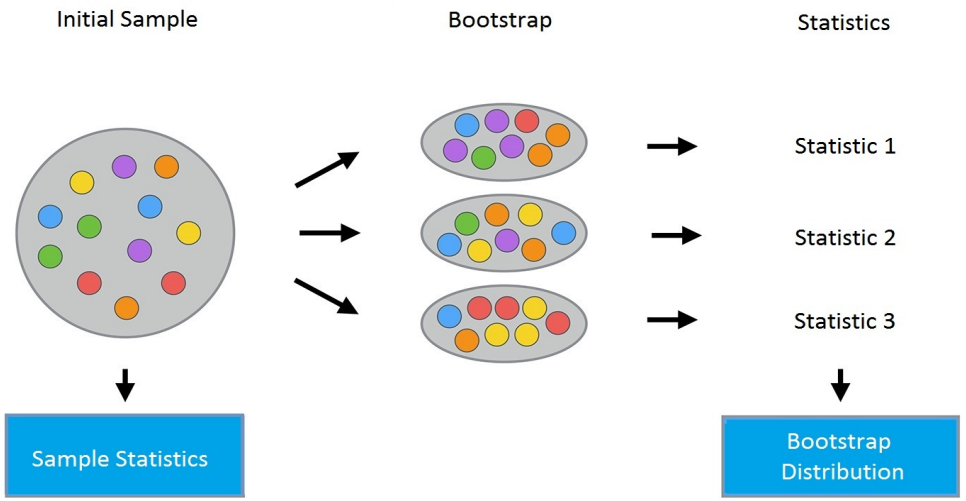
Bagging

Bagging = Bootstrap +aggregation

Bootstrap:引导程序

A standard "resampling" technique from statistics for estimating quantities about a population by averaging estimates from multiple small data samples. Importantly, samples are constructed by drawing observations from a large data sample one at a time and returning them to the data sample after they have been chosen.



https://mlcourse.ai/articles/topic5-part1-bagging/

Bagging

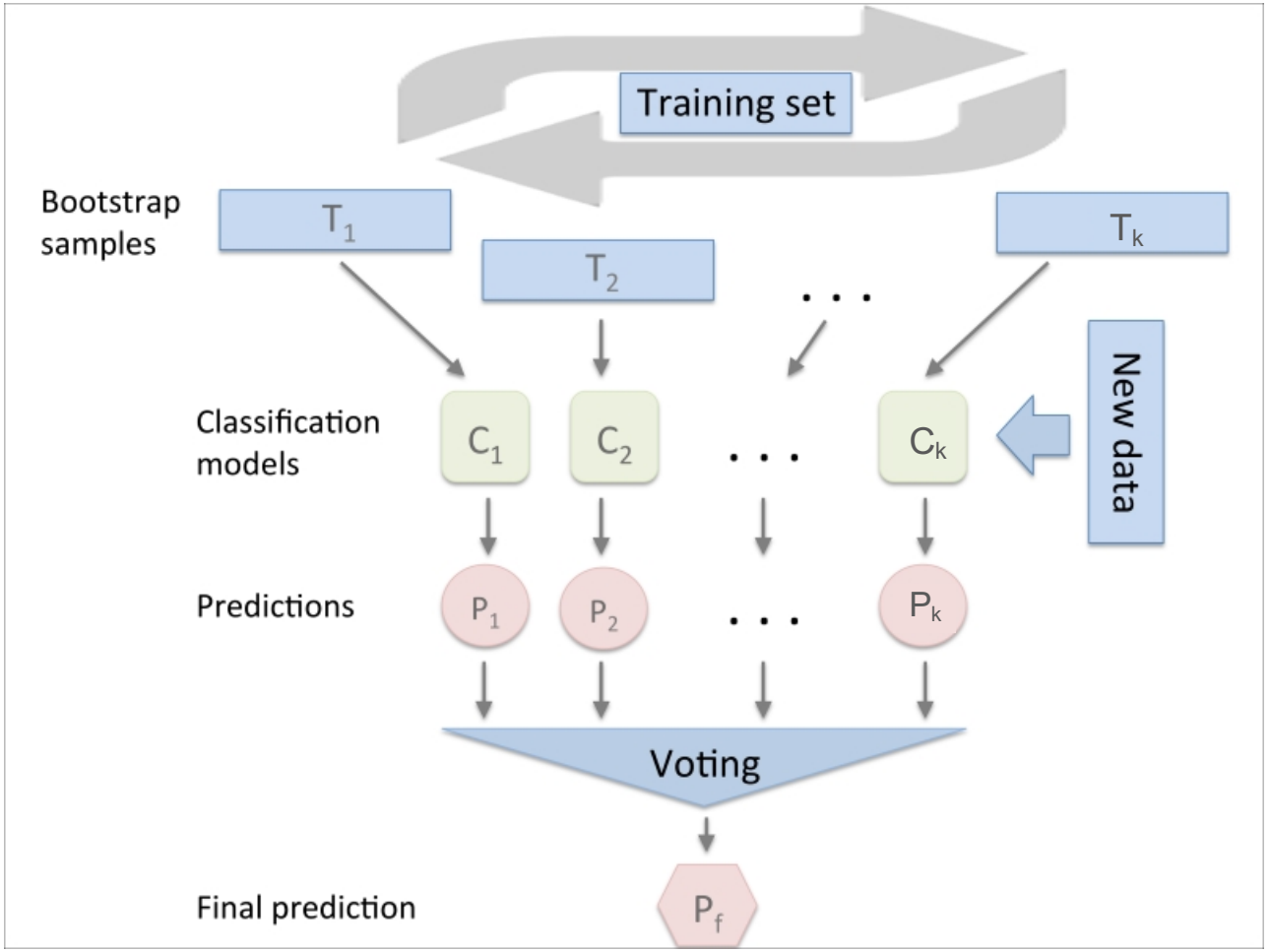
Bootstrap:

- Create a random subset of data by sampling with replacement
- Draw m' samples from m sample with replacement ($m' \leq m$)

Bagging:

- Repeat k times to generate k subsets
  - Some of the samples get repeated and some will not be left out
- Train one classifier on each subset
- To test, aggregate the output of k classifiers that you trained in the previous step using either majority vote / unweighted average

Bagging



https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781783555130/7/ch07lvl1sec46/bagging-building-an-ensemble-of-classifiers-from-bootstrap-samples

Bias / Variance in Bagging

Error of any model has two components:

- Bias: due to model choice which
  - can be reduced by increasing complexity
- Variance: due to small sample size or high complexity of the model
  - Can be reduced by increasing the data or reducing the complexity

Bagging is used to reduce variance among different models, cannot reduce Bias!

