Machine learning workshops for material scientists

Lecture 7: Supervised tree model with Python

November 7, 2022

Sira Sriswasdi, PhD

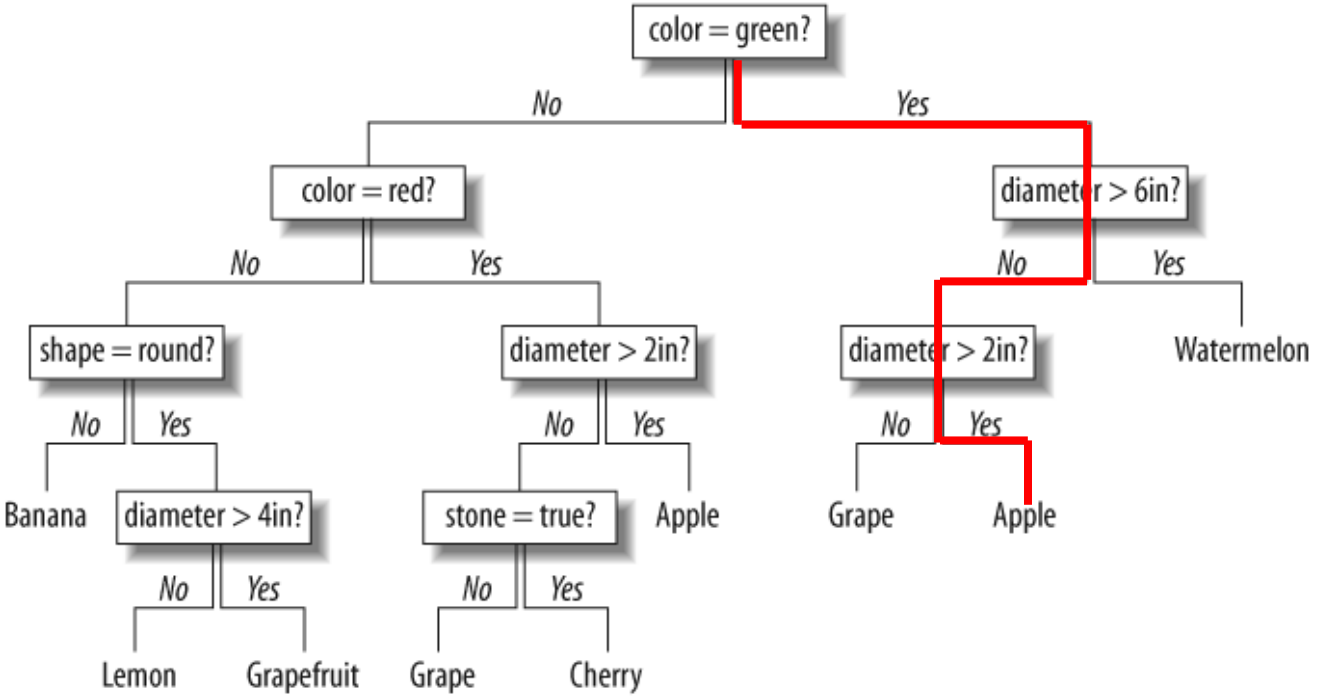
- Research Affairs
- Center of Excellence in Computational Molecular Biology (CMB)
- Center for Artificial Intelligence in Medicine (CU-AIM)

Today's goals

- Quick recap of tree models
- Key hyperparameters for tree models
- Get to know Python library
- Practice on Colab

Decision tree

Classification tree



Source: Programming collective intelligence by Toby Segaran

Regression tree

Predictors | Target

Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

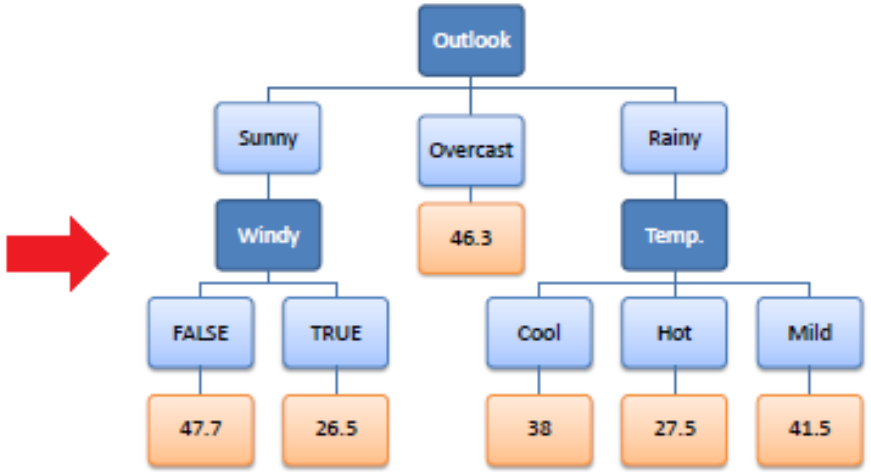


Image from saedsayad.com

- Predict using the average or median from data points in each branch

Building the tree

100 healthy, 100 sick

85 healthy
5 sick

15 healthy
95 sick

100 healthy, 100 sick

45 healthy
51 sick

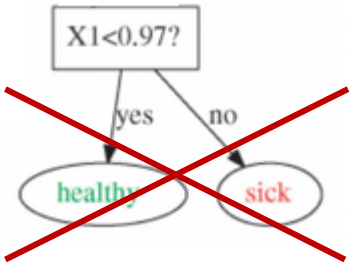
55 healthy
49 sick

- Objective for making a split
  - Gini impurity: $\sum p(1-p)$ or Entropy: $-\sum p \ln(p)$
  - Lowest for a perfect split, highest for a 50-50 split

