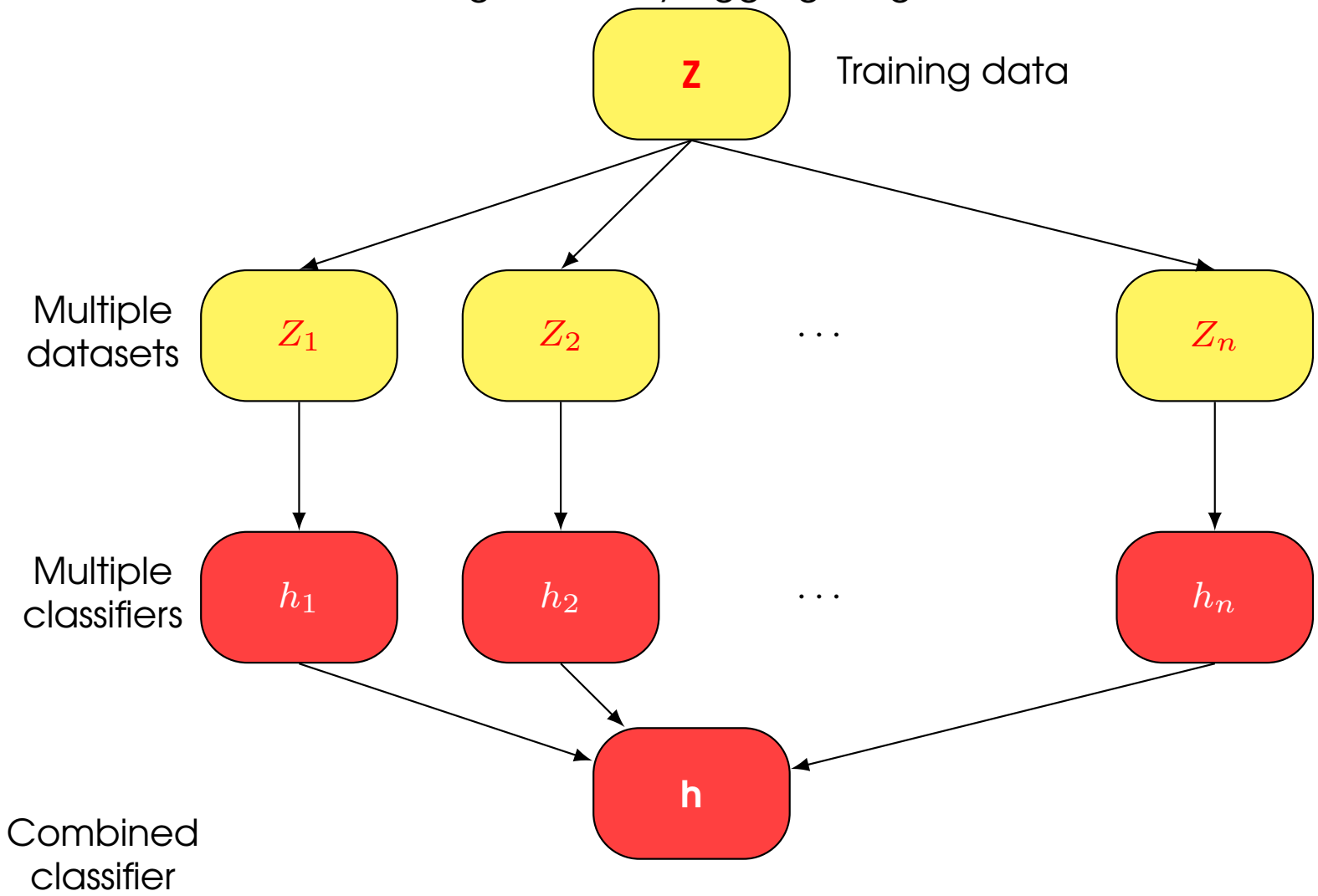
Ensemble methods

Prediction (classification / regression) by aggregating a set of weak learners.



Idea: Build different experts, and let them vote.

Advantages

- Improve predictive performance
- Other types of classifiers can be directly included
- Easy to implement
- No too much parameter tuning

Drawbacks

- The resulting classifier h is not so transparent (black box)
- Not a compact representation

Example

25 base classifiers $h_i, 1 \leq i \leq 25$

- Each classifier h_i has error rate $\epsilon = 0.35$
- independence among classifiers

P: Probability that the ensemble classifier h makes a wrong prediction:

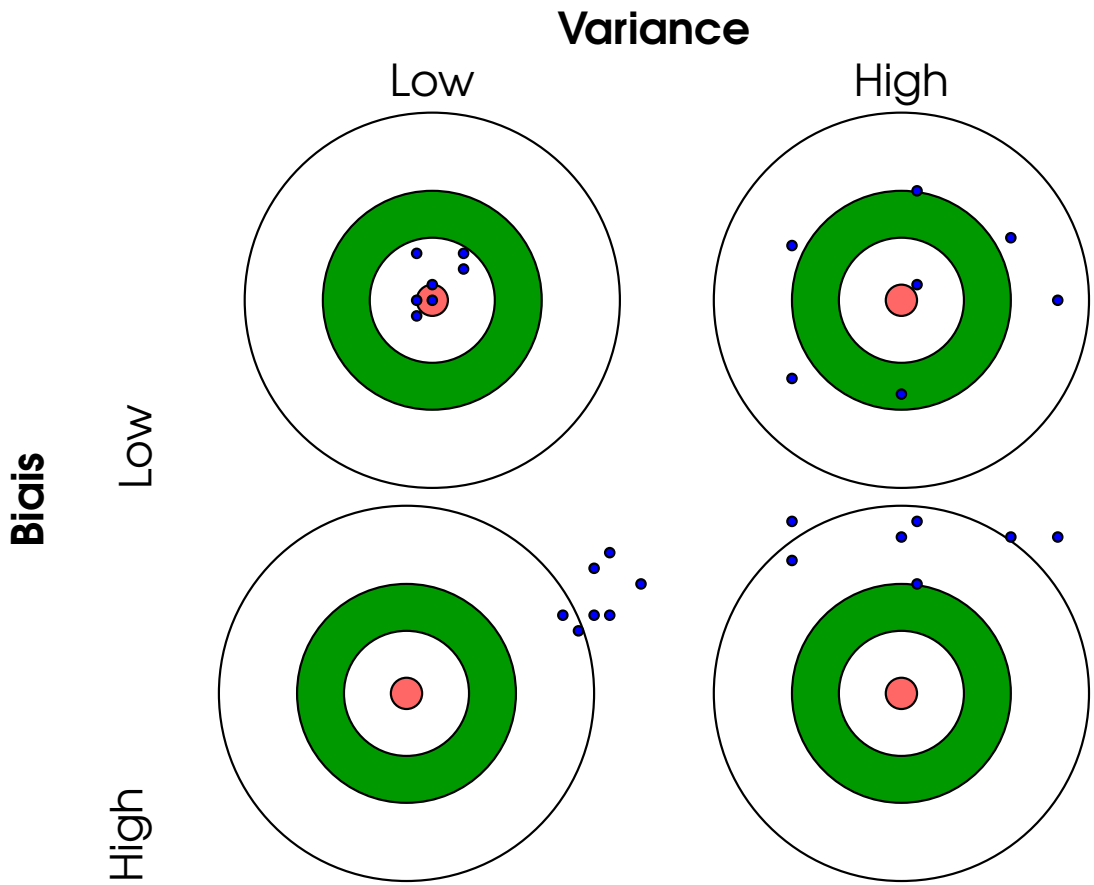
$$P = \sum_{i=1}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

In the following...

