Special Topics: ENSEMBLES

PHYS 453

Dr Daugherity

Ensembles

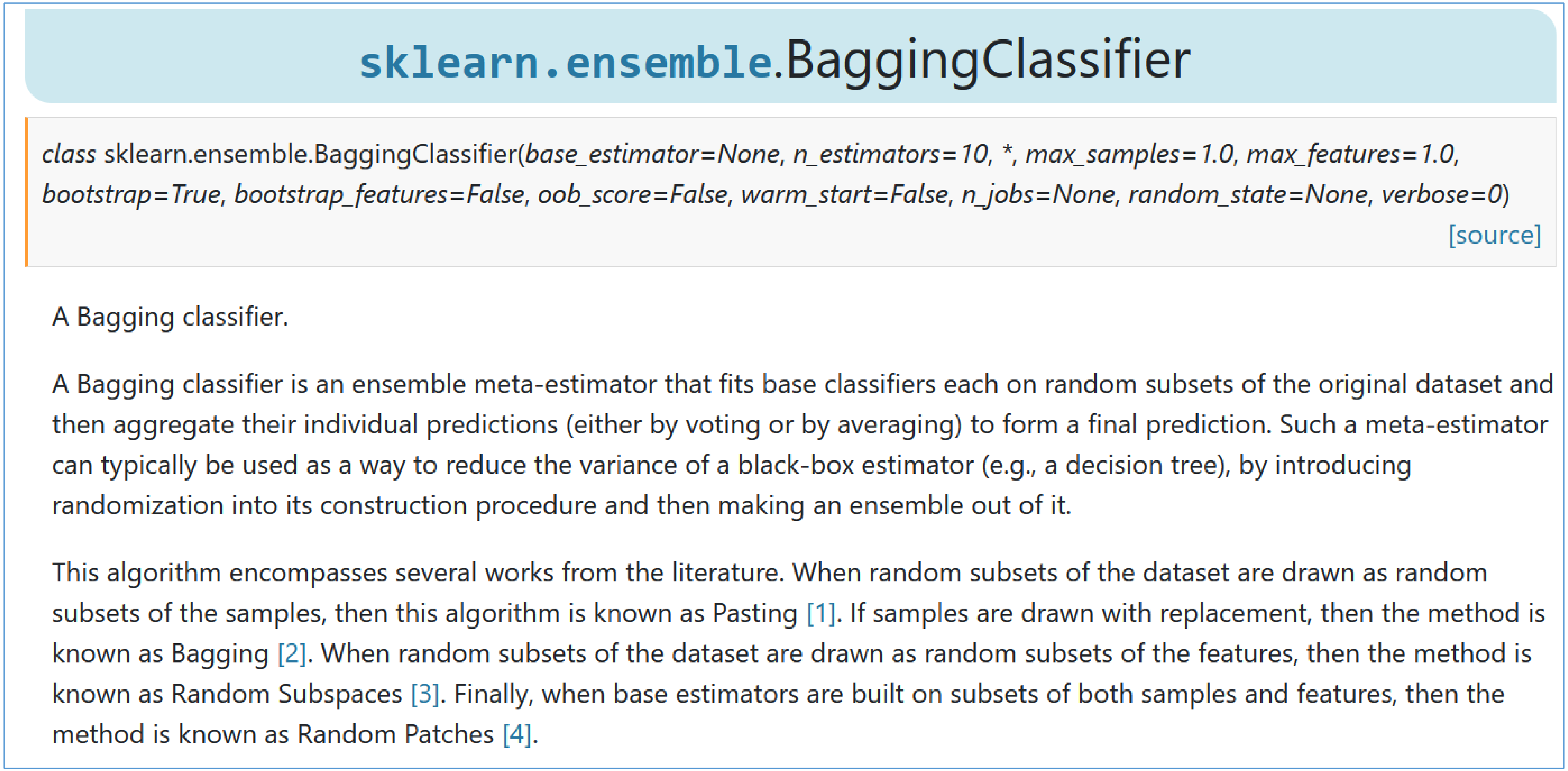
Ensemble Methods

- Basically a series of sneaky tricks to improve performance using multiple classifiers/regressors
- Several different approaches:
  - Reward classifiers that perform well
  - Increase weight of important training data
  - Specifically train new classifiers to help special cases
- Generally combining several models improves performance, reduces variance, and limits overfitting
- Possible huge bonus running in parallel