- Search for feature and cutoff values that decrease the impurity score

Regularizing tree complexity

1. Too few samples to make a split



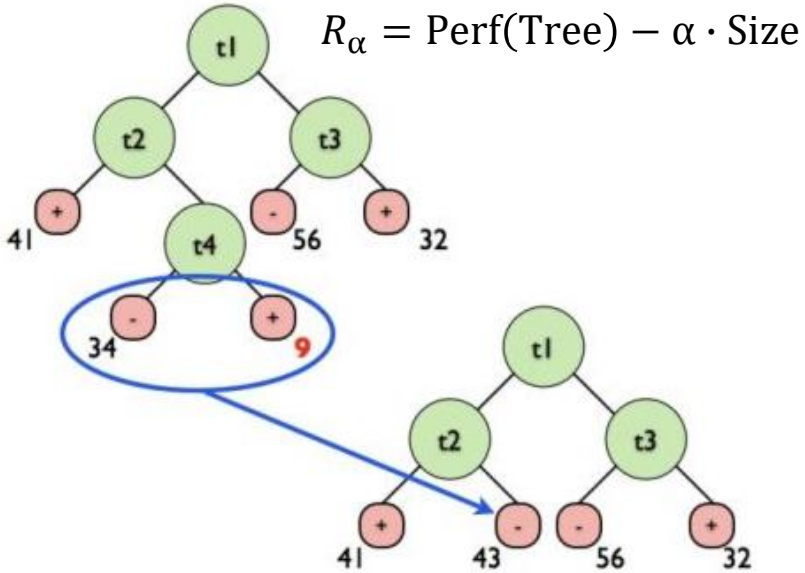
3. Impurity or entropy does not improve much after the split

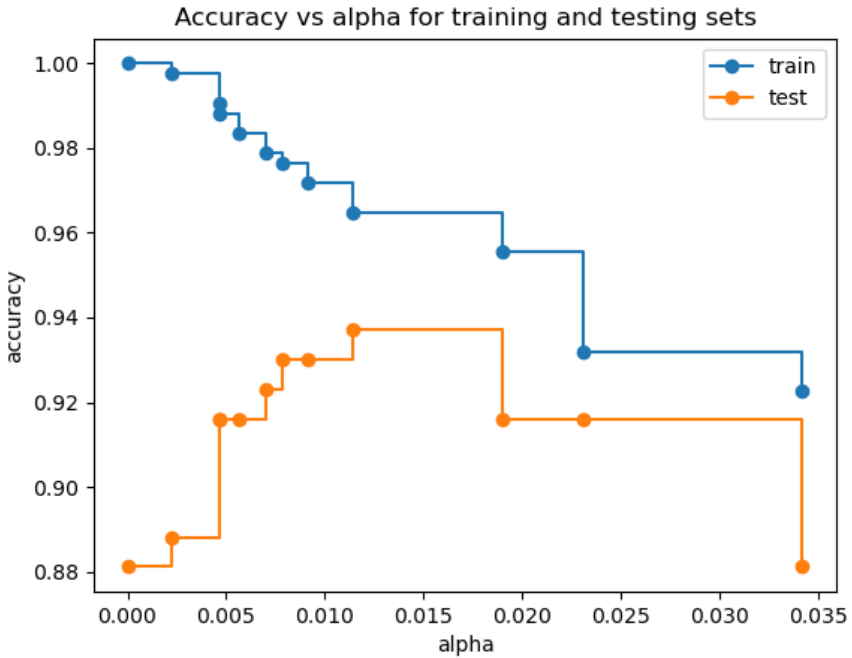
2. Too few samples on either branch

- Restrict tree size (number of nodes, depth)
- Require certain reduction in impurity score
- Require minimal number of samples to support a split