Classifiers/regressors will be decision trees, but can be any other algorithm

Bias-Variance trade-off

Quality of predictive models are evaluated by their bias-variance properties.



Bias-Variance trade-off

Highly challenging to design the perfect model f_Z (i.e. low bias and low variance)

$$\mathbb{E}_{(x,y)}\ell(f_Z(x),y) = \mathbb{E}_{(x,y)} \underbrace{\ell(f_Z(x),\bar{f}(x))}_{\substack{\text{error between}\\ \text{model and average}\\ \text{over all predictions}\\ \text{VARIANCE}}} + \underbrace{\ell(\bar{f}(x),y)}_{\substack{\text{error between}\\ \text{average predictor}\\ \text{and target}\\ \text{BIAS}}}$$

with

- ℓ: loss function
- Z: training set on which f is trained
- Z^*: true data distribution
- $f_Z(x)$: predictive value on x
- y : target value
- $\bar{f}$: average predictor, $\bar{f}(x) = \int_{Z' \subset Z^*} f_{Z'}(x)p(Z')dZ'$

Reducing Variance

Minimizing the variance

- Decision trees have low bias but high variance.
- Find a way to reduce variance: $MinE_{(x,y)}\ell(f_Z(x), \bar{f}(x))$
- Idea: take the average of multiple solutions $\rightarrow$ Ensemble methods

$$f_Z(x) = \frac{1}{m}\sum_{i=1}^{M} f_{Z_j}(x) \underset{M\rightarrow\infty}{\longrightarrow} \bar{f}(x)$$

Why ?

x_i random variables iid with mean $\bar{x}$

Law of large numbers: $\lim_{M\rightarrow\infty} \frac{1}{m}\sum_{i=1}^{M} x_i = \bar{x}$

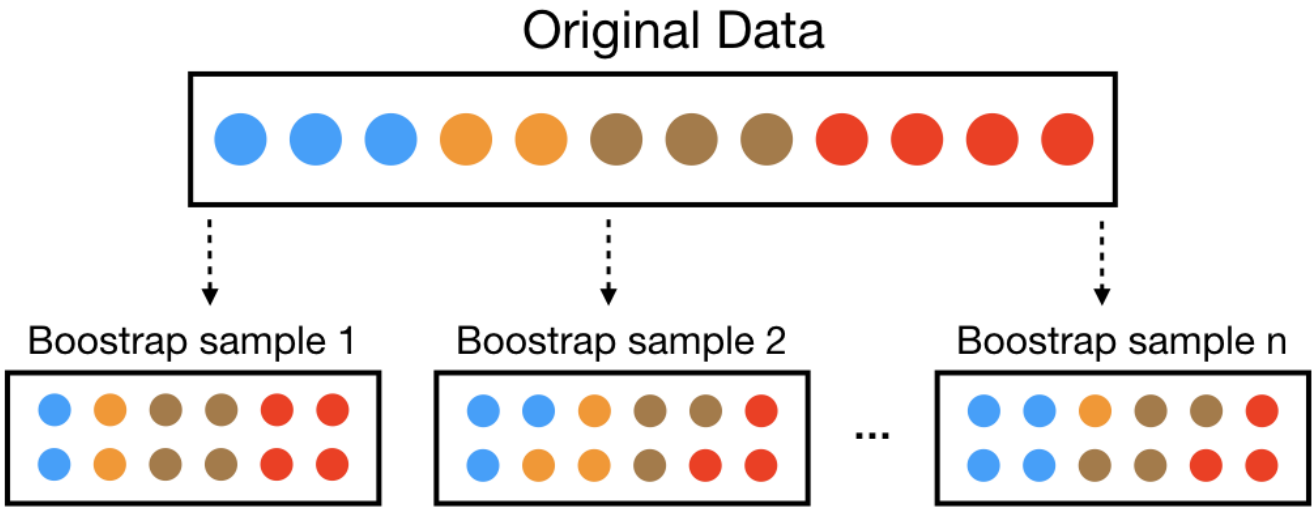
How to choose the Z_i's ?