Bagging is applied on a collection of low-bias high-variance models and by averaging them the bias would not get affected, however the variance will be reduced

Bagging error

Assumption:

- If learners are independent
- If each learner makes an error with probability p (same error probability for all learners)

The probability that k' out of k learner make an error is:

$$\binom{k}{k'} p^{k'} (1-p)^{k-k'}$$

If we use majority voting to decide about the output, then the error happens if more than $\frac{k}{2}$ of learners make an error, so the error for majority voting is:

$$\sum_{k' > \frac{k}{2}} \binom{k}{k'} p^{k'} (1-p)^{k-k'}$$

If $k = 10$ and $p = 0.1$, then $error(majority\ voting) < 0.001$

Bagging

Advantage:

- Reduces overfitting (harder for the aggregated model to memorize full dataset, since some of the data will not be seen by the training models)
- This improves performance in almost all cases specially if learning scheme is unstable (i.e. decision trees are unstable because with slightly different sub-sample, the tree structure may change drastically)
- Can be applied to numeric prediction and classification
- Can help a lot if data is noisy

Bagging more precisely

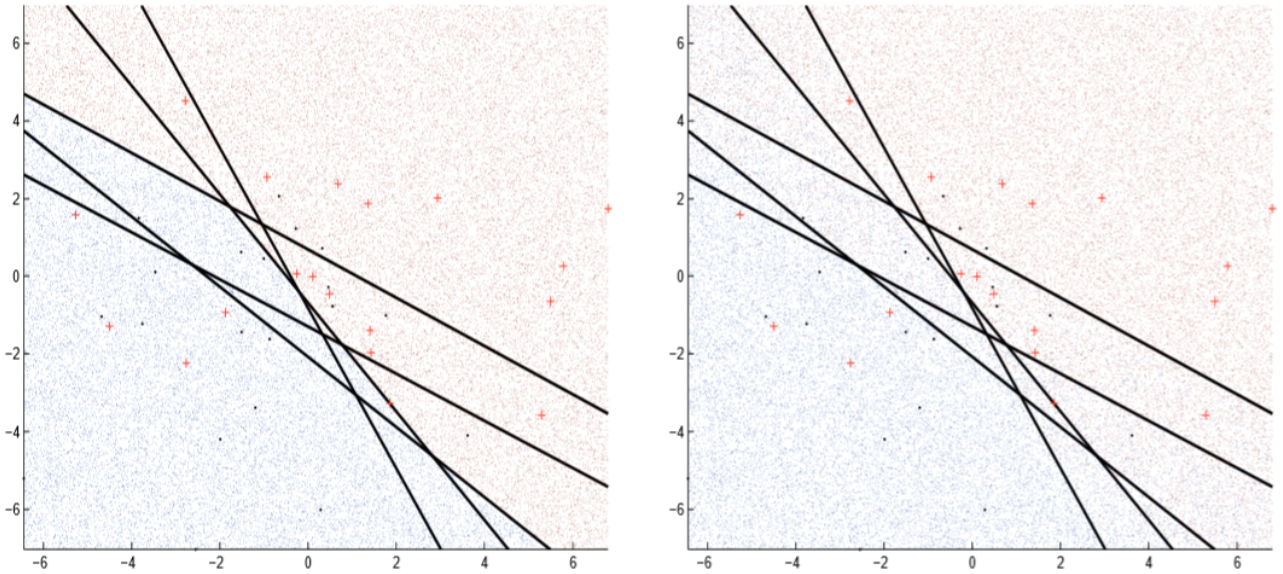
Algorithm Bagging$(D, T, \mathcal{A})$ // train ensemble from bootstrap samples

Input: dataset D; ensemble size T; learning algorithm $\mathcal{A}$.

Output: set of models; predictions to be combined by voting or averaging.

```
for t = 1 to T do
    bootstrap sample D_t from D by sampling |D| examples with replacement
    run A on D_t to produce a model M_t
end
return {M_t | 1 ≤ t ≤ T}
```

Bagging linear classifiers



(left) An ensemble of five basic ***linear classifiers*** built from bootstrap samples with bagging. The decision rule is majority vote, leading to a piecewise linear decision boundary. (right) If we turn the votes into probabilities, we see the ensemble is effectively grouping instances in different ways, with each segment obtaining a slightly different probability.

Bagging trees

Example:

模拟的

An experiment with simulated data:

- A sample set of size $\mathrm{m} = 30$, two classes $\{0,1\}$ and 5 features
- Each feature has a standard Gaussian distribution with pairwise correlation 0.95
- The outputs are generated according to:

$$\Pr(Y = 1 \mid x_1 \leq 0.5) = 0.2 \text{ and } \Pr(Y = 1 \mid x_1 > 0.5) = 0.8$$

- A test set of size 2000 from same population
- The Bayesian error for this data is 0.2

Bagging trees

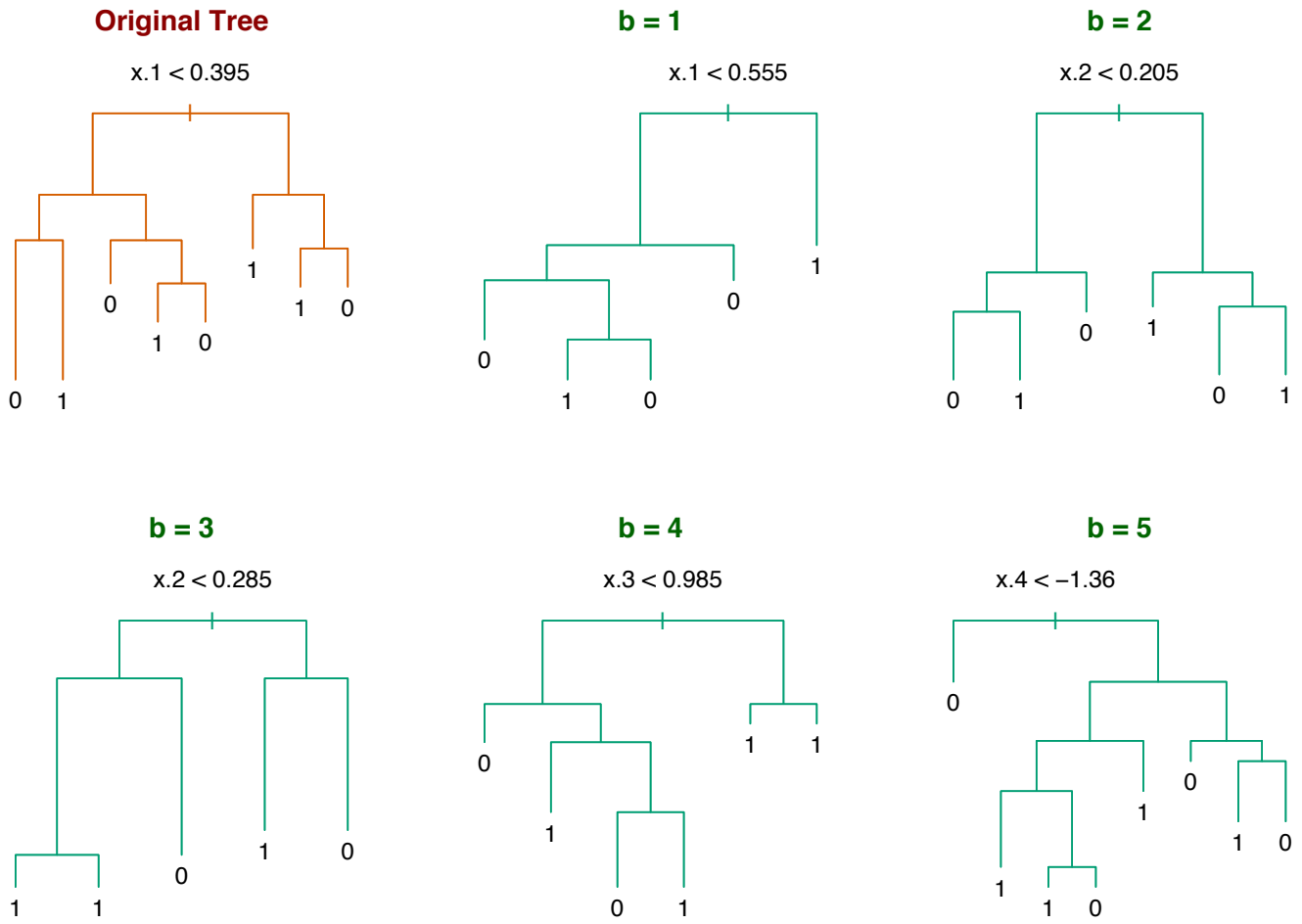
Approach:

- fit classification trees to training sample using 200 bootstrap samples
- No pruning was used
- trees are different (tree induction is *unstable*)
- therefore have high variance