Tree pruning

$$R_\alpha = \text{Perf}(\text{Tree}) - \alpha \cdot \text{Size}$$





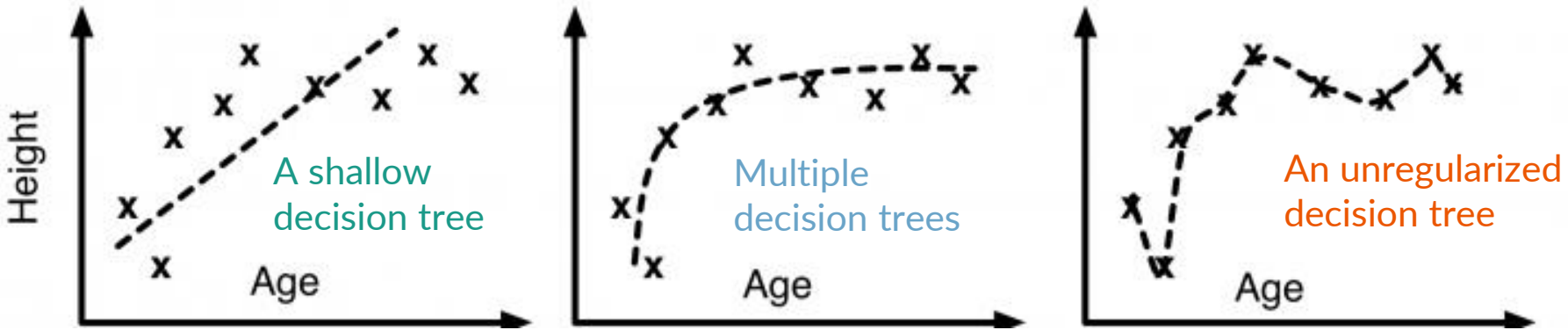
Patel, N. and Upadhyay, S. “Study of Various Decision Tree Pruning Methods with their Empirical Comparison in WEKA” IJCA 2012

Image from scikit-learn,org

Ensemble approach

Boosting solves underfitting

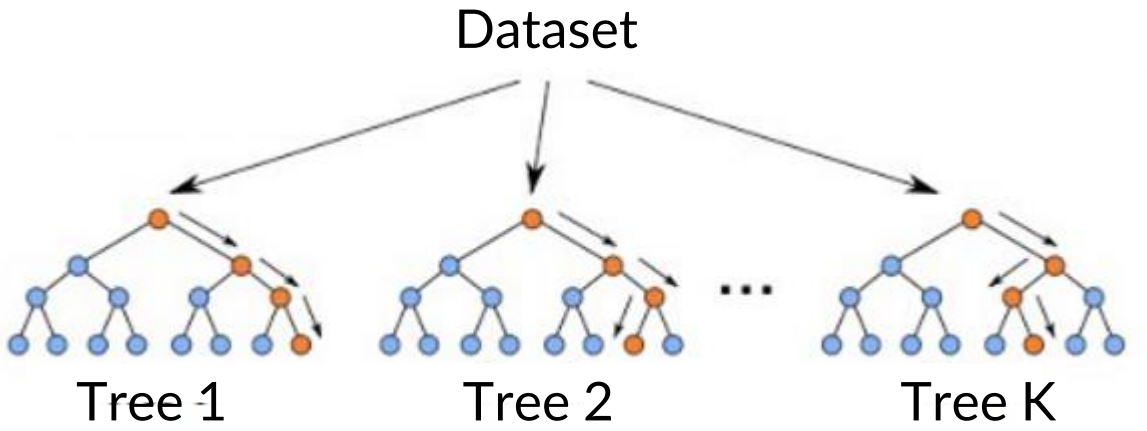
Bagging prevents overfitting



https://realkm.com/2018/04/23/optimization-and-complexity-the-cost-of-complexity-systems-thinking-modelling-series/

Bagging

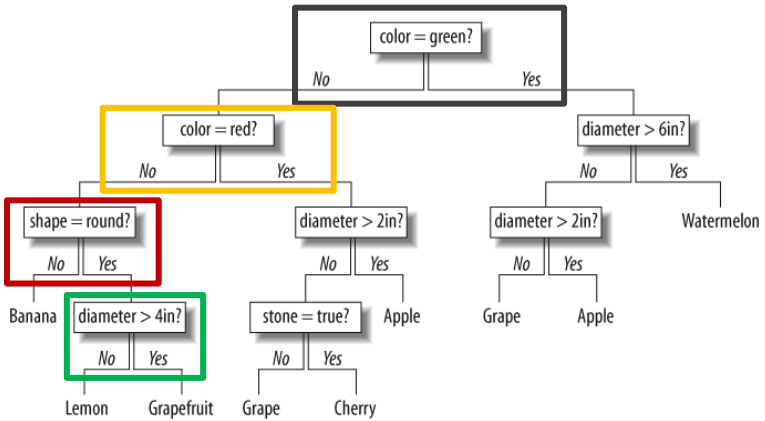
Sample bagging



- Sample 80% of the dataset to train each decision tree
- Each tree may be overfit to different part of the dataset
- But the consensus should be correct

Feature bagging

- Force a decision tree to utilize multiple features
  - Similar to ridge and LASSO
- For each tree
  - Only consider *N* random features
  - Only consider fraction *f* of all features
- For each splitting within a tree