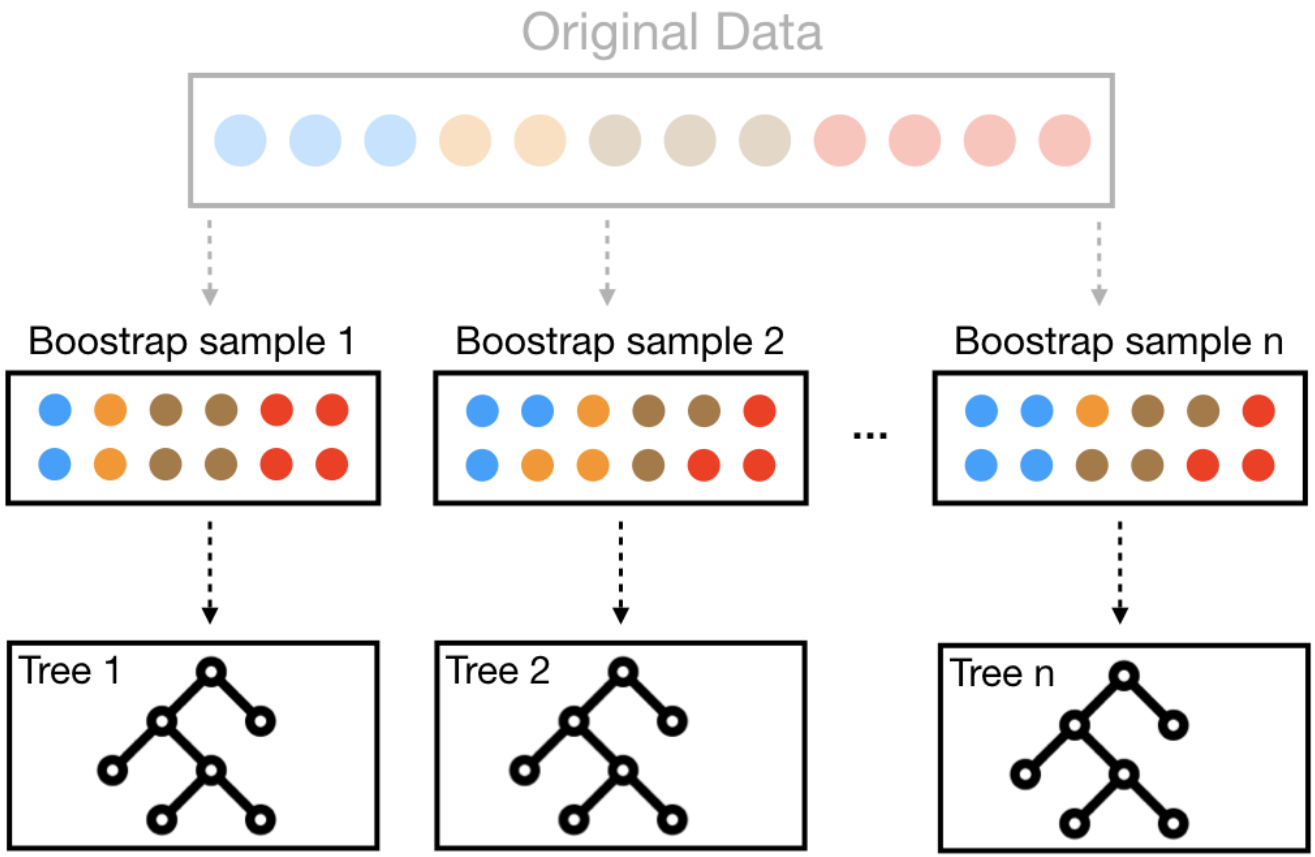
1. Sampling Z with replacement: Bagging
2. Sampling without replacement and with strategies: Boosting

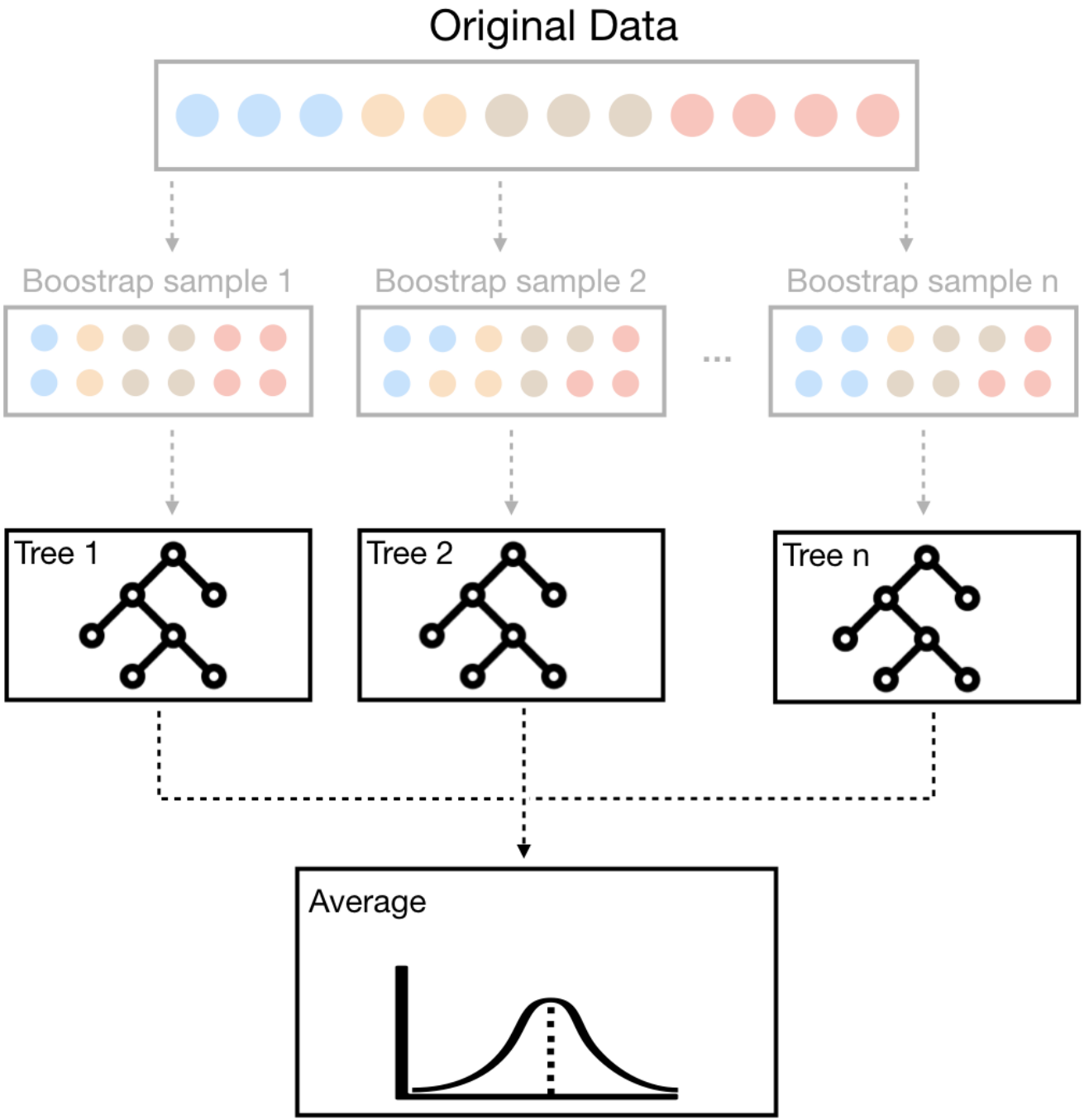
Bagging

Bagging = Bootstrap Aggregation

1. Learning stage
   - Given a dataset Z, at each iteration i, a training set Z_i is sampled with replacement (bootstrap) from Z, $|Z_i| \leq |Z|$
   - A classifier h_i is learned for each Z_i
2. Classification stage on x
   - each h_i returns its prediction
   - The bagged classifier h votes and assigned the class with the most votes to x
3. Regression stage on x
   - each h_i returns its prediction
   - The bagged classifier h votes and assigned the mean value to x


Original Data
Boostrap sample 1
Boostrap sample 2
...
Boostrap sample n


Original Data
Boostrap sample 1
Boostrap sample 2
...
Boostrap sample n
Tree 1
Tree 2
Tree n


Original Data
Boostrap sample 1
Boostrap sample 2
Boostrap sample n
...
Tree 1
Tree 2
Tree n
Average

Bagging

Why does bagging work?