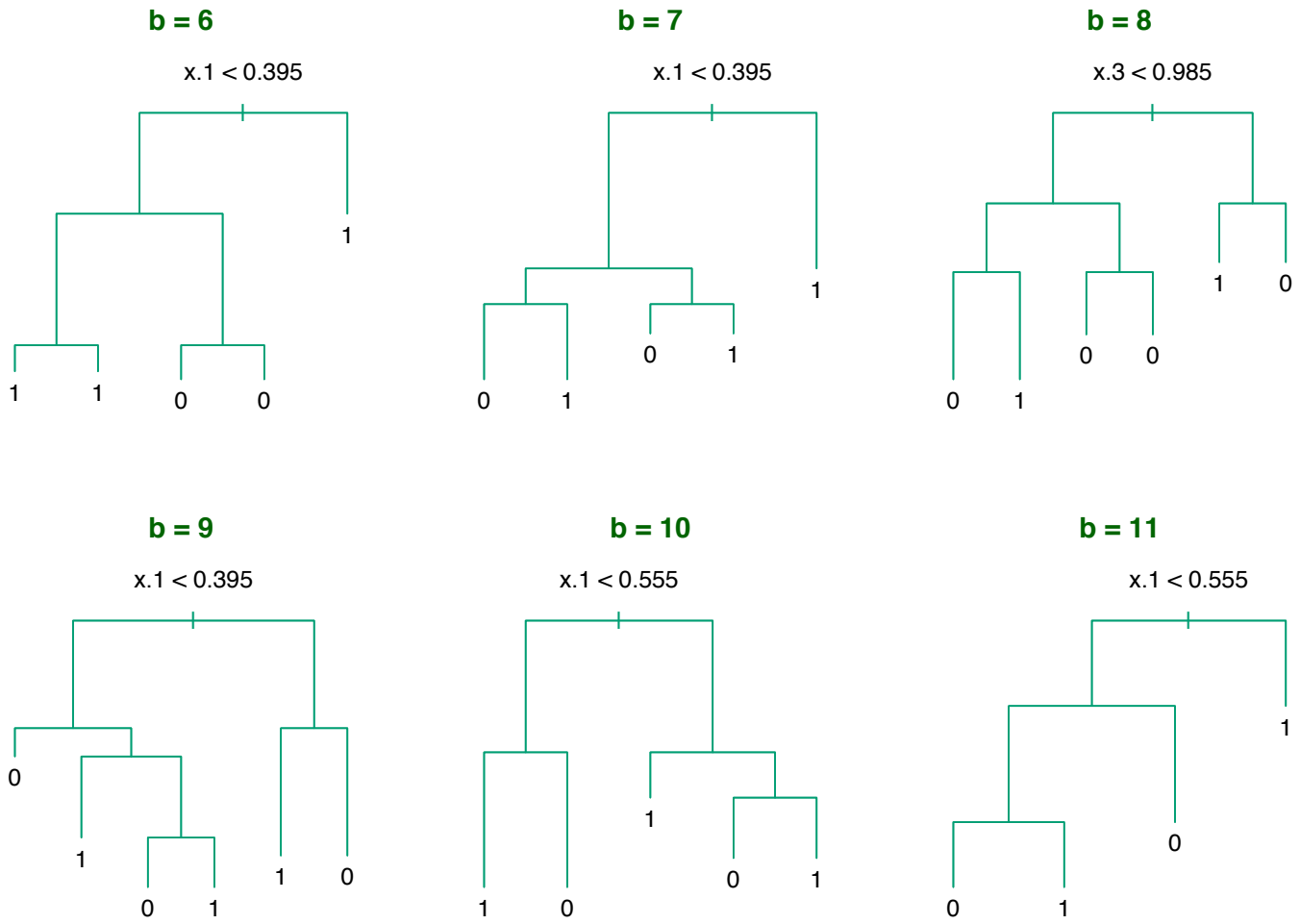
Results:

- averaging reduces variance and leaves bias unchanged

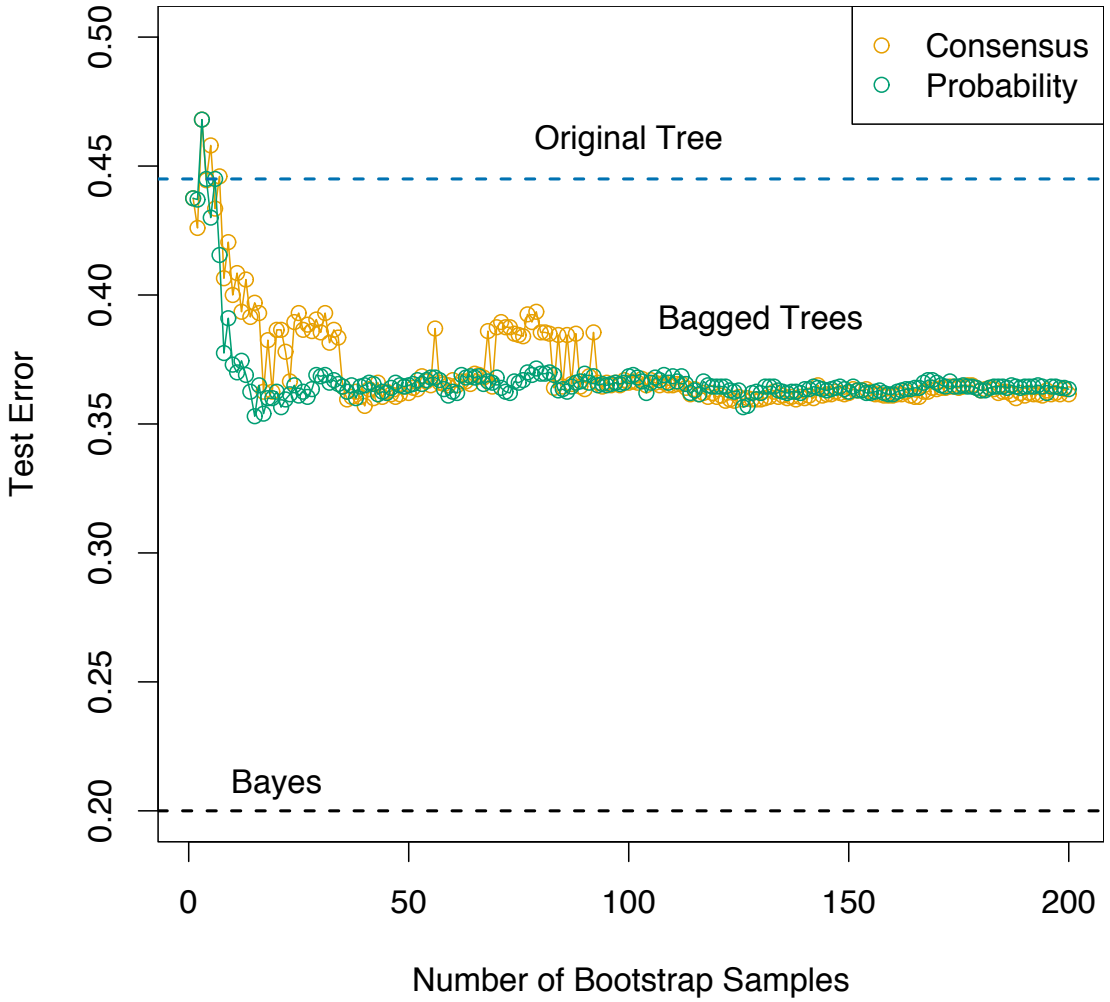
Bagging trees



Bagging trees



Bagging trees



Bagging trees

Pros:

- Reduces the overfitting
- Generalizes better

Cons:

- when we bag a model, any simple structure is lost
  - this is because a bagged tree is no longer a tree ... but a forest, so this reduces claim to interpretability
  - *stable* models like nearest neighbor not very affected by bagging but *unstable* models like trees most affected by bagging
- With lots of data, we usually learn the same classifier, so averaging them does not help

Random Forest

What is the solution for ensemble decision trees with lots of data?