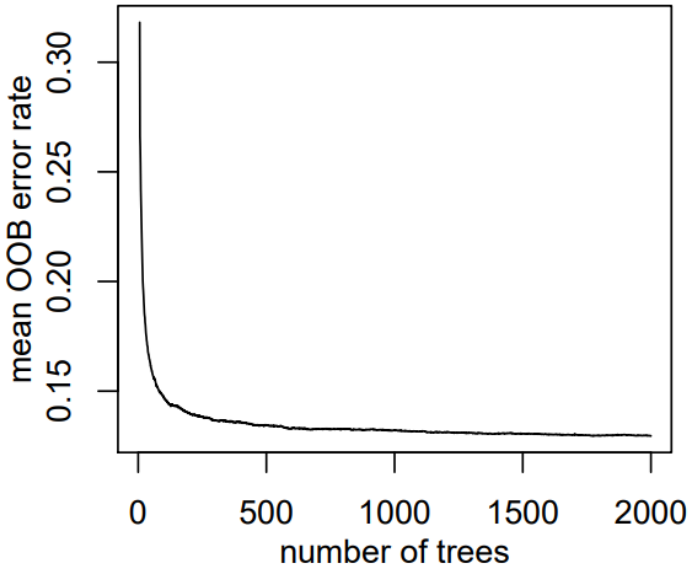
Random forest in Python

sklearn.ensemble.RandomForestClassifier

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*) [source]

- n_estimators = number of tree
- max_depth = the depth of the tree
- criterion / min_impurity_decrease = control impurity
- min_samples_split / min_samples_leaf = number of samples to support a split
- max_features = number of random features to consider at each split
- ccp_alpha = pruning strength

Random forest tuning tips



- Set n_estimators to high number (300-500)
- Good performance without tuning other hyperparameters

Random forest tuning tips

params	mean_test_accuracy	std_test_accuracy
{'max_depth': 10, 'max_features': 0.2, 'min_sa...	0.538946	0.010295
{'max_depth': 10, 'max_features': 0.3000000000...	0.534225	0.011477
{'max_depth': 10, 'max_features': 0.2, 'min_sa...	0.533124	0.011708
{'max_depth': 10, 'max_features': 0.4, 'min_sa...	0.531550	0.011740
{'max_depth': 10, 'max_features': 0.3000000000...	0.530921	0.011460

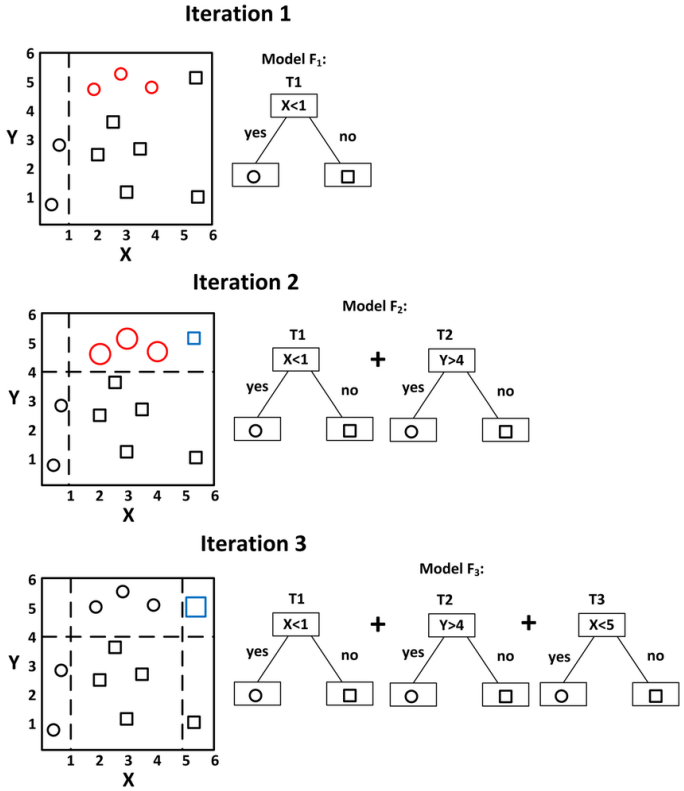
- No need to tune aggressively
- Difference among top hyperparameters are often less than 1-3% (well with in the standard deviation across cross-validation folds)

Random forest tuning tips

- Default max_features = sqrt(number of features) generally works ok
  - Try tuning on steps of 10% (0.1, 0.2, 0.3, 0.4, ...)
- max_depth depends on the complexity of the task
  - Try tuning on (2, 5, 10, 20, None)
- Small ccp_alpha is generally beneficial
  - Try tuning on log scale 0.1, 0.01, ...
- criterion / min_impurity_decrease typically don't need tuning
- min_samples_split / min_samples_leaf
  - Only one of the two needs to be tuned

Boosting

Boosting for classification = weight the error



- Ensemble predictor = weighted average
  - $C_n(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots + \alpha_n f_n(x)$
- Weight samples for training the n^{th} model based on errors made by the first $n-1$ models
- Confidence in the n^{th} model, α_n, is based on the performance of $f_n(x)$ on errors made by the first $n-1$ models