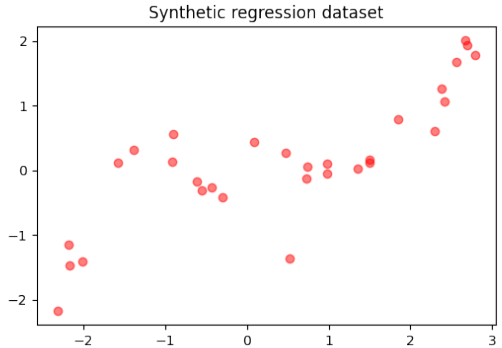
- Break the assumption of the law of large numbers (Z_i data not iid)
- But ...it almost always reduces variance by voting/averaging
- Reduce variance without increasing the error of an unbiased model
- Does not focus on any particular instance of the training data
  $\Rightarrow$ less susceptible to model overfitting when applied to noisy data
- Usually, the more classifiers the better

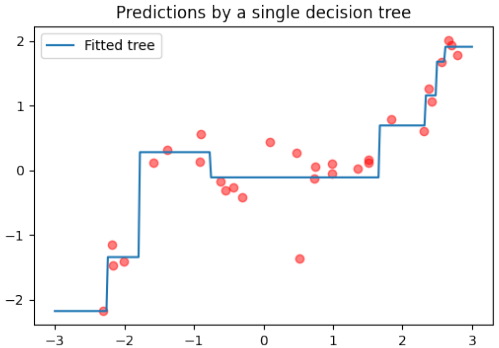
Out-of-Bag

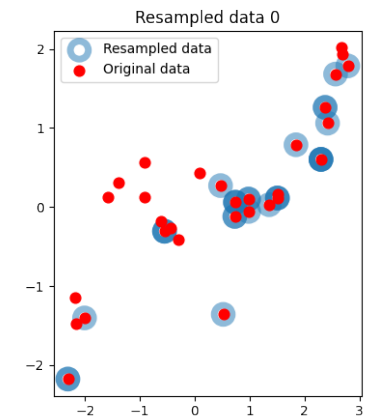
- If n is large, after n samples have been drawn, the probability that a sample has not been drawn yet is $\left(1 - \frac{1}{n}\right)^n \approx e^{-1}$
- Each Z_i contains $(1 - e^{-1}) \approx 63.2\%$ of the samples
- $\approx 36.8\%$ of the samples can be used for efficient assessment of model performance (Out-of-bag evaluation)

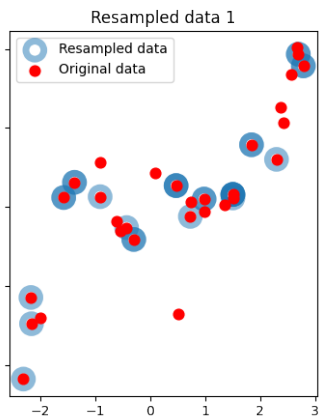
```
from sklearn.svm import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
bagging = BaggingClassifier(base_estimator=DecisionTreeClassifier(),n_estimators=10, random_state=0)
bagging.fit(X, y)
```