- Have extra variability in the learners and introduce more randomness to the procedure

How?

- For every model, use only a subset of features
  - This will create diversity in the trees
- Then, similar to before, take average/majority vote over trees (learners)

An ensemble tree with random subset of features is called Random Forest

Random Forests

Algorithm RandomForest(D, T, d) // train ensemble of randomized trees

Input: data set D; ensemble size T; subspace dimension d.

Output: set of models; predictions to be combined by voting or averaging.

1 **for** $t = 1$ to T **do**
2 | bootstrap sample D_t from D by sampling $|D|$ examples with replacement
3 | select d features at random and reduce dimensionality of D_t accordingly
4 | train a tree model M_t on D_t without pruning
5 **end**
6 **return** $\{M_t | 1 \leq t \leq T\}$

Random Forest (PlayTennis dataset)

Day	Outlook	Temp.	Humidity	Wind	Play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Random Forest (PlayTennis dataset)

Example:

Step 1: select 14 samples from the dataset with replacement

Day	Outlook	Temp.	Humidity	Wind	Play
D3	Overcast	Hot	High	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D3	Overcast	Hot	High	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D14	Rain	Mild	High	Strong	No
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D12	Overcast	Mild	High	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D9	Sunny	Cool	Normal	Weak	Yes
D1	Sunny	Hot	High	Weak	No
D14	Rain	Mild	High	Strong	No

Random Forest (PlayTennis dataset)

Example:

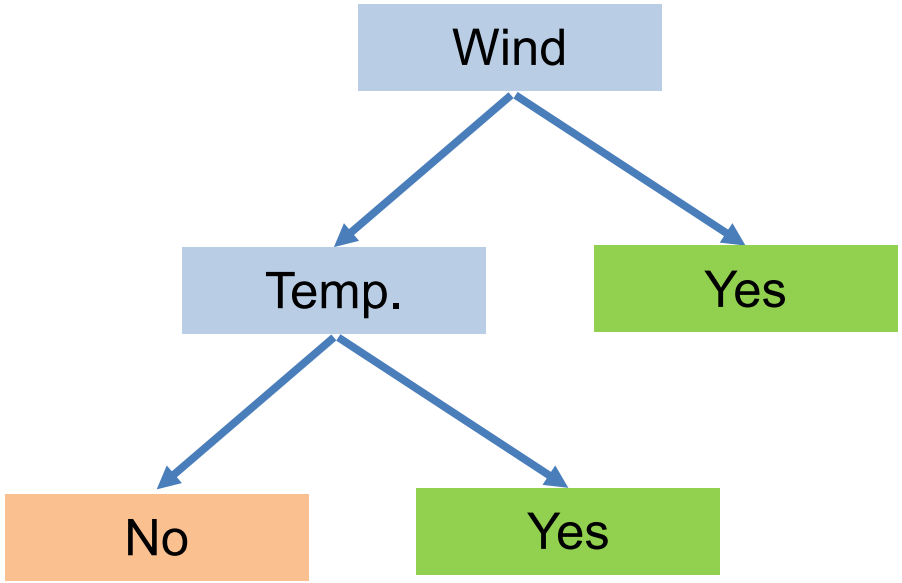
Step 2: select a subset of features randomly (for example 2 features out of 4)

Day	Temp.	Wind	Play
D3	Hot	Weak	Yes
D11	Mild	Strong	Yes
D3	Hot	Weak	Yes
D8	Mild	Weak	No
D14	Mild	Strong	No
D5	Cool	Weak	Yes
D6	Cool	Strong	No
D12	Mild	Strong	Yes
D8	Mild	Weak	No
D9	Cool	Weak	Yes
D10	Mild	Weak	Yes
D11	Mild	Strong	Yes
D9	Cool	Weak	Yes
D1	Hot	Weak	No
D14	Mild	Strong	No

Random Forest (PlayTennis dataset)

Example:

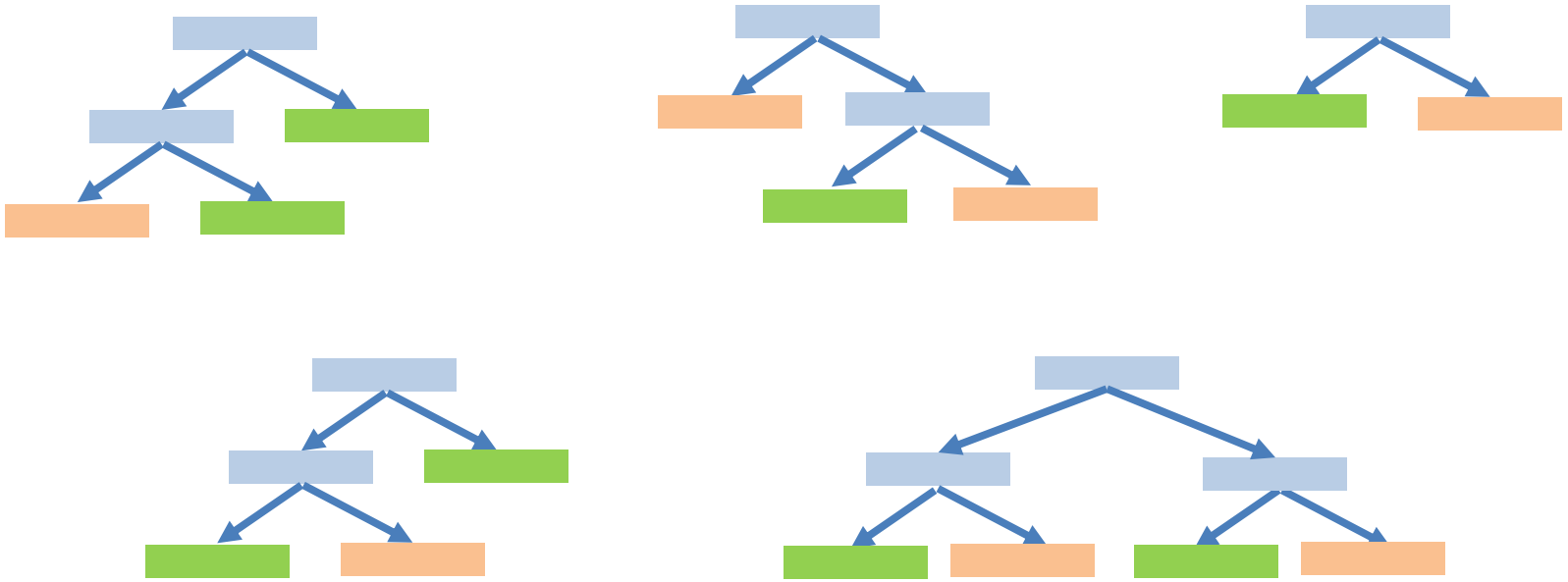
Step 3: build a full tree without pruning using the selected features and samples



Random Forest (PlayTennis dataset)

Example:

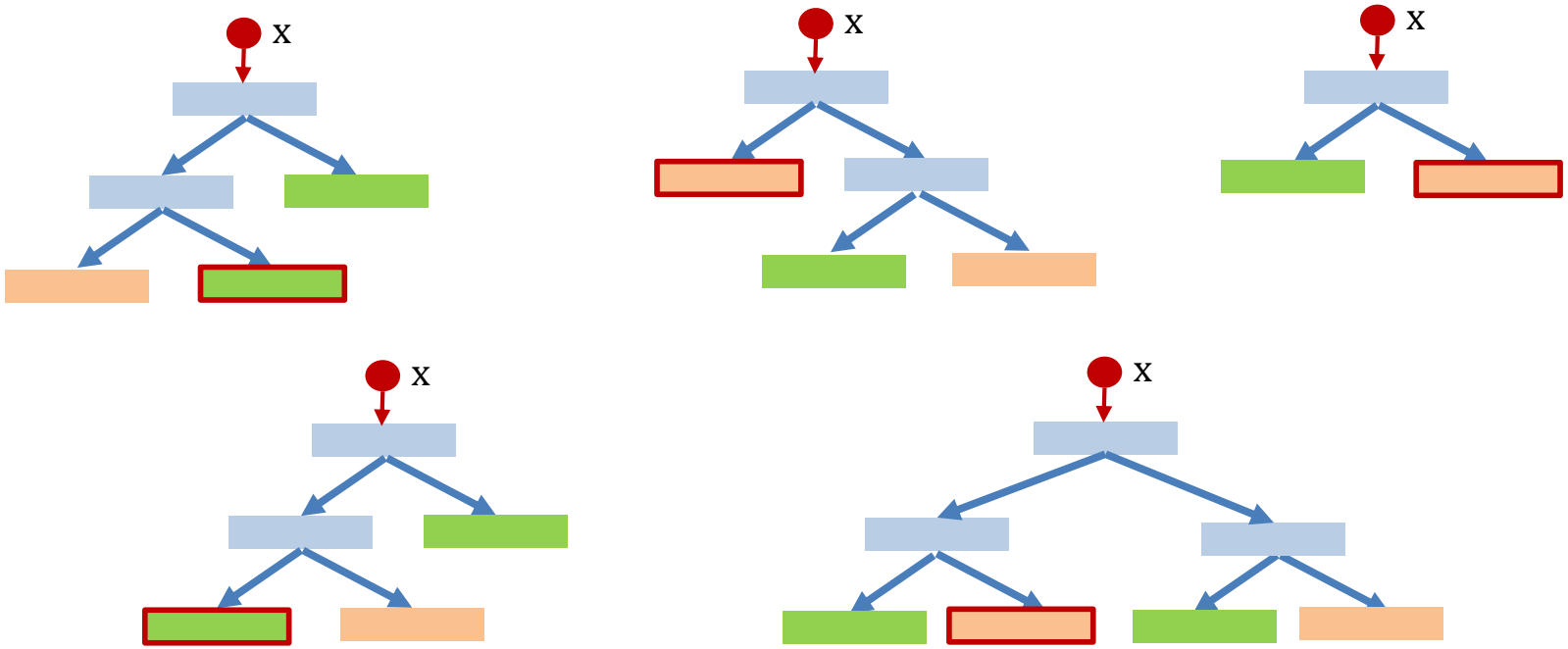
Step 4: Repeat previous steps $\mathrm{k} = 5$ times (the value for k can be tuned by using cross validation)



Random Forest (PlayTennis dataset)

Example:

For every new sample, predict the output using all trees and then take the average/majority vote



Random Forests

Leo Breiman's Random Forests algorithm is essentially like Bagging for trees, except the ensemble of tree models is trained from bootstrap samples and random subspaces.

- each tree in the forest is learned from
  - a bootstrap sample, i.e., sample from the training set with replacement
  - a subspace sample, i.e., randomly sample a subset of features
- advantage:
  - forces more diversity among trees in ensemble
  - less time to train since only consider a subset of features

Note: combining linear classifiers in an ensemble gives a piecewise linear (i.e., non-linear) model

Randomization

- Some algorithms already have a random component: e.g., initial weights in a neural net
- Most algorithms can be randomized, e.g., greedy algorithms:
  - Pick N options at random from the full set of options, then choose the best of those N choices
  - E.g.: attribute selection in decision trees
- More generally applicable than bagging: e.g., we can use random subsets of features in a nearest-neighbor classifier
  - Bagging does not work with stable classifiers such as nearest neighbor classifiers
- Can be combined with bagging
  - When learning decision trees, this yields the Random Forest method for building ensemble classifiers

Ensemble methods

So far we have seen several ensemble methods:

1. Simple ensembles like majority vote or unweighted average
2. weighted averages / weighted votes
3. Treat the output of each model as a feature
4. Mixture of experts
5. Bagging (Bootstrap Aggregation)
6. Adding randomization to introduce diversity in the models (e.g. feature subset selection)
7. What is next?

Boosting

So far we have focused on building parallel learners and then combine their decision (equal weights / unequal weights)

Problem with parallel learners:

what if all learners are mistaken in the same region?

Boosting:

- Uses "weak" learners
- Learners are trained sequentially
- New learners focus on errors of earlier learners
- New learners try to get these "hard" examples right by operating on a weighted training set in favor of misclassified instances
- Combine all learners in the end

Boosting is a method that convert a sequence of weak learners into a very complex model to predict

Boosting

Re-weighted training set:

- Start with same weight for all the instances
- Misclassified instances gain higher weights: so the next classifier is more likely to classify it correctly
- Correctly classified instances lose weight

Learning with a weighted training set:

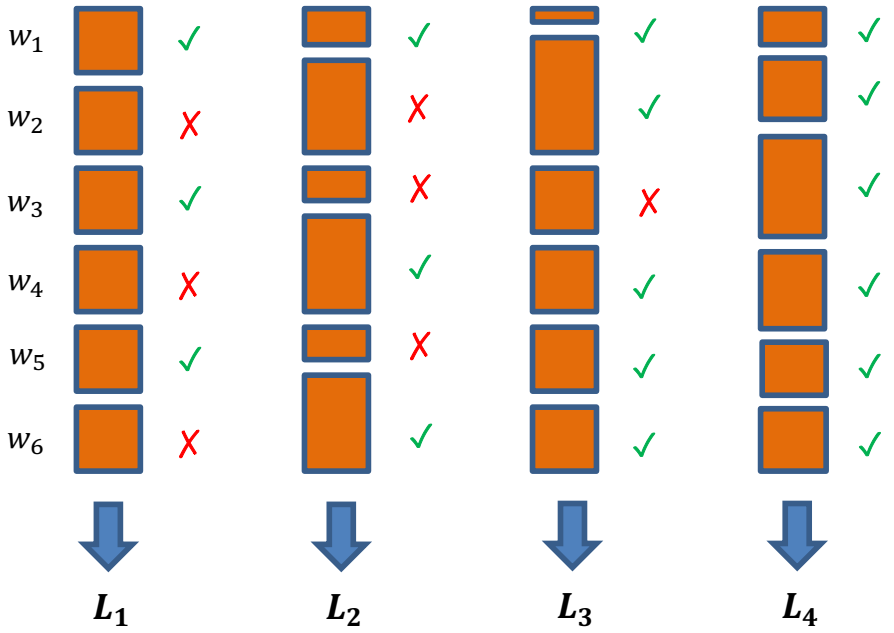
1. Give different weights to the loss function for different instances
   - So the algorithm will be forced to learn instances with higher weights
2. Create a new collection of data with multiple copies of samples with higher sample weight

Boosting algorithm

1. set $w_i = 1$ for $i = 1, \ldots, n$
2. Repeat until sufficient number of hypothesis
    - Train model L_j using the dataset with weight w
    - Increase w_i for misclassified instances of L_j
3. Ensemble hypothesis is the weighted majority/weighted average of k learners $L_1, \ldots, L_k$ with weight $\lambda_1, \ldots, \lambda_k$ which are proportional to the accuracy of L_j

Boosting

Reweighting training data:



We always aim to minimize some cost function:

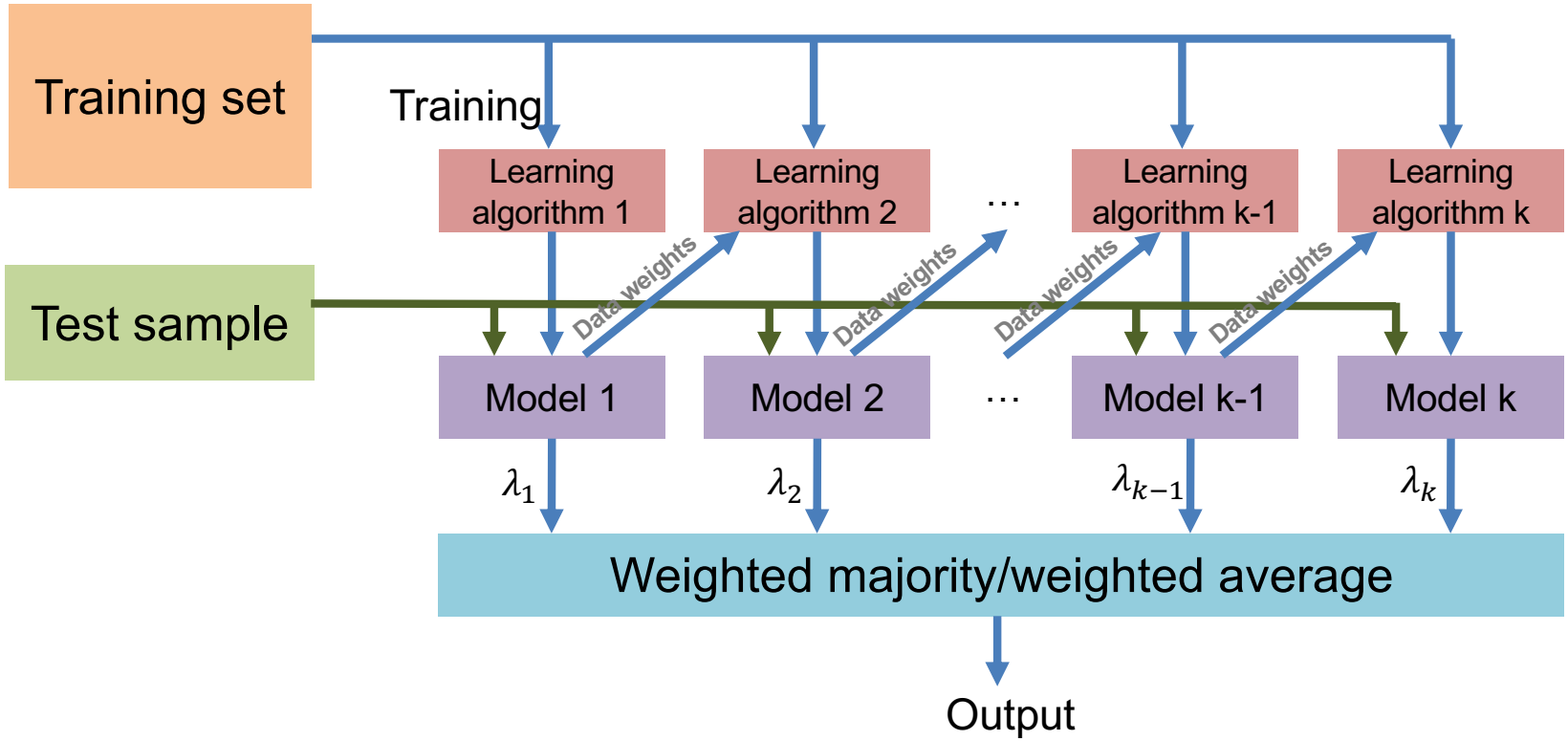
- Unweighted average loss:

$$J(\theta) = \frac{1}{N}\sum_i J_i\,(\theta, \mathrm{x}_i)$$

- Weighted average loss:

$$J(\theta) = \sum_i w_i J_i\,(\theta, \mathrm{x}_i)$$

Boosting


Training set
Training
Learning algorithm 1
Learning algorithm 2
…
Learning algorithm k-1
Learning algorithm k
Test sample
Data weights
Data weights
Data weights
Data weights
Model 1
Model 2
…
Model k-1
Model k
λ_1
λ_2
λ_{k-1}
λ_k
Weighted majority/weighted average
Output

Boosting

Boosting works well as long as we use weak learners.

- weak learner is a model that is slightly better than random

Examples of weak learners:

- Perceptron
- Decision stumps (trees with one node)
- Naïve Bayes models
- etc.

AdaBoost (Adaptive Boosting)

- AdaBoost usually uses **stump trees** (trees with one node and two leaves) as the base learner
    - Not very accurate at classification on their own
- AdaBoost combines stumps to boost the performance so it creates a forest of stumps instead of forest of tress
- In AdaBoost, stumps are created sequentially
- The error of each stump affects the training data weight in the next stump
- Depending on the performance, each stump gets different weight (λ_i) in the final classification decision

AdaBoost (Adaptive Boosting)

- $w_i = \frac{1}{N} \ \forall \ i$
- For j = 1 to k do
  - Learn L_j with data weight w
  - $\epsilon_j = \sum_{i=1}^{n} w_i^{(j)} 1[L_j(\mathrm{x}_i) \neq y_i] / \sum_{i=1}^{n} w_i^{(j)}$
  - $\lambda_j = \frac{1}{2} \log(\frac{1-\epsilon_j}{\epsilon_j})$
  - $w_i^{(j+1)} = w_i^{(j)} \exp(\lambda_j)$ for misclassified instances
  - $w_i^{(j+1)} = w_i^{(j)} \exp(-\lambda_j)$ for correct instances
- End

- Make prediction using the final model: $y(\mathrm{x}) = sign(\sum_{j=1}^{k} \lambda_i L_j(\mathrm{x}))$

AdaBoost

Example:

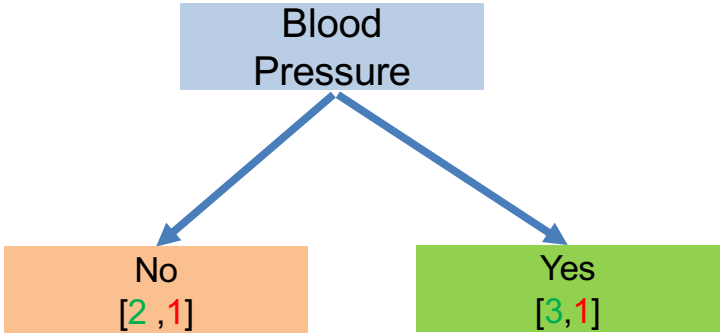
Step 1: give similar weigh to your sample (weights have to be normalized)

Blood pressure	Weight	Age	Heart disease	Sample weight
Yes	68	56	Yes	$1/7$
No	75	44	No	$1/7$
Yes	80	35	Yes	$1/7$
Yes	76	49	No	$1/7$
No	78	50	Yes	$1/7$
No	83	38	No	$1/7$
Yes	85	60	Yes	$1/7$

AdaBoost

Step 2: Find the best stump by testing each feature/attribute and compute the stump weight (λ_1)

Blood pressure	Weight	Age	Heart disease	Sample weight
Yes	68	56	Yes	$1/7$
No	75	44	No	$1/7$
Yes	80	35	Yes	$1/7$
Yes	76	49	No	$1/7$
No	78	50	Yes	$1/7$
No	83	38	No	$1/7$
Yes	85	60	Yes	$1/7$



$$Total\ error = \frac{2}{7}$$

$$\lambda_1 = \frac{1}{2}\log\left(\frac{1 - Total\ error}{Total\ error}\right) = 0.45$$

AdaBoost

Step 3: Modify sample weights (w_i)

Blood pressure	Weight	Age	Heart disease	Sample weight
Yes	68	56	Yes	$1/7$
No	75	44	No	$1/7$
Yes	80	35	Yes	$1/7$
Yes	76	49	No	$1/7$
No	78	50	Yes	$1/7$
No	83	38	No	$1/7$
Yes	85	60	Yes	$1/7$