Boosting for regression = fit the residual

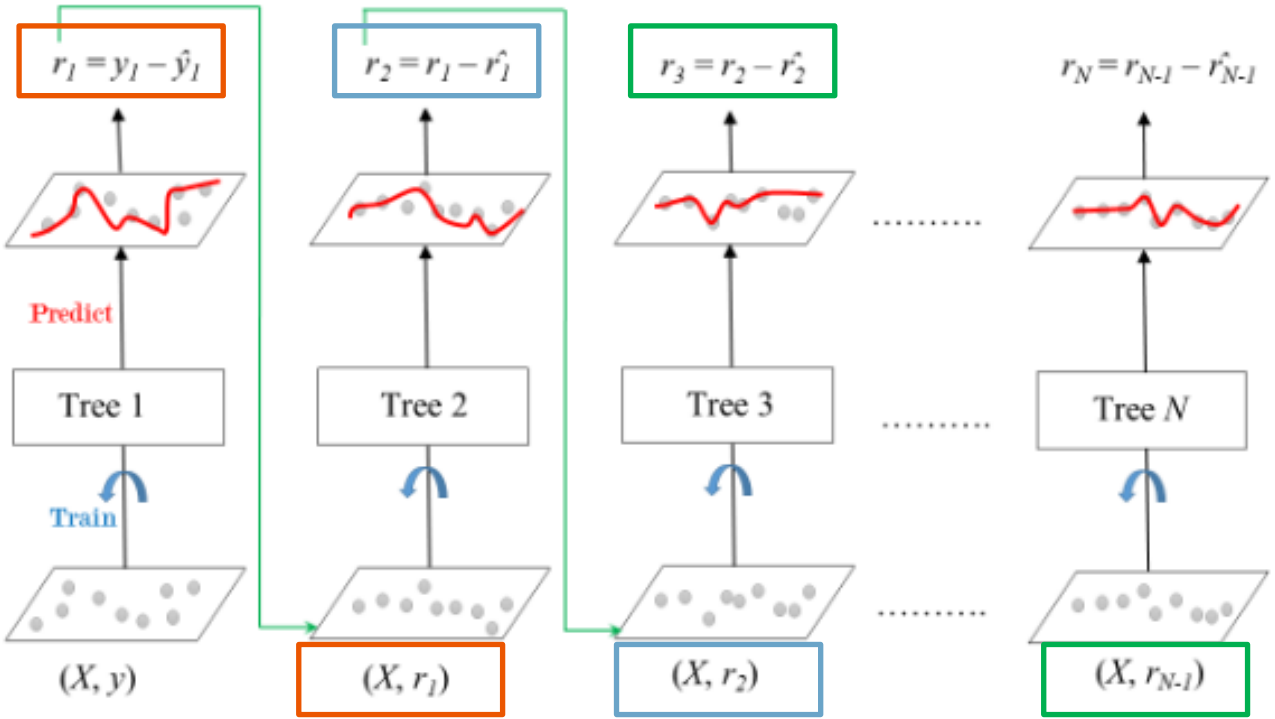


Image from geeksforgeeks.org

Boosting in Python (scikit-learn)

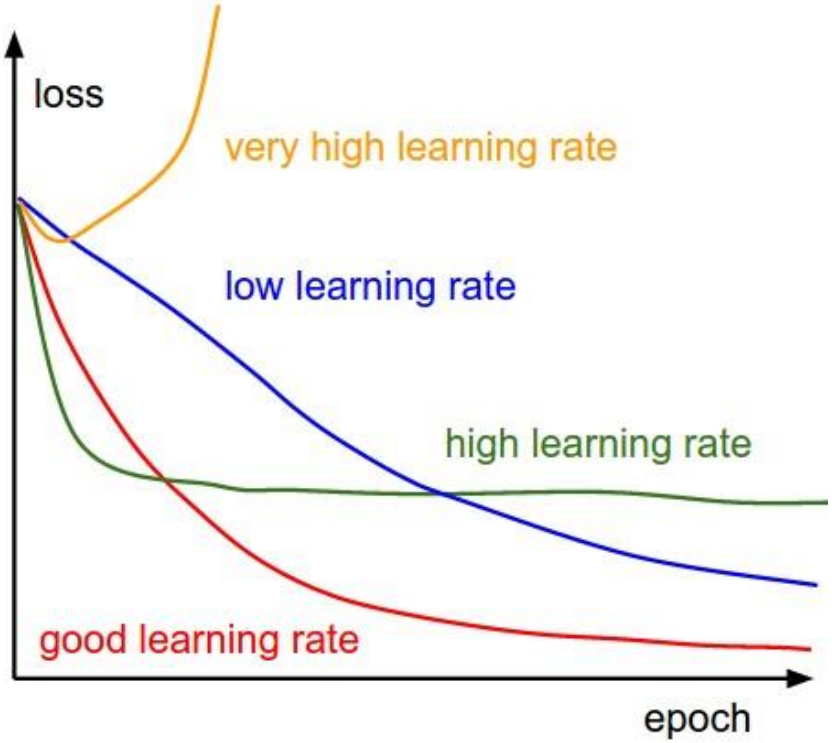
sklearn.ensemble.AdaBoostClassifier

class `sklearn.ensemble.`**`AdaBoostClassifier`**(*base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None*)