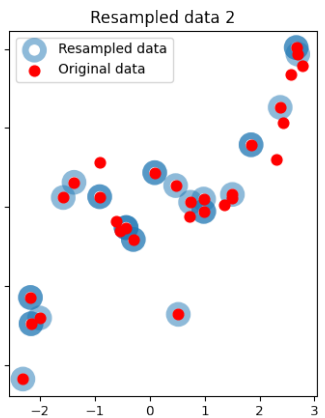
Regression example

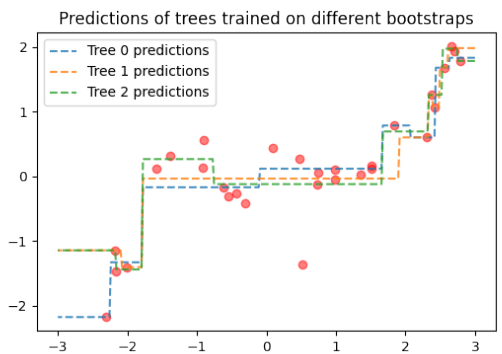


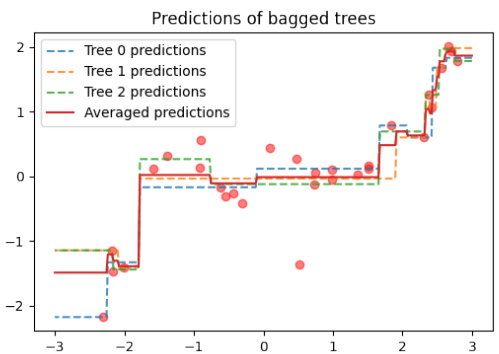




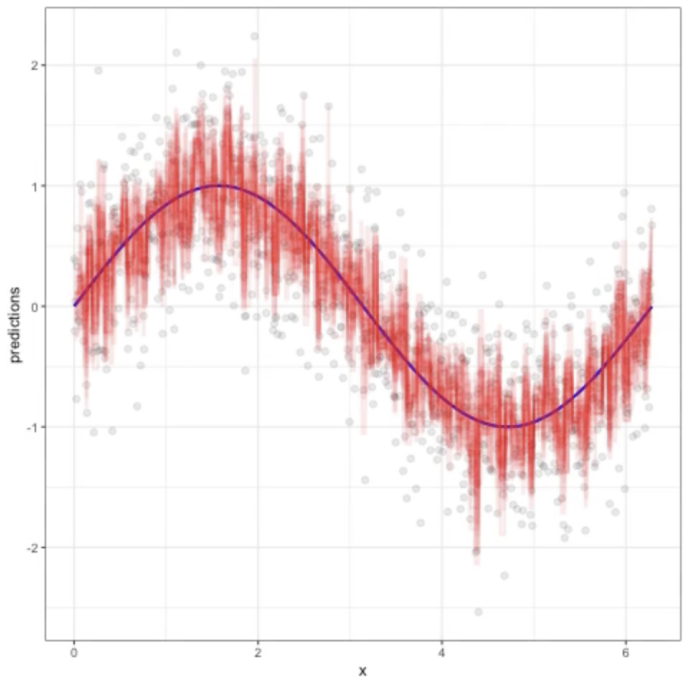




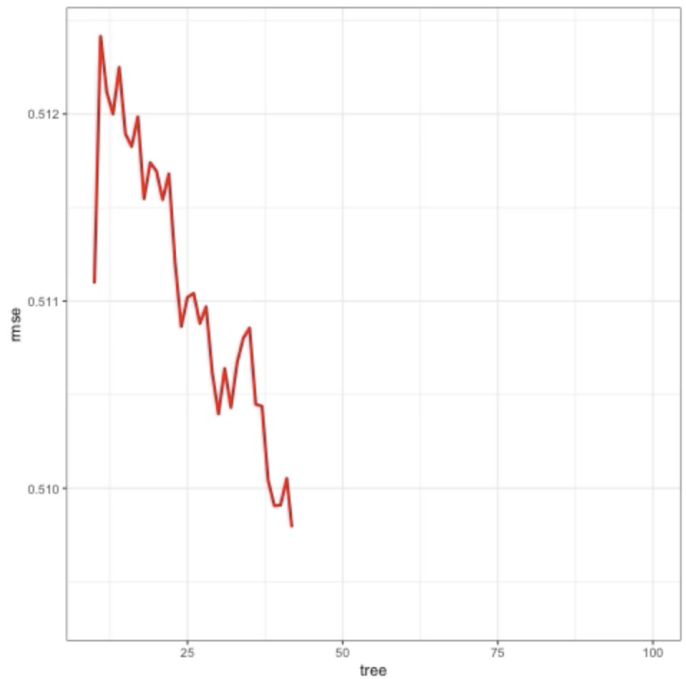




Add trees...



Adding trees



Average prediction error decreases

But...

Bagging results in tree correlation
$\Rightarrow$ Prevents from optimally reduce variance of the predictive values.

Definition

Bagging to much results in tree correlations $\Rightarrow$ Find a way to bag unique trees.

Definition

- Combination of tree predictors
- Each tree depends on the values of a random vector sampled in dependently
- The generalization error depends on the strength of the individual trees and the correlation between them
- Using a random selection of features yields results favorable to AdaBoost, and are more robust w.r.t. noise

Algorithm 1: Random forest classifier

Data: $Z = \{(x_i, y_i), 1 \leq i \leq n, y_i \in \{-1, 1\}\}, x \in X, |X| = d, M$ nb of weak classifiers, k
Result: h: strong classifier
for $i \leftarrow 1$ **to** k **do**
  Build subset Z_i by sampling with replacement from Z
  Learn tree h_i from Z_i:
  for *Each node* **do**
    randomly select $p \leq d$ features and choose best of these p
  Each tree grows to the largest extend, no pruning
  Make predictions according to majority vote of the set of k trees

Advantages

Advantages

- Runs efficiently on large data bases.
- can handle thousands of input variables without variable deletion.
- Gives estimates of what variables are important in the classification.
- Generates an internal unbiased estimate of the generalization error as the forest building progresses.
- Possible estimation of missing data
- Handle unbalanced datasets

Parameters

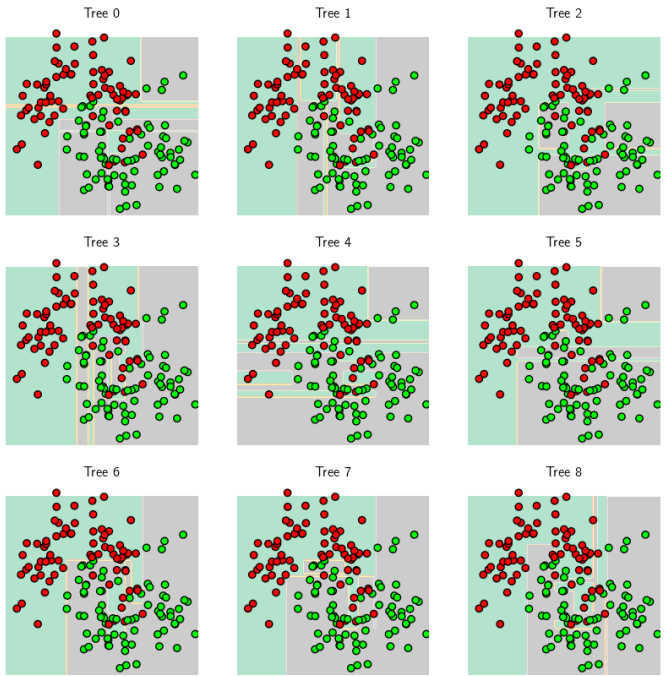
Rules of thumb

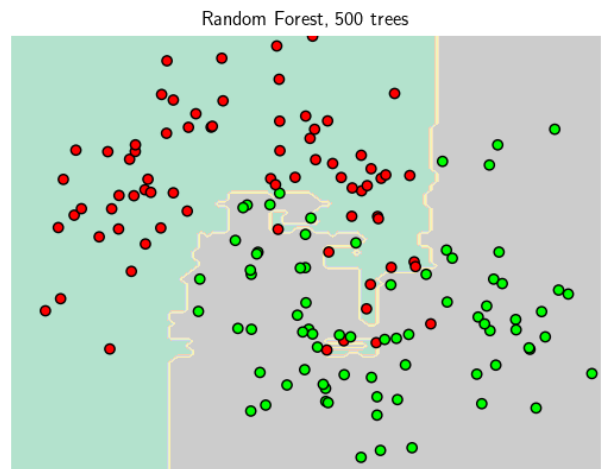
- Number of trees: start with $10d$ and adjust
- d: Regression trees: $p = d/3$; Classification trees $p = \sqrt{d}$
- Node size: 5 (regression), 1 (classification)
- Split rule: variance (regression), Gini/cross entropy (classification)

```
from sklearn.ensemble import RandomForestClassifier
X, y = ...
rf = RandomForestClassifier(max_depth=2, random_state=0)
rf.fit(X, y)
```