Bagging

- Bagging = Bootstrap Aggregation
- Train multiple classifiers on different random samples of training data
- Each classifier can run in parallel!
- Decision Trees are a common choice

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



sklearn.ensemble.BaggingClassifier

class sklearn.ensemble.BaggingClassifier(*base_estimator=None*, *n_estimators=10*, ***, *max_samples=1.0*, *max_features=1.0*, *bootstrap=True*, *bootstrap_features=False*, *oob_score=False*, *warm_start=False*, *n_jobs=None*, *random_state=None*, *verbose=0*) [source]

A Bagging classifier.

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

This algorithm encompasses several works from the literature. When random subsets of the dataset are drawn as random subsets of the samples, then this algorithm is known as Pasting [1]. If samples are drawn with replacement, then the method is known as Bagging [2]. When random subsets of the dataset are drawn as random subsets of the features, then the method is known as Random Subspaces [3]. Finally, when base estimators are built on subsets of both samples and features, then the method is known as Random Patches [4].

Boosting

- Train models sequentially learning from previous mistakes!
- Examples:
  - Adaboost
  - Gradient Tree Boosting
  - XGBoost

AdaBoost

Adaptive Boosting:

1. Train basic (“weak”) classifier
2. Increase weight of misclassified training samples
3. Train new classifier on weighted samples
4. Repeat. Final is average of all classifiers.