sklearn.ensemble.GradientBoostingClassifier

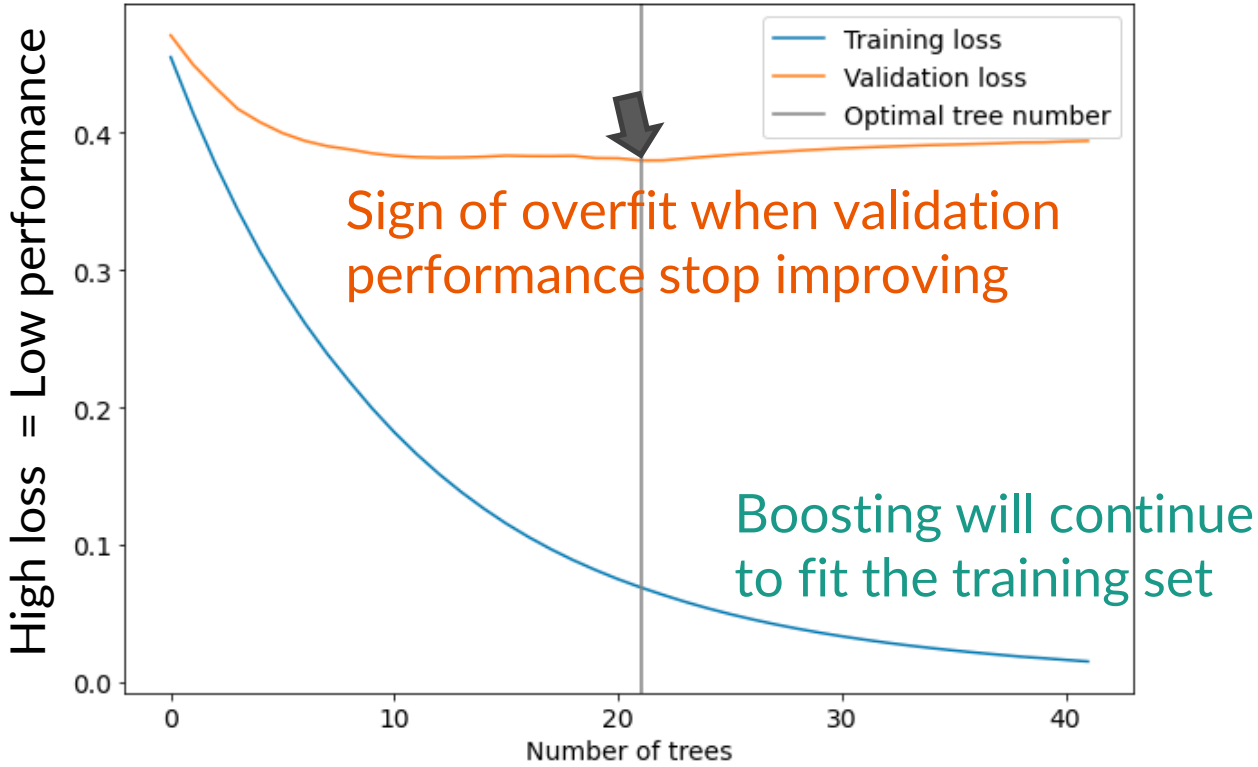
class `sklearn.ensemble.`**`GradientBoostingClassifier`**(**, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, 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*) [source]

Impact of learning rate



- Learning rate = amount of trust on each new model
- Too high = overfitting
- Too low = too many models

Early stopping



Early stopping with validation set

sklearn.ensemble.GradientBoostingClassifier

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*) [source]

- validation_fraction = automatically allocate some fraction of training samples for validation
- n_iter_no_change = number of rounds without improvement in validation performance to terminate the training early

Mixing boosting and bagging

sklearn.ensemble.GradientBoostingClassifier

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) [source]