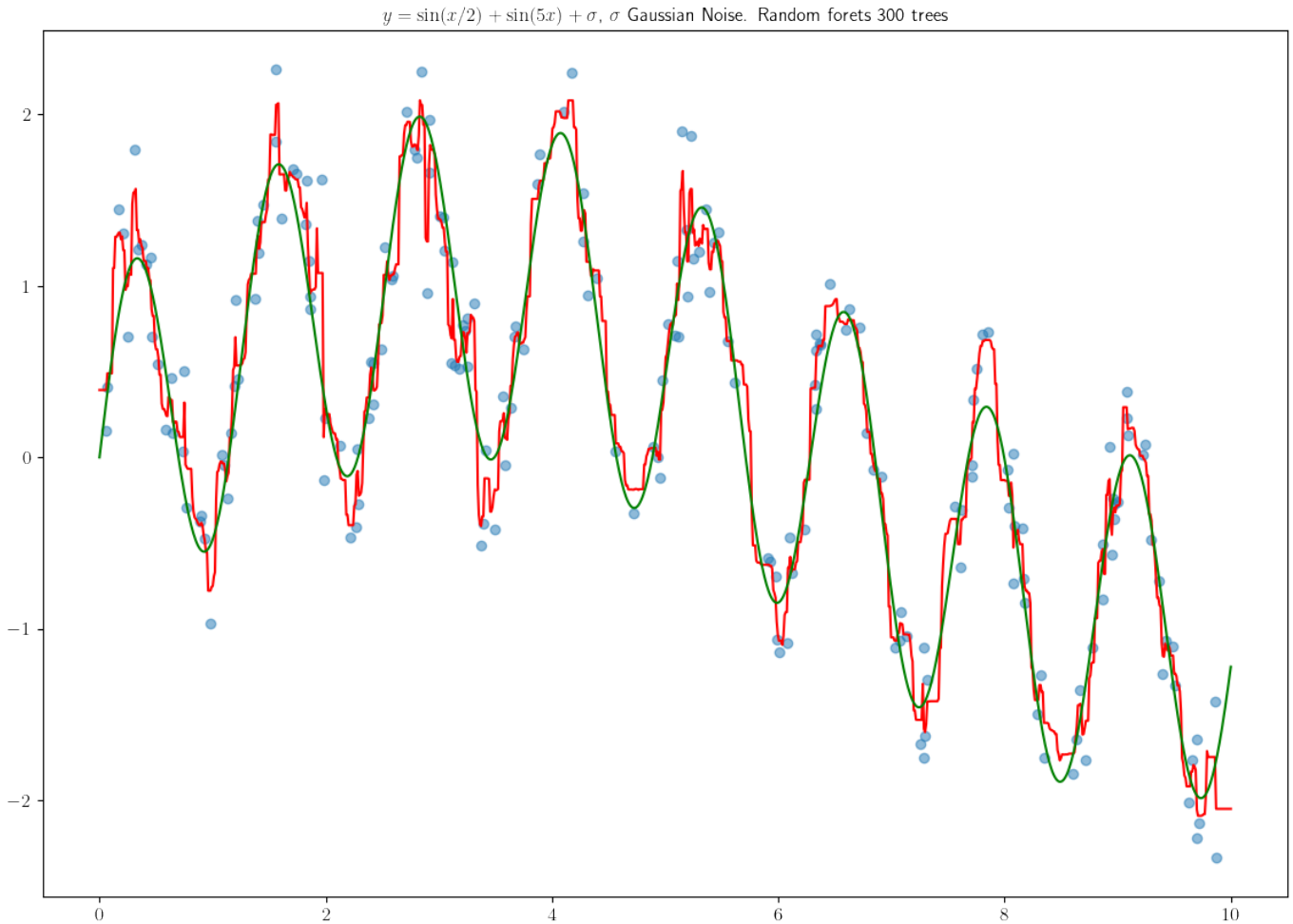
Classification

Two-moons binary classification, 500 trees.





Regression


$y = \sin(x/2) + \sin(5x) + \sigma$, σ Gaussian Noise. Random forets 300 trees
2
1
0
−1
−2
0
2
4
6
8
10

Boosting

Context: high bias models (e.g decision trees with limited depth)

Question

Can we design an ensemble method that combines a large number of weak learners to lower the bias ?

Principle

- Meta-algorithm for reducing bias
- Family of machine learning algorithms which convert weak classifiers to a strong one
- Pays higher focus on examples which are misclassified or have higher errors by preceding weak classifiers.

FIRST ALGORITHM

Algorithm 2: Shapire's algorithm

Data:
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}, x \in X$
Result: h: strong classifier

1. Z_1 : subset of $n_1 < n$ samples of Z randomly drawn without replacement
2. Learn a weak classifier h_1 on Z_1
3. Z_2 : subset of $n_2 < n$ samples of Z, half of which are badly classified by h_1
4. Learn a weak classifier h_2 on Z_2
5. Z_3 : Set of samples on which h_1 and h_2 disagree
6. Learn a weak classifier h_3 on Z_3

$$h(x) = sign\left(\sum_{i=1}^{3} h_i(x)\right)$$

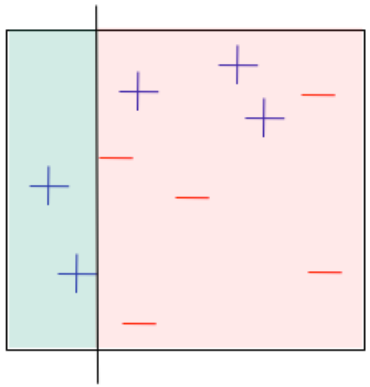
Algorithm 3: Adaboost

Data:
$Z = \{(x_i, y_i), 1 \leq i \leq n, x_i \in X, y_i \in \{-1, 1\}\}, x \in X$, M nb of weak classifiers
Result: h: strong classifier
Weight initialization w : $\forall i \in \{1 \cdots n\}, (w_i = \frac{1}{n})$

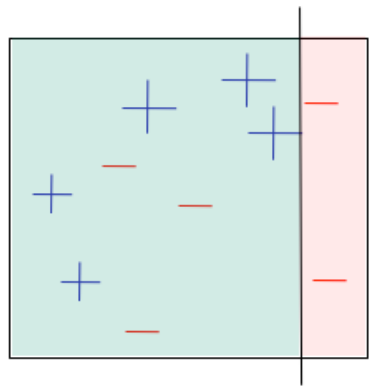
for $i \leftarrow 1$ **to** M **do**
- Compute h_i on Z weighted by w
- Compute the error $\epsilon_i = \sum_{j=1}^{n} \mathbb{I}_{h_i(x_j) \neq y_j}$
- Compute the weight of the weak classifier $\alpha_i \leftarrow \frac{1}{2} log\left(\frac{1-\epsilon_i}{\epsilon_i}\right)$
- **for** $j \leftarrow 1$ **to** n **do**
  - $w_j \leftarrow w_j exp\left[-\alpha_i y_j h_i(x_j)\right]$
- Weight normalization: $W = \sum_{j=1}^{n} w_j$
- **for** $j \leftarrow 1$ **to** n **do**
  - $w_j \leftarrow w_j / W$

$$h(x) = sign\left[\sum_{j=1}^{M} \alpha_j h_j(x)\right]$$

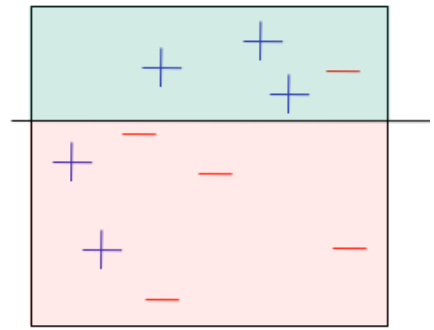
$M = 3$, classify blue crosses / red lines.