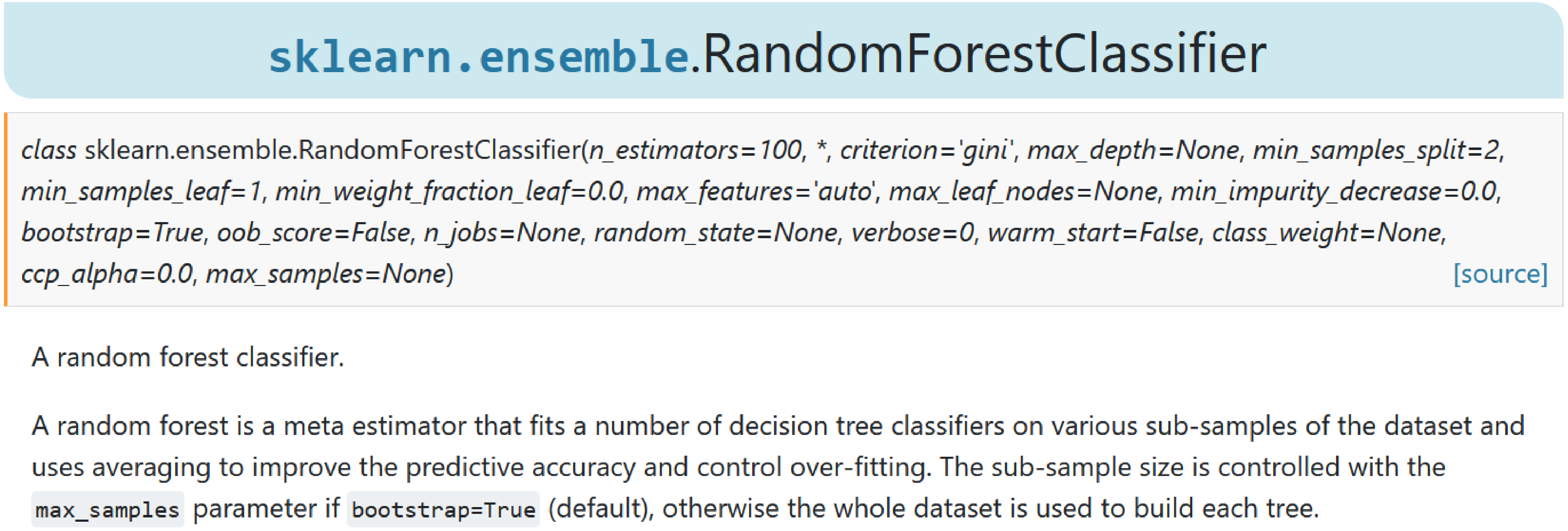
Boosting

Algorithm 11.3, p.335

Algorithm Boosting($D, T, \mathscr{A}$) – train an ensemble of binary classifiers from reweighted training sets.

```
Input  : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.
Output : weighted ensemble of models.
1  $w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$ ;                                   // start with uniform weights
2  for $t = 1$ to $T$ do
3      run $\mathscr{A}$ on $D$ with weights $w_{ti}$ to produce a model $M_t$;
4      calculate weighted error $\epsilon_t$;
5      if $\epsilon_t \geq 1/2$ then
6          set $T \leftarrow t - 1$ and break
7      end
8      $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ ;                                   // confidence for this model
9      $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$ ;             // increase weight
10     $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$ ;   // decrease
11 end
12 return $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$
```

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



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='auto', 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]

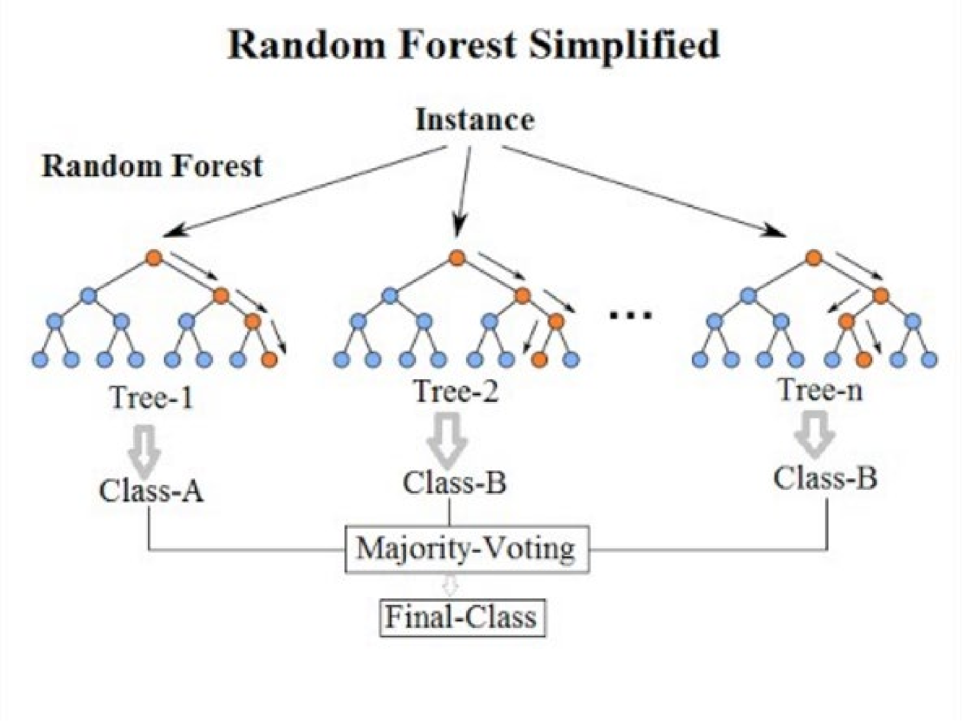
A random forest classifier.