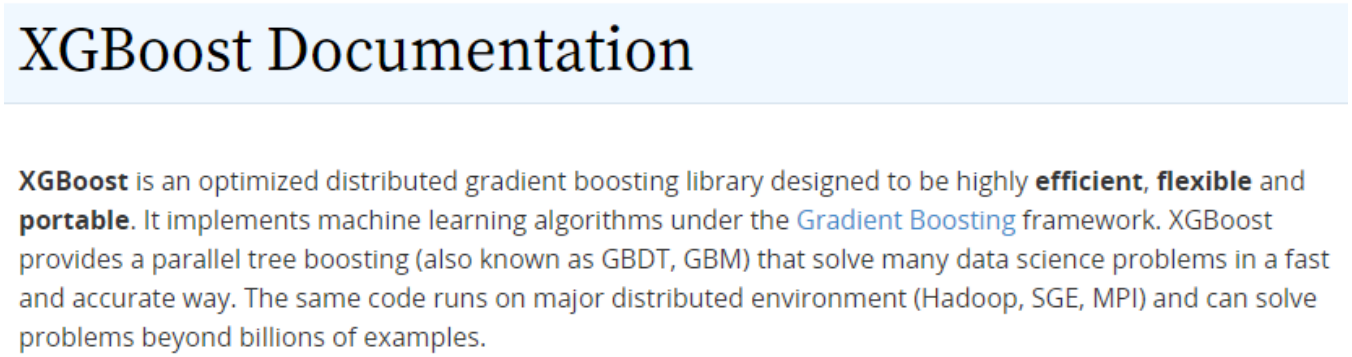
- Conceptually, boosting is supposed to operate on weak models
- All samples and features are considered to get the best fit
- But some sample bagging and feature bagging can still help

Mixing bagging and boosting

params	mean_test_accuracy	std_test_accuracy
{'colsample_bytree': 0.4, 'subsample': 0.9}	0.540834	0.010314
{'colsample_bytree': 0.3, 'subsample': 0.9}	0.538474	0.011719
{'colsample_bytree': 0.7000000000000002, 'subs...	0.538159	0.011733
{'colsample_bytree': 0.7000000000000002, 'subs...	0.537844	0.013849
{'colsample_bytree': 0.4, 'subsample': 1.0}	0.537530	0.008767

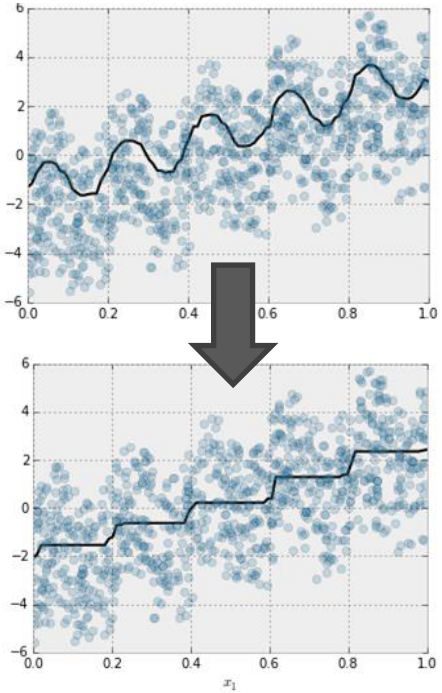
- Small improvement when adding some sample bagging and feature bagging to gradient boosting trees

Boosting in Python (xgboost)



- Full control of training process
  - Monitor multiple loss functions and datasets
- Handle missing data as information
- Support monotonic constraint

Monotonicity constraint



- Monotonicity = if x in creases, then y should increase

- Easy to learn by a linear model: signs of coefficients
- Difficulty to learn by a tree model

- XGBoost can be set to reject branching split that violate the constraint
  - “If values of a feature x is higher in the left branch, then the proportion of positive samples in that branch must also be higher”

Terminology in XGBoost

***class* xgboost.XGBClassifier(*, *objective='binary:logistic'*, *use_label_encoder=True*, ***kwargs*)**

Bases: `xgboost.sklearn.XGBModel`, `object`

Implementation of the scikit-learn API for XGBoost classification.

Parameters

- **n_estimators** (*int*) – Number of boosting rounds.
- **use_label_encoder** (*bool*) – (Deprecated) Use the label encoder from scikit-learn to encode the labels. For new code, we recommend that you set this parameter to False.
- **max_depth** (*Optional[int]*) – Maximum tree depth for base learners.
- **learning_rate** (*Optional[float]*) – Boosting learning rate (xgb's "eta")
- **verbosity** (*Optional[int]*) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
- **objective** (*Union[str, Callable[[numpy.ndarray, numpy.ndarray], Tuple[numpy.ndarray, numpy.ndarray]], NoneType]*) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
- **booster** (*Optional[str]*) – Specify which booster to use: gbtree, gblinear or dart.
- **tree_method** (*Optional[str]*) – Specify which tree method to use. Default to auto. If this parameter is set to default, XGBoost will choose the most conservative option available. It's recommended to study this option from the parameters document: https://xgboost.readthedocs.io/en/latest/treemethod.html.
- **n_jobs** (*Optional[int]*) – Number of parallel threads used to run xgboost. When used with other Scikit-Learn algorithms like grid search, you may choose which algorithm to parallelize and balance the threads. Creating thread contention will significantly slow down both algorithms.
- **gamma** (*Optional[float]*) – Minimum loss reduction required to make a further partition on a leaf node of the tree.

