Validation and Interpretation

Bias-Variance Decomposition

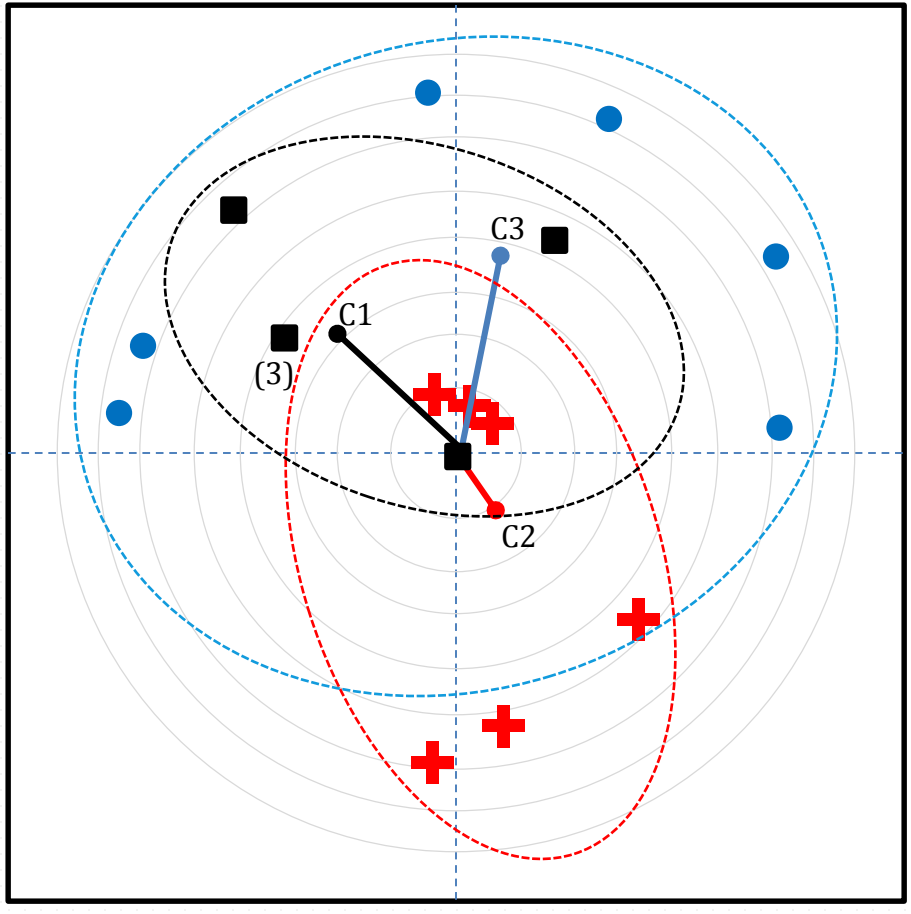
A formal method for analyzing the prediction error of a predictive model.

The Intuition

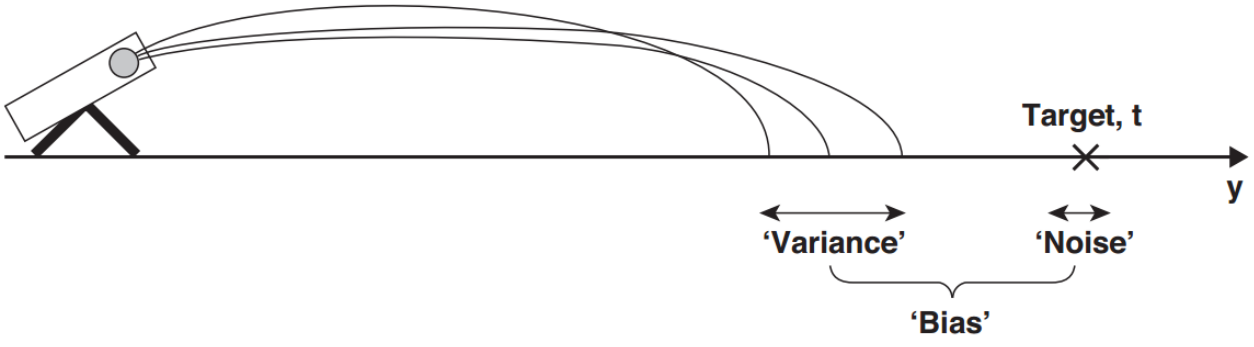
Bias

Variance

Noise



The Intuition



The Intuition

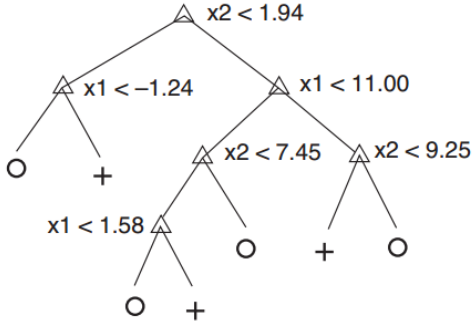
$$\boldsymbol{d_{f,\theta}(y,t) = \mathbf{Bias}_{\theta} + \mathbf{Variance}_{f} + \mathbf{Noise}_{t}}$$