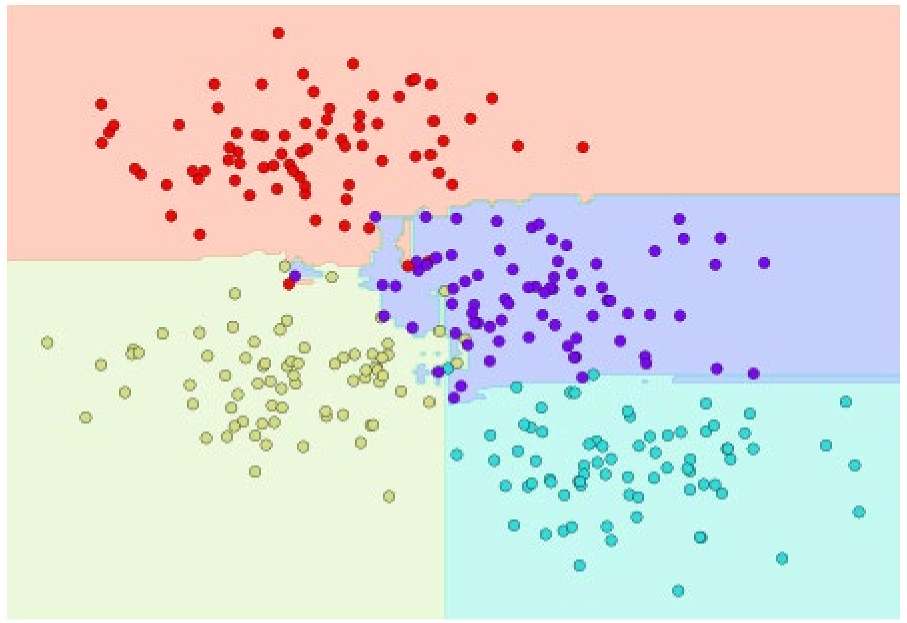
A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is controlled with the `max_samples` parameter if `bootstrap=True` (default), otherwise the whole dataset is used to build each tree.

Random Forest

One common approach to minimize problems is to create a random forest:

- train lots of trees on different random samples of data
- each tree splits on random feature (instead of minimum impurity)
- the whole forest votes on classification





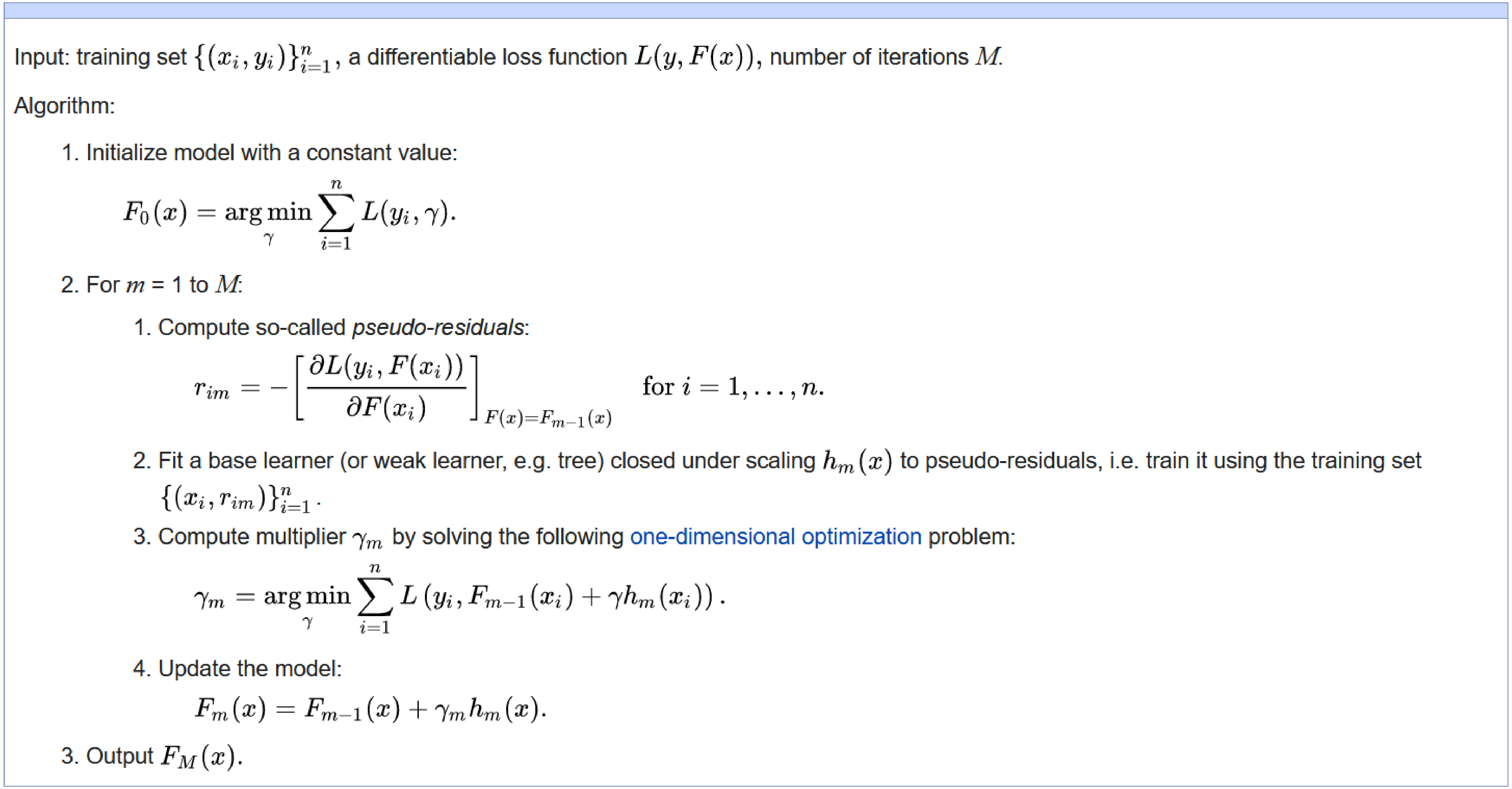
http://nbviewer.jupyter.org/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb

Gradient Tree Boosting (GBDT)

Gradient Tree Boosting:

1. Train basic (“weak”) model
2. Calculate the error
3. Train a new model to the only the error of previous model(s)
4. Combined model is weighted sum of all previous.
5. Repeat.

https://en.wikipedia.org/wiki/Gradient_boosting



Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

   $$F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For *m* = 1 to M:
   1. Compute so-called *pseudo-residuals*:

      $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

   2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
   3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

      $$\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

   4. Update the model:

      $$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

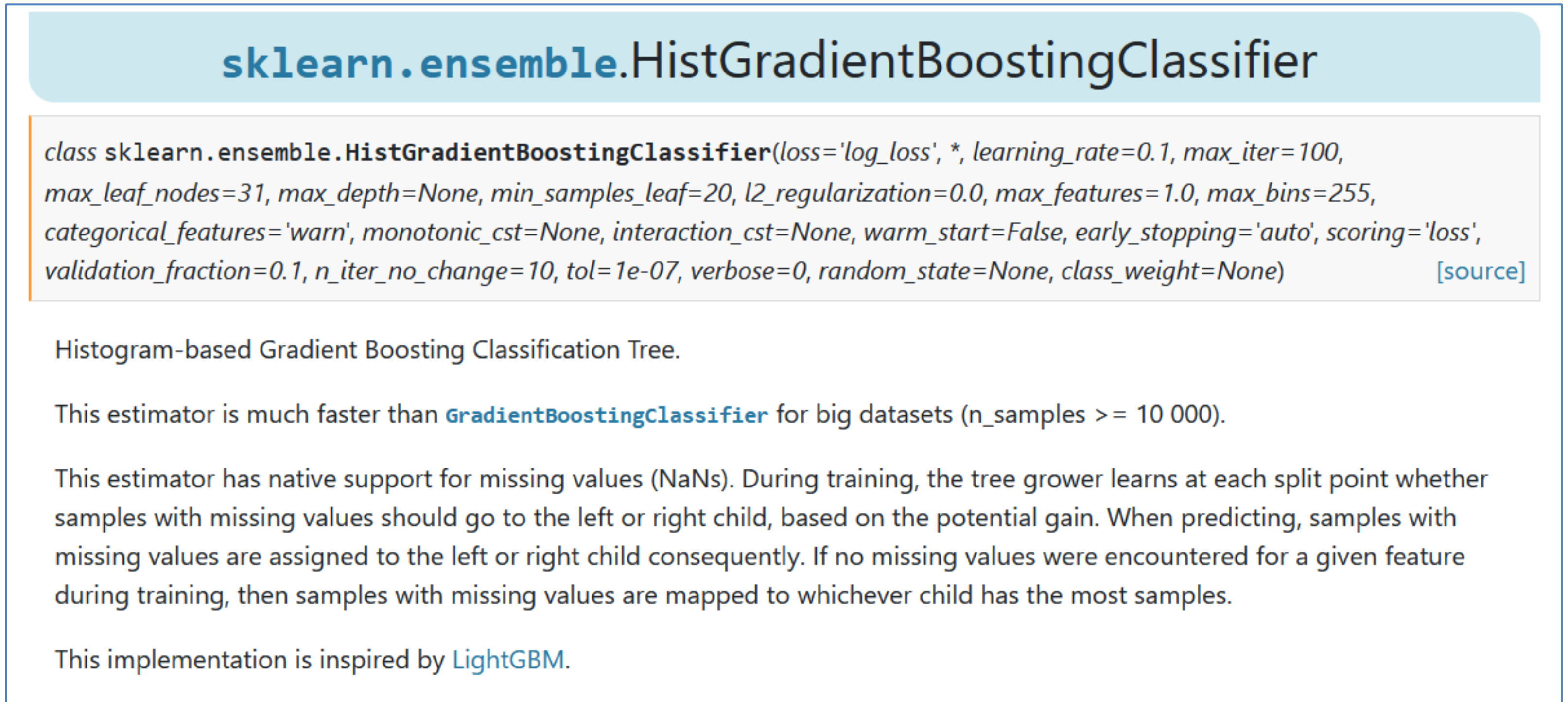
3. Output $F_M(x)$.