$New\ w_i = w_i \times e^{\lambda_i}$ for misclassified samples

$$w_4 = \frac{1}{7} \times e^{0.45} = 0.22$$

$$w_5 = \frac{1}{7} \times e^{0.45} = 0.22$$

$New\ w_i = w_i \times e^{-\lambda_i}$ for correct samples

$$w_1 = \frac{1}{7} \times e^{-0.45} = 0.09$$

$$w_1 = w_2 = w_3 = w_6 = w_7 = 0.09$$

AdaBoost

Step 4: Normalise sample weights

Blood pressure	Weight	Age	Heart disease	Sample weight	New weight	Norm. weight
Yes	68	56	Yes	$1/7$	0.09	0.1
No	75	44	No	$1/7$	0.09	0.1
Yes	80	35	Yes	$1/7$	0.09	0.1
Yes	76	49	No	$1/7$	0.22	0.25
No	78	50	Yes	$1/7$	0.22	0.25
No	83	38	No	$1/7$	0.09	0.1
Yes	85	60	Yes	$1/7$	0.09	0.1

Normalized $w_i = \frac{w_i}{\sum w_i}$

AdaBoost

Step 5: create a new stump using the weights

Blood pressure	Weight	Age	Heart disease	Norm. weight
Yes	68	56	Yes	0.1
No	75	44	No	0.1
Yes	80	35	Yes	0.1
Yes	76	49	No	0.25
No	78	50	Yes	0.25
No	83	38	No	0.1
Yes	85	60	Yes	0.1

There are two ways to include the weights:

- use weighted information gain to find the best next stump

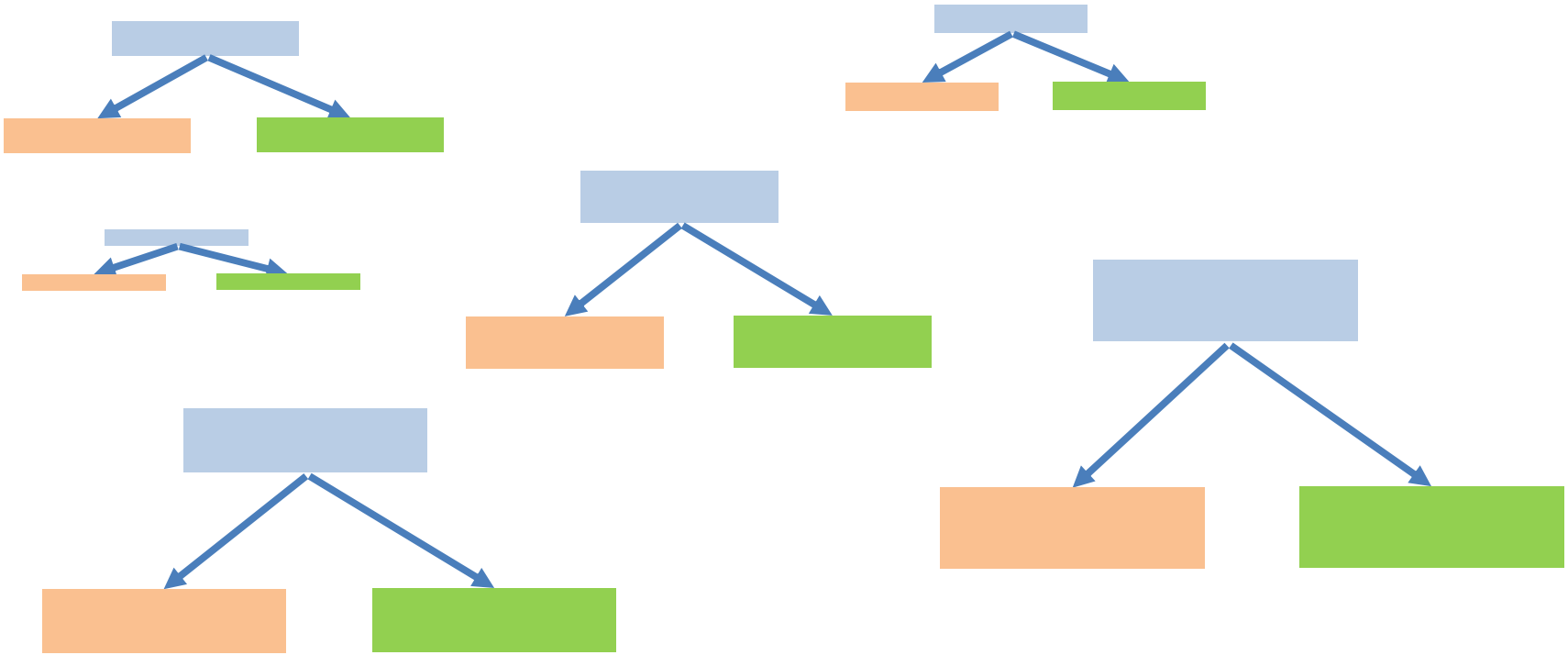
or

- generate new sample collection from the training set based on the sample weights and find the best stump on that

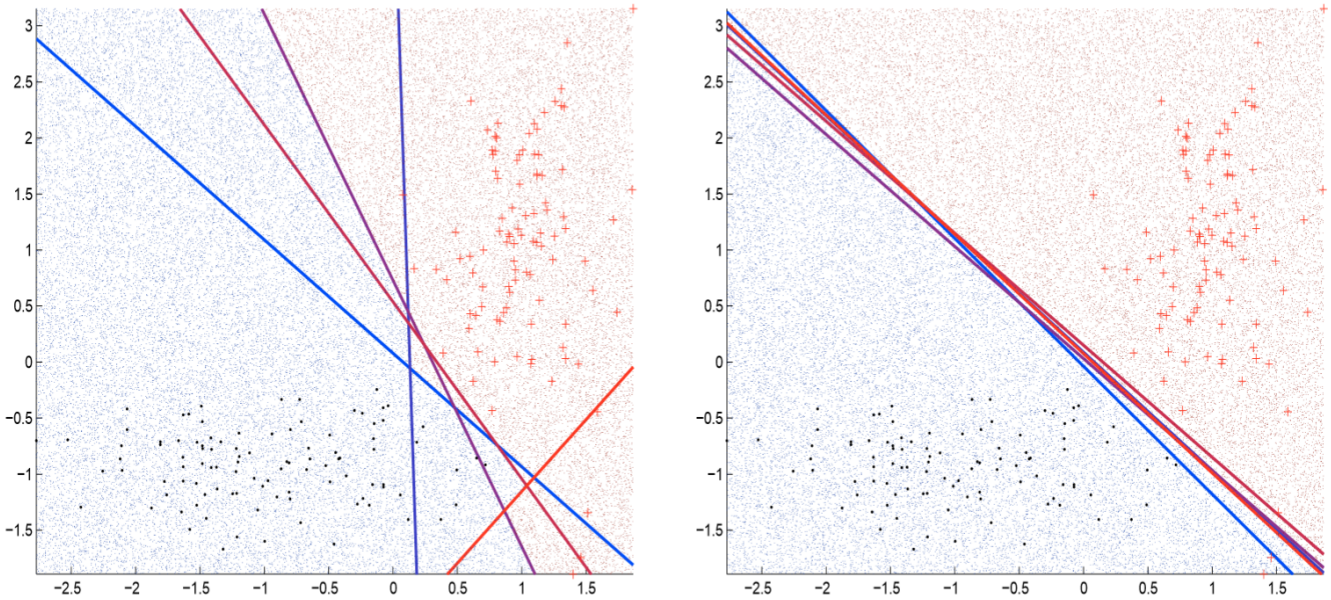
AdaBoost

- Go back to step 2 and repeat k times

You will end up with k stumps that each has a weight for final prediction and you will pass your test sample to all of them and take the weighted majority vote/averagre



Boosting



(left) An ensemble of five boosted basic linear classifiers with majority vote. The linear classifiers were learned from blue to red; none of them achieves zero training error, but the ensemble does. (right) Applying bagging results in a much more homogeneous ensemble, indicating that there is little diversity in the bootstrap samples.