(a) $h_1, \epsilon_1 = 0.3$

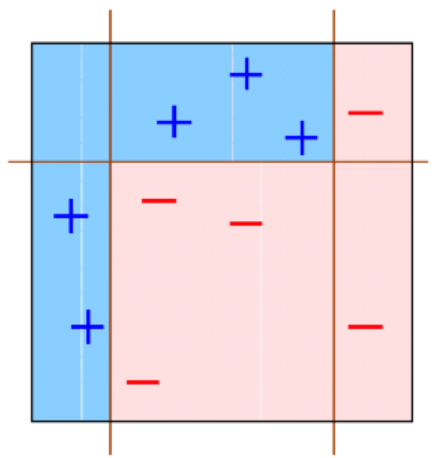


(b) $h_2, \epsilon_1 = 0.21$



(c) $h_3, \epsilon_3 = 0.14$

Adaboost gives weights $\alpha_1 = 0.42$, $\alpha_2 = 0.65$ and $\alpha_3 = 0.92$ and computes h



Adaboost error

If the error of h_i is $\frac{1}{2} - \gamma_i$, then the error of h is at most

$$exp\left(-2\sum_{i=1}^{M}\gamma_i^2\right)$$

which tends to 0 if we can guarantee $\gamma_i \geq \gamma$ for a fixed γ.

```
from sklearn.ensemble import AdaBoostClassifier
X,y = ...
ada = AdaBoostClassifier(n_estimators=100, random_state=0)
ada.fit(X, y)
```

Pros and cons

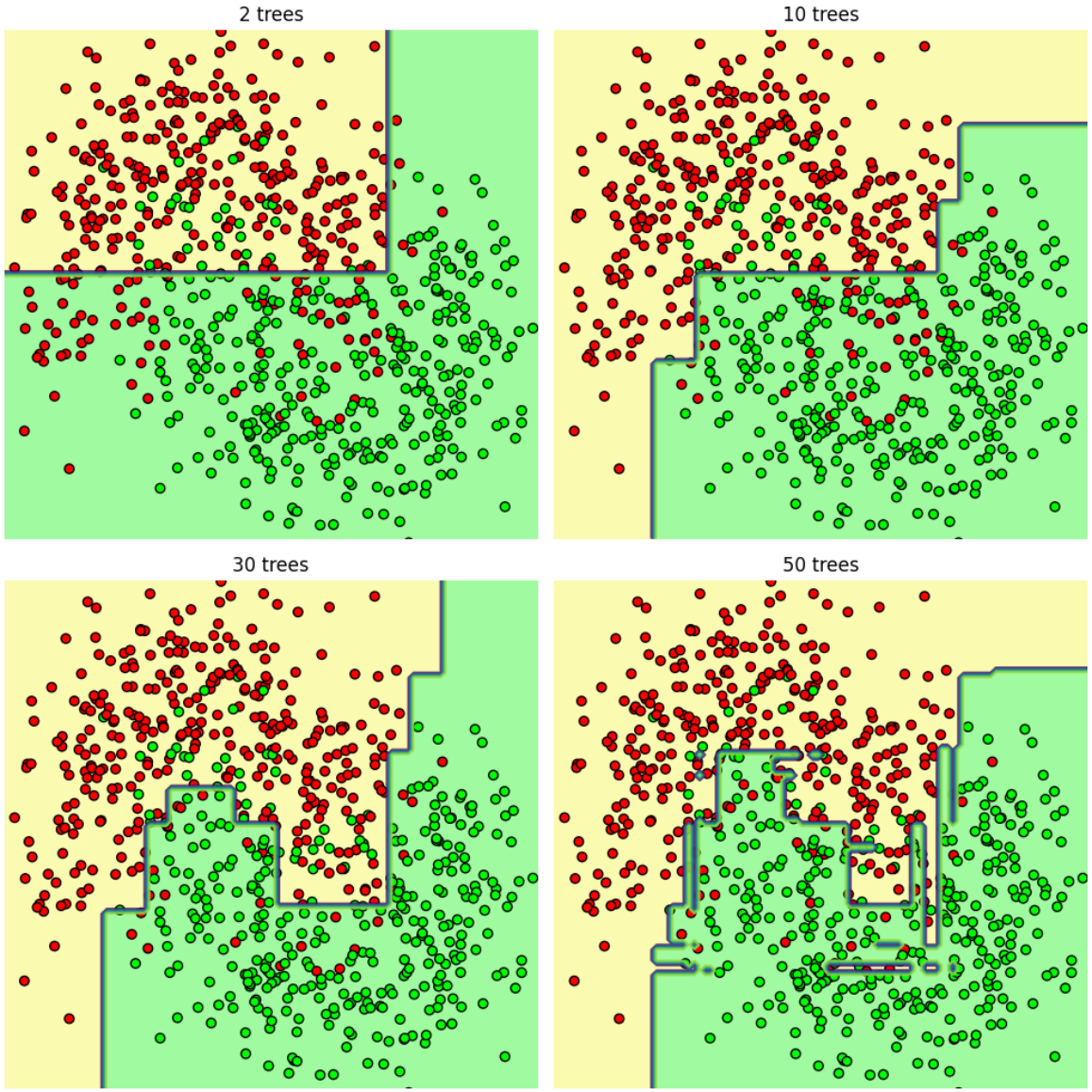
Advantages

- Simple and easy to implement
- Flexible
- No parameters to tune (except M)
- No prior knowledge needed about weak learner
- Provably effective
- Can be applied on a wide variety of problems

Drawbacks

- Performance of AdaBoost depends on data and weak classifier
- if weak classifier too complex (resp too weak) $\rightarrow$ overfitting (resp. underfitting)
- sensitive to uniforme noise

Weak classifiers: decision trees. Influence of M



Variations

- Real AdaBoost: Fits an additive logistic regression model: stagewise optimization of $\mathbb{E}\left(e^{-yf(x)}\right)$
- LogitBoost: Uses adaptive Newton steps for fitting an additive symmetric logistic model by maximum likelihood

Multiclass

One v.s. All seems to work very well most of the time.
Error output code seems to be useful when the number of classes is big.

- Aggregate multiple hypotheses generated by the same learning algorithm invoked over different distributions of training data
- Generate a classifier with a smaller error on the training data as it combines multiple hypotheses which individually have a large error

But...

1. Training set
   - Bagging replicates training sets by sampling with replacement from the training instances
   - Boosting uses all instances but weights them and therefore produces different classifiers
2. Classifiers
   - Bagging: classifiers have equal vote. Majority wins
   - Boosting: vote dependent on the classifier's accuracy. Extra weightage to the opinion of some

- Random forest algorithm is more robust and faster to train
- It can handle missing and unbalanced data
- But... the feature selection process is not explicit
- and it has weaker performance on small size training data

Definition

Adaboost is an optimization algorithm:

$$Min\ J(f) = \sum_{i=1}^{m} exp\left(-y_i f((\mathbf{x}_i))\right)$$

Gradient Boosting

Generalizing this approach with different objective functions and their gradients.