Usage

sklearn has 2 implementations:

1) GradientBoostingClassifier – slightly more accurate, much slower
2) HistGradientBoostingClassifier – slightly less accurate, much faster

Use the fast one.



sklearn.ensemble.HistGradientBoostingClassifier

class `sklearn.ensemble.HistGradientBoostingClassifier`(*loss='log_loss', *, learning_rate=0.1, max_iter=100, max_leaf_nodes=31, max_depth=None, min_samples_leaf=20, l2_regularization=0.0, max_features=1.0, max_bins=255, categorical_features='warn', monotonic_cst=None, interaction_cst=None, warm_start=False, early_stopping='auto', scoring='loss', validation_fraction=0.1, n_iter_no_change=10, tol=1e-07, verbose=0, random_state=None, class_weight=None*) [source]

Histogram-based Gradient Boosting Classification Tree.

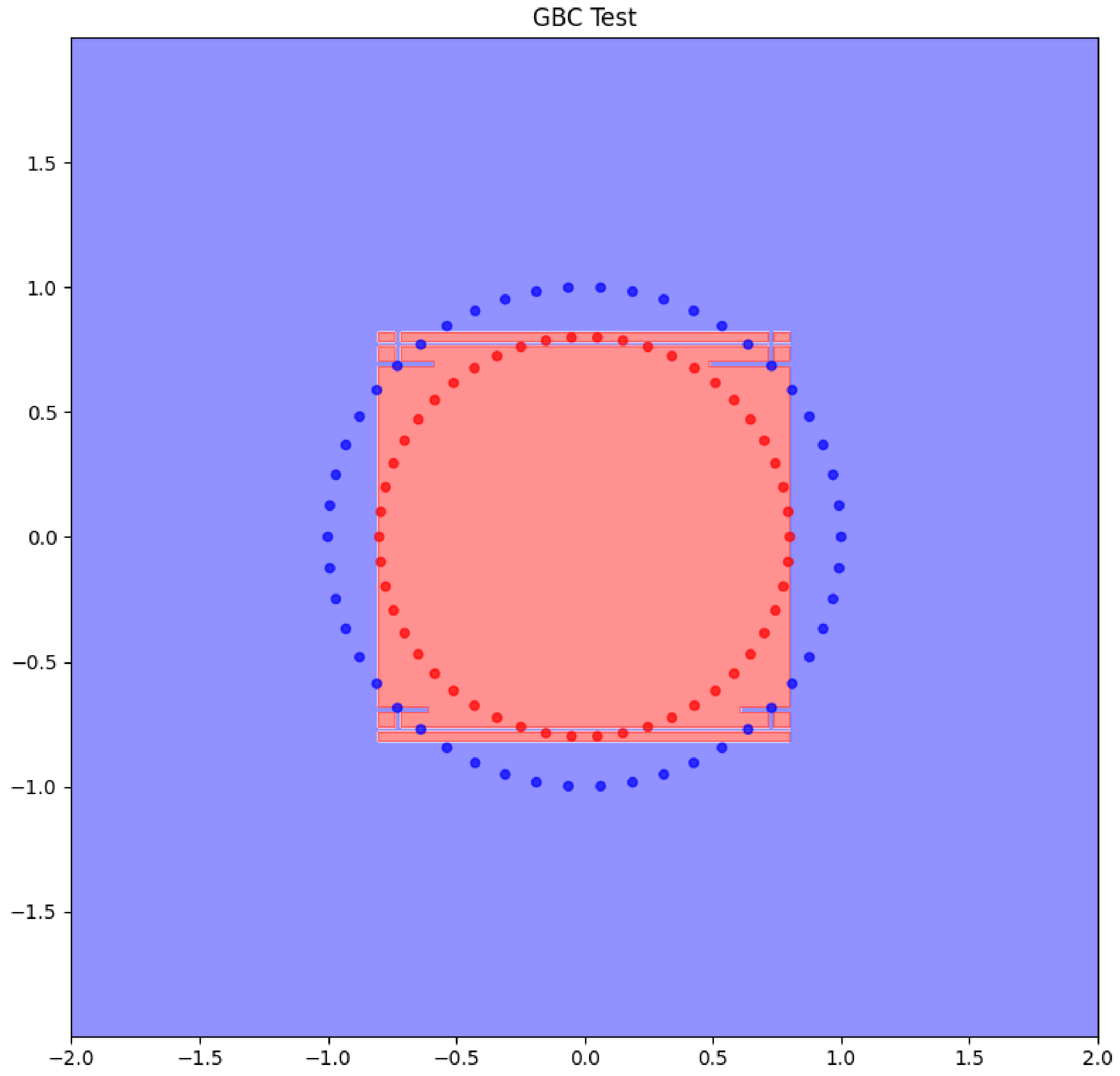
This estimator is much faster than `GradientBoostingClassifier` for big datasets (n_samples >= 10 000).