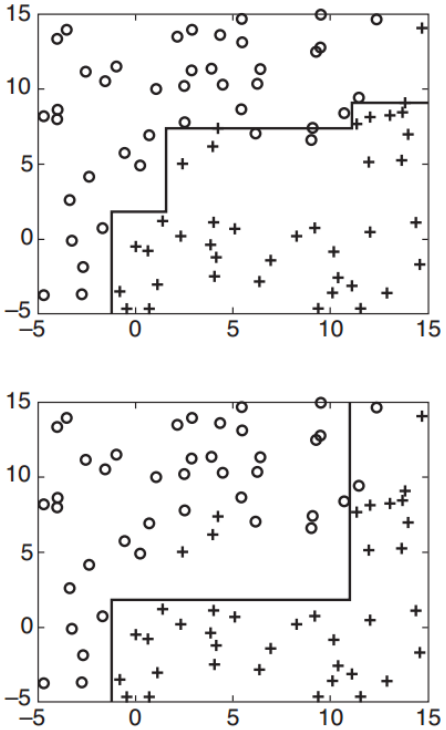
- f refers to the amount of force applied
- θ denotes the angle of the launcher
- t corresponds to the location of the target

Bias-Variance in Classifiers

- The task of predicting a class label can be analyzed using the same approach
- Predictions may turn out to be correct, while others can be way off the mark
- The error of a classifier can be decomposed as a sum of the three terms previously described
- Classifiers minimize the error in the training set but must be able to generalize to unseen instances

Bias-Variance in Classifiers





- T_1 and T_2 are generated from the same training data
- T_2 is obtained by pruning T_1
- These design choices introduce a bias analogous to that of the projectile launcher into the classifier
- The larger the assumptions made about the decision boundaries, the larger the **bias**
  - T_2 has a larger bias

Figure 5.33. Two decision trees with different complexities induced from the same training data.

Bias-Variance in Classifiers

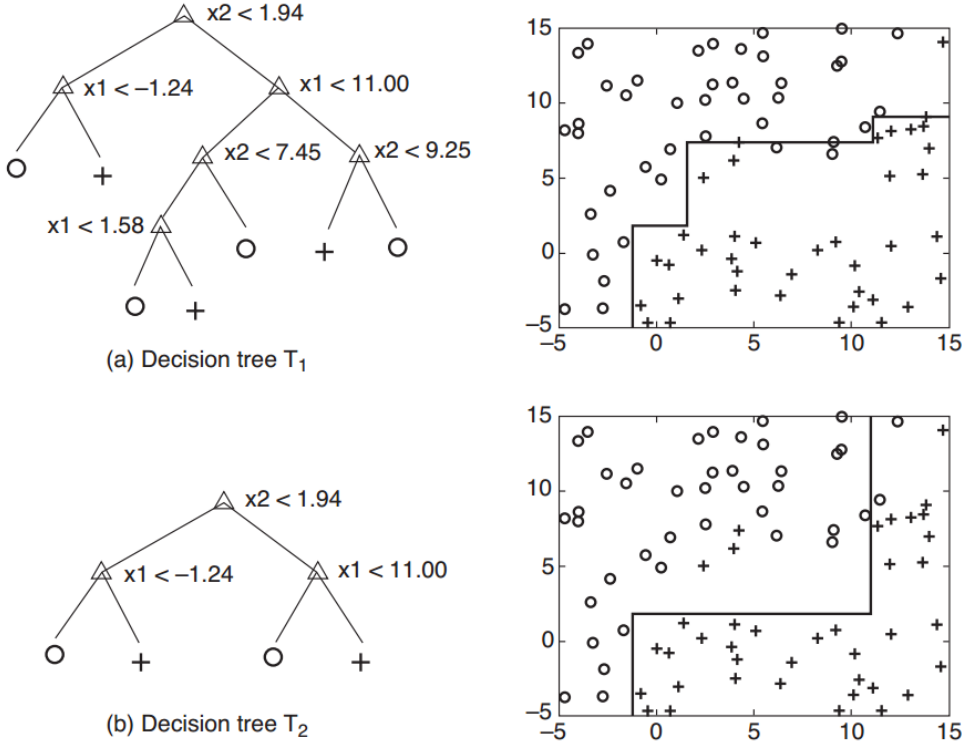