Principle

- Learn a classifier/regressor h_1
- Error: $E_{h_1} = \sum_{i=1}^{n} \ell\left(y_i, h_1(\mathbf{x}_i)\right)$
- Residual on $\mathbf{x}_i$: $e_i = y_i - h_1(\mathbf{x}_i)$
- If $\exists \hat{h}$ such that $\forall i\ \hat{h}(\mathbf{x}_i) = e_i$, then $F = h_1 + \hat{h}$ will have a null error.
- $\hat{h}$ hard to find $\Rightarrow$ Find h_2 such that $\forall i\ |h_2(\mathbf{x}_i) - e_i| < \epsilon$
- ...

Example:

$$\ell\left(y, h_1(\mathbf{x})\right) = \frac{1}{2}(y - h_1(\mathbf{x}))^2$$

Then:

$$e = y - h_1(\mathbf{x}) = -\frac{\partial}{\partial h_1(\mathbf{x})}\ell\left(y, h_1(\mathbf{x})\right)$$

e_i: opposite of the gradient.
$\Rightarrow$ new learning set $\tilde{\mathcal{S}} = \{\mathbf{x}_i, e_i\}_{1 \leq i \leq m}$ to learn h_2.

Algorithm

Algorithm 4: Gradient Boosting

Data: $Z = \{\mathbf{x}_i, y_i\}_{1 \leq i \leq n}, T, \ell$

Result: F

Learn h_1 on Z

for $2 \leq t \leq T$ **do**

Compute $(\forall i \in [\![1 \cdots n]\!])\ e_i = -\frac{\partial}{\partial h_{t-1}(\mathbf{x}_i)} \ell(y_i, h_{t-1}(\mathbf{x}_i))$

Build $\tilde{\mathcal{S}} = \{\mathbf{x}_i, e_i\}_{1 \leq i \leq n}$

Learn g on $\tilde{Z}$

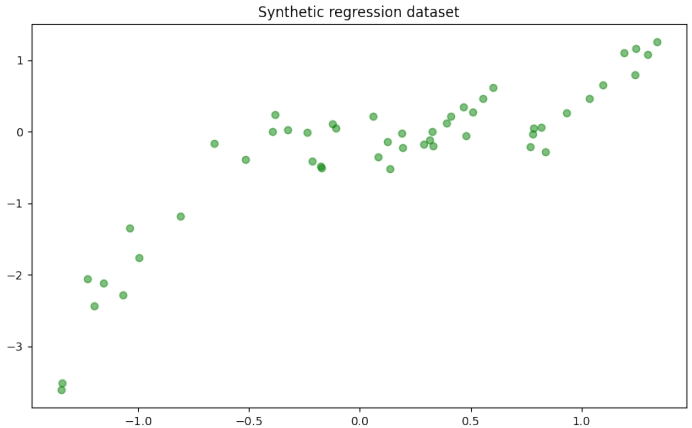
Compute $\lambda_t = arg\min_{\lambda} \left(\sum_{i=1}^{n} \ell \left(y_i, h_{t-1}(\mathbf{x}_i) + \lambda g(\mathbf{x}_i) \right) \right)$

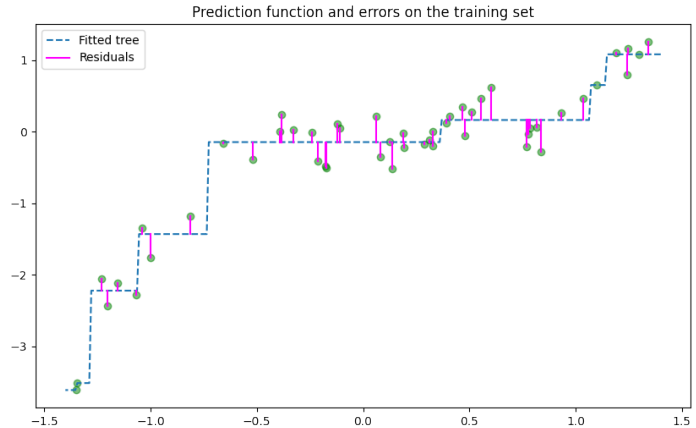
Define $h_t = h_{t-1} + \lambda_t g$

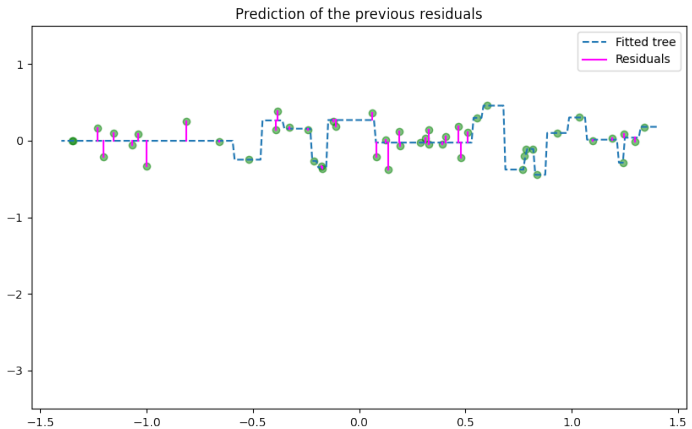
$F = h_T$

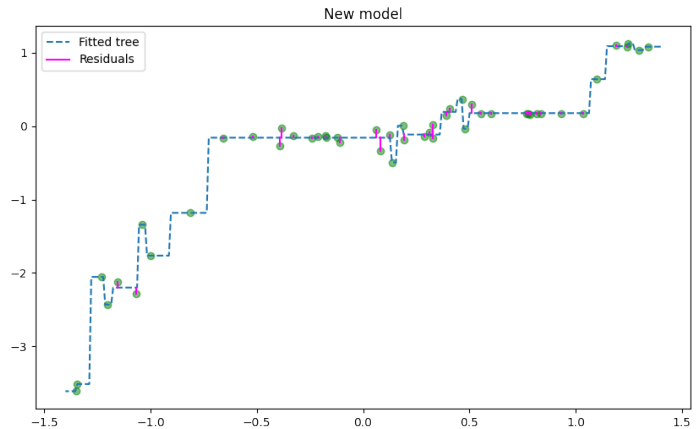
```python
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import GradientBoostingRegressor
X, y = ...
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0,max_depth=1, random_state=0)
gbc.fit(X, y)

gbr = GradientBoostingRegressor(random_state=0)
gbr.fit(X, y)
```

Example


Synthetic regression dataset


Prediction function and errors on the training set
Fitted tree
Residuals


Prediction of the previous residuals
Fitted tree
Residuals


New model
Fitted tree
Residuals