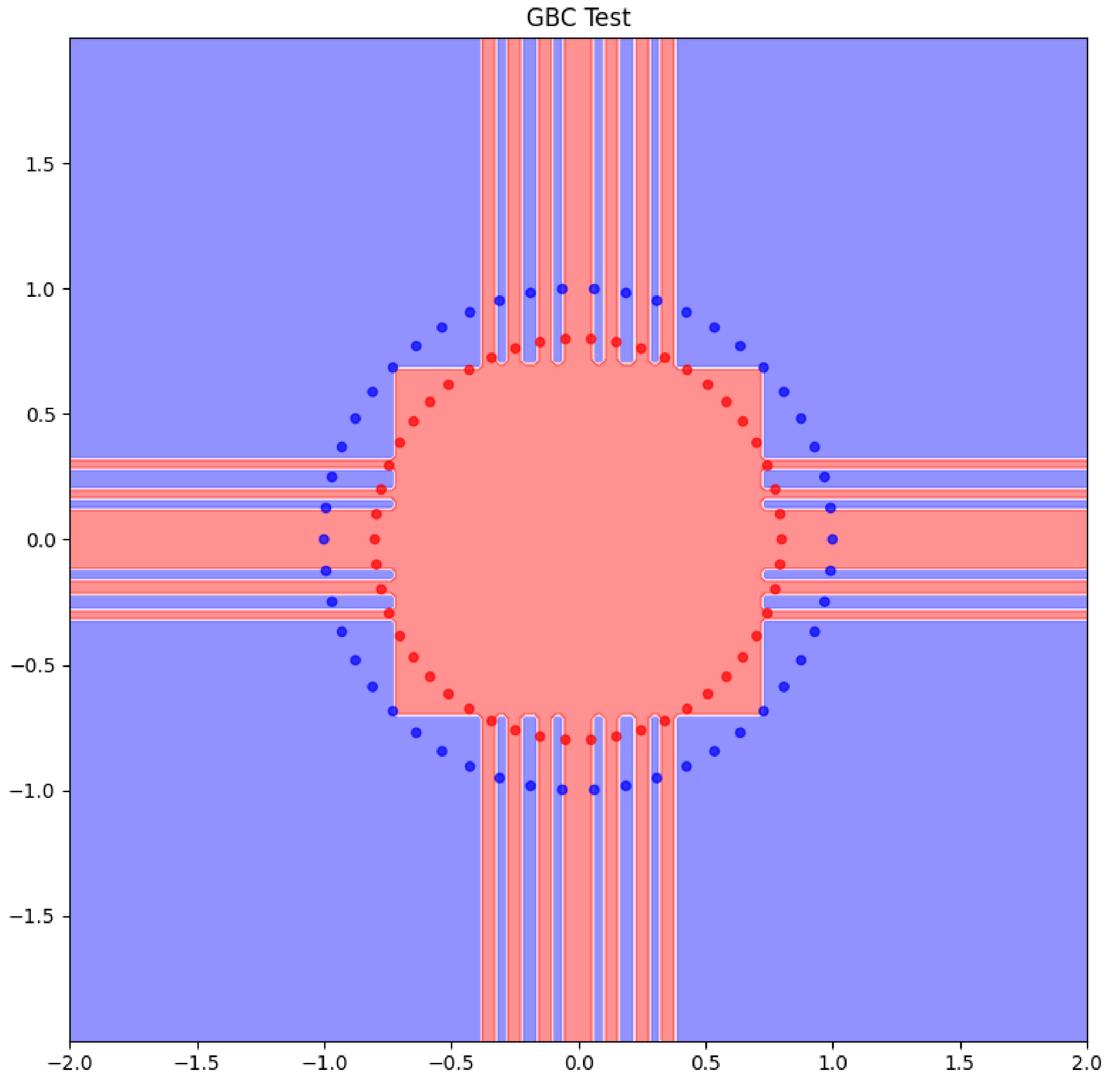
This estimator has native support for missing values (NaNs). During training, the tree grower learns at each split point whether samples with missing values should go to the left or right child, based on the potential gain. When predicting, samples with missing values are assigned to the left or right child consequently. If no missing values were encountered for a given feature during training, then samples with missing values are mapped to whichever child has the most samples.

This implementation is inspired by LightGBM.

NON-HIST

HIST





GBC Usage

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

Parameters to know:

- n_estimators = 100, the main parameter. Best to set by early stopping.
- learning_rate = 0.1, effect of each new learner. Default works well.

Early stopping:

- n_iter_no_change=None, stop if val score doesn’t change within tolerance
- validation_fraction=0.1, size of validation set
- tol=1e-4, early stopping tolerance

HistGBC Usage

HistGBC API: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html

Parameters to know:

- max_iter=100, the main parameter. Best to set by early stopping.
- learning_rate = 0.1, effect of each new learner. Default works well.

Early stopping:

- early_stopping='auto', auto early stops if N>10k
