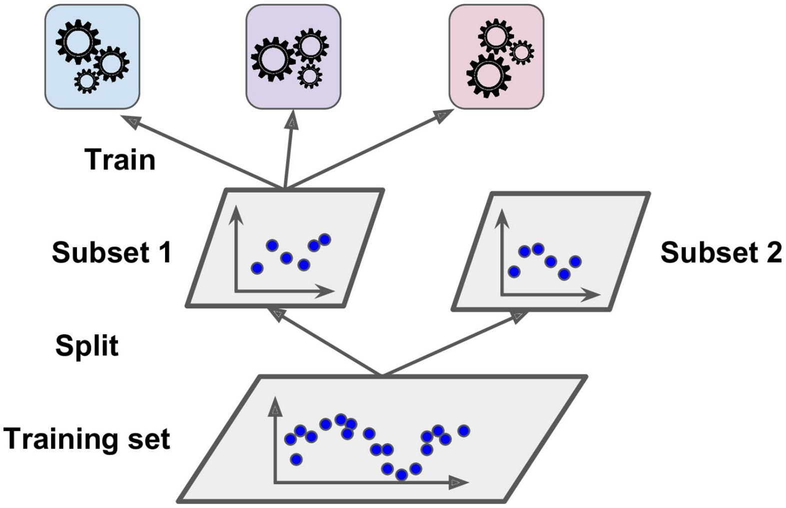


Affect of hyperparameters:

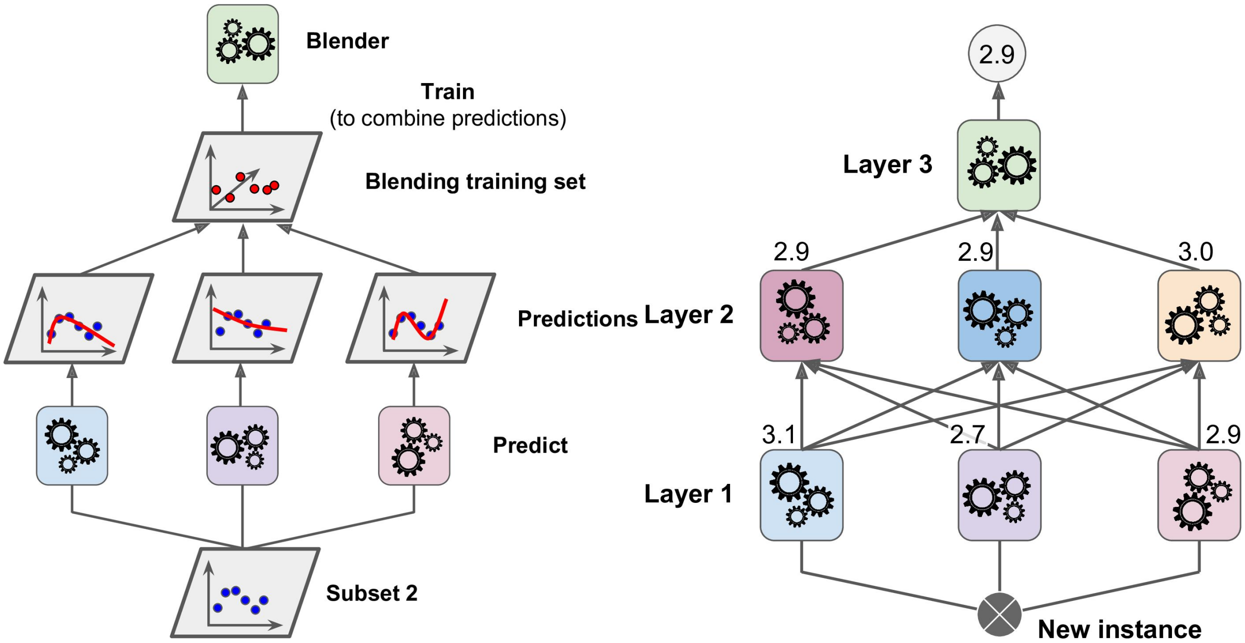


- + Blending:

*Splitting:



*Blender & predict:



- Feature importances: Tree-based algorithms support finding importance of features via feature_importance_ method.

  Scikit-Learn measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest). More precisely, it is a weighted average, where each node's weight is equal to the number of training samples that are associated with it. Scikit-Learn computes this score automatically for each feature after training, then it scales the results so that the sum of all importances is equal to 1.

3. Exercise

- Questions:
    - If a Decision Tree is overfitting the training set, is it a good idea to try decreasing max_depth?
    - If your AdaBoost ensemble underfits the training data, which hyperparameters should you tweak and how?
    - If your Gradient Boosting ensemble overfits the training set, should you increase or decrease the learning rate? → normal cases
- Practice use ensemble for **Make_moons** dataset (Logistic Regression, SVM, Random Forest) → Homework: used for **California housing dataset**

4. Lab

- Re-introduce workflow of a Machine Learning project

B1. Define the scope of work and objective

+ Context: How is your solution be used?
+ Metrics, contraints?
+ If manually?
+ Back test: list the available assumptions, and verify if possible.

B2. Get the data

+ Where, Format/Store data
+ Check the overview (size, type, sample, description, statistics)
+ Data cleaning

B3. EDA & Data transformation

+ Attribute and its characteristics (missing values, type of distribution, usefulness)
+ Visualize the data
+ Study the correlations between attributes
+ [Feature selection], Feature Engineering, [Feature scaling]
+ Write functions for all data transformations

B4. Train models

+ Automate as much as possible
+ Train promising models quickly using standard parameters. Measure and compare their performance
+ Error analysis
+ Shortlist the top three of five most promising models, preferring models that make different types of errors.

B5. Fine-tuning

+ Treat data transformation choices as hyperparameters, expecially when you are not sure about them (e.g., replace missing values with zeros or with the median value)
+ Unless there are very few hyperparameter values to explore, prefer random search over grid search.
+ Try ensemble methods
+ Test your final model on the test set to estimate the generalizaiton error. Don't tweak your model again, you would start overfitting the test set.

- Fashion MNIST dataset