Terminology in XGBoost

scikit-learn	XGBoost	Description
min_samples_split	min_child_weight	Control the amount of evidence to support a split. [0, inf] scale. Higher value = more regularized
min_impurity_split	gamma	Minimum improvement in impurity/loss. [0, inf] scale. Higher value = more regularized
alpha (for L1)	reg_alpha	Regularization strength on L1 term. Default = 0
alpha (for L2)	reg_lambda	Regularization strength on L2 term. Default = 1
max_features	colsample_bytree colsample_bylevel colsample_bynode	Number of random features considered by each tree Number of random features considered at each depth Number of random features considered at each split

More options in XGBoost

Parameter	Description
tree_method	Tree building algorithm (exact, approx, hist, etc.)
monotonic_constraints interaction_constraints	Apply monotonic constrain (-1, 0, 1) for each feature Enforce interaction constraints (allowed feature combinations)
eval_metric	Scoring metrics used for tracking the training and early stopping
early_stopping_rounds	Number of rounds without improvement in validation performance to terminate the training early
callbacks	More advanced functions to perform at the end of each epoch We will learn more about these in the deep learning sections
objective	Define the training objective/loss function

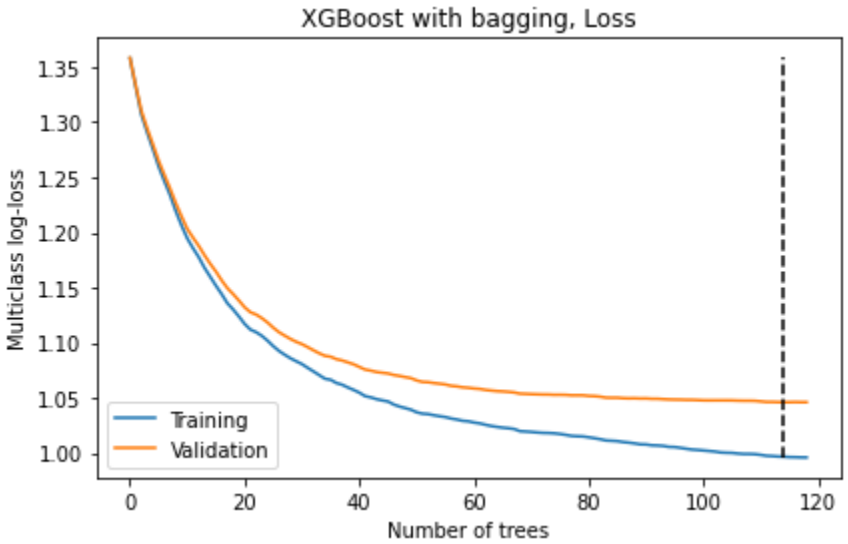
Example of xgboost setup

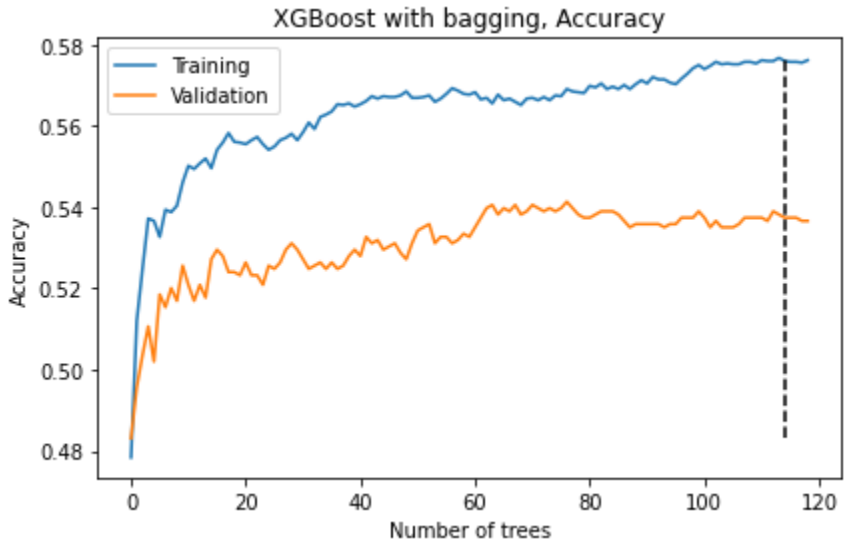
```
import xgboost as xgb

tuned_xgb_fast = xgb.XGBClassifier(n_estimators = 2000, use_label_encoder = False,
                                   tree_method = 'hist', max_depth = 5, gamma = 2,
                                   min_child_weight = 0.7, colsample_bytree = 0.4,
                                   subsample = 0.9, learning_rate = 0.5)

tuned_xgb_fast.fit(X_train, y_train,
                   eval_set = [(X_train, y_train), (X_val, y_val)],
                   eval_metric = ['merror', 'mlogloss'],
                   early_stopping_rounds = 5, verbose = 0)
```

Loss curve information in xgboost






```
train_loss = xgb_model.evals_result_['validation_0']['mlogloss']
val_loss = xgb_model.evals_result_['validation_1']['mlogloss']
stopped_n_tree = xgb_model.best_iteration
```

Any question?