- n_iter_no_change=10, stop if val score doesn't change within tolerance
- validation_fraction=0.1, size of validation set
- tol=1e-7, early stopping tolerance

Training Tips

Easiest method for good results:

- early_stopping=True
- learning_rate=0.1
- set max_iter high enough so that it stops early:
  - can set verbose=1
  - after fitting check n_iter_
  - plot train_score_ and validation_score (adjust n_iter_no_change or tol)

Works well without a grid search!

EXTREME Gradient Boost (XGBoost)

XGBoost: advanced implementation of gradient boosted trees with built-in parallelization, regularization, and other bonuses. Famous for winning the Higgs Boson challenge in 2015.

Same general concept as Gradient Trees but with extra features and support for high-end computing (like running in parallel on GPUs)

XGBoost is considered by many as the best, state-of-the-art, gold standard classifier/regressor to use!

XGBoost

- Not officially part of sklearn, but has a wrapper for easy use
- https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn
- https://www.kaggle.com/code/stuarthallows/using-xgboost-with-scikit-learn/notebook
- https://towardsdatascience.com/a-visual-guide-to-gradient-boosted-trees-8d9ed578b33
  - https://colab.research.google.com/drive/1tRzL1GGOJDgz7CHSY1HECd4IwNAjuip5?usp=sharing