Boosting

Advantages:

- No need to use complex models
- Can boost the performance of any weak learner
- Very simple to implement
- Decreasing the bias
- Decreasing the variance
- Good generalization

Disadvantage:

- Lack of interpretability
- Slow during training and potentially testing

Gradient Boosting

Gradient boosting is applying similar idea for regression (however gradient boosting can also be used for classification as well).

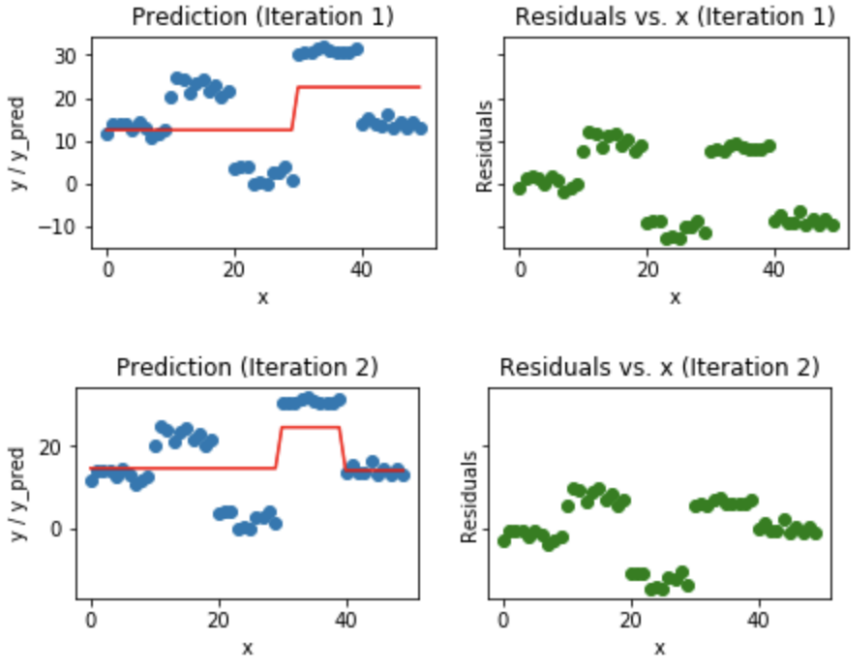
Simple linear regression or simple regression trees can be used as weak learners

Repeat the following procedure:

- Learn a regression predictor
- Compute the error residual
- learn to predict residual

Gradient Boosting

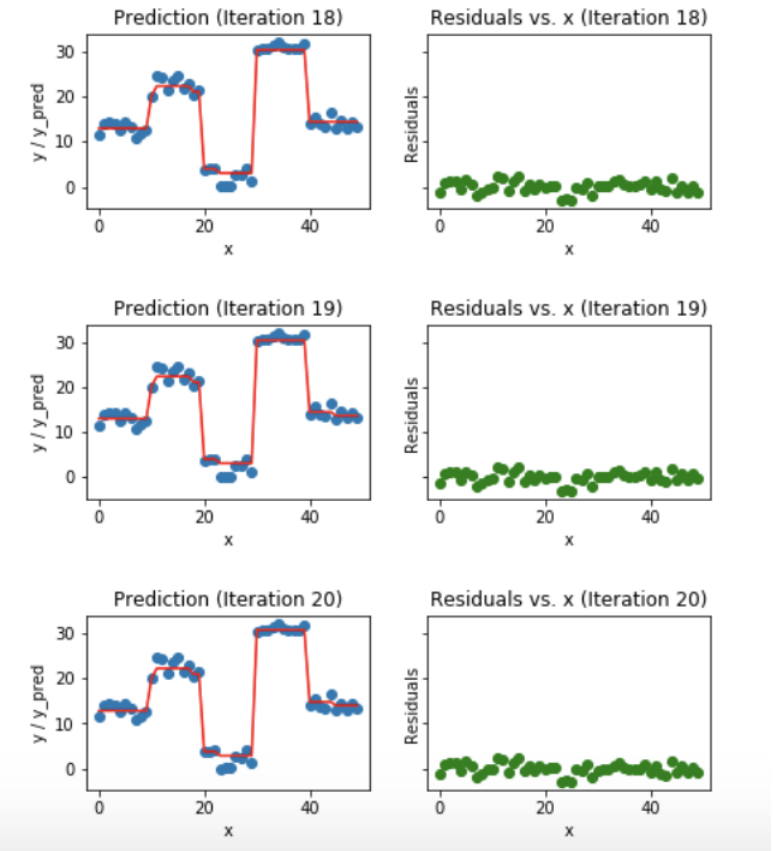
- It learn a sequence of predictors
  - Learns a new predictor for the residuals at each iteration
- Sum of the predictors creates more complex model
- The model gets more accurate at each step adding more predictors



https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Gradient Boosting

- In this example regression trees are used as the base learner



Gradient Boosting

- Start with training a weak learner and make a set of predictions $\hat{y}_i$
- The error of our prediction is = $J(y_i, \hat{y}_i)$
    - If we are working with MSE then $J = \sum_i (y_i - \hat{y}_i)^2$
- We can improve $\hat{y}_i$ by gradually reducing the error
  - $\hat{y}_i = \hat{y}_i + \alpha\, \partial J(y, \hat{y}) / \partial \hat{y}_i$ (we want the loss function to move towards the actual y_i or to be minimised)
    - For the MSE, $\nabla J(y_i, \hat{y}_i) = y_i - \hat{y}_i$ and
    - we can estimate this with $f(i) \approx \nabla J(y_i, \hat{y}_i)$
- Each new learner s estimating the gradient of loss $f_k(i)$

Gradient descent: taking sequence of steps to reduce $J(y_i, \hat{y}_i)$

Gradient boosting: sum of predictors weighted by some step size α

Ensemble Learning

Important points to remember

Low-bias models tend to have high variance, and vice versa.

Bagging is predominantly a variance-reduction technique, while boosting is primarily a bias-reduction technique which reduces the variance by taking the weighted average.

This explains why bagging is often used in combination with high-variance models such as tree models (as in Random Forests), whereas boosting is typically used with high-bias models such as linear classifiers or univariate decision trees (also called decision stumps).

Ensemble Learning

- Bias-variance decomposition breaks down error, suggests possible fixes to improve learning algorithms
- Stability idea captures aspects of both bias and variance
- Bagging is a simple way to run ensemble methods
- Random Forests are a popular bagging approach for trees
- Boosting has a more theoretically justified basis and may work better in practice to reduce error, but can be susceptible 易受影响的 to very noisy data
- Many other variants of ensemble learning

Acknowledgements

- Material derived from slides for the book “Elements of Statistical Learning (2nd Ed.)” by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009) http://statweb.stanford.edu/~tibs/ElemStatLearn/
- Material derived from slides for the book “Machine Learning: A Probabilistic Perspective” by P. Murphy MIT Press (2012) http://www.cs.ubc.ca/~murphyk/MLbook
- Material derived from slides for the book “Machine Learning” by P. Flach Cambridge University Press (2012) http://cs.bris.ac.uk/~flach/mlbook
- Material derived from slides for the book “Bayesian Reasoning and Machine Learning” by D. Barber Cambridge University Press (2012) http://www.cs.ucl.ac.uk/staff/d.barber/brml
- Material derived from slides for the book “Machine Learning” by T. Mitchell McGraw-Hill (1997) http://www-2.cs.cmu.edu/~tom/mlbook.html
- Material derived from slides for the course “Machine Learning” by A. Srinivasan BITS Pilani Goa Campus, India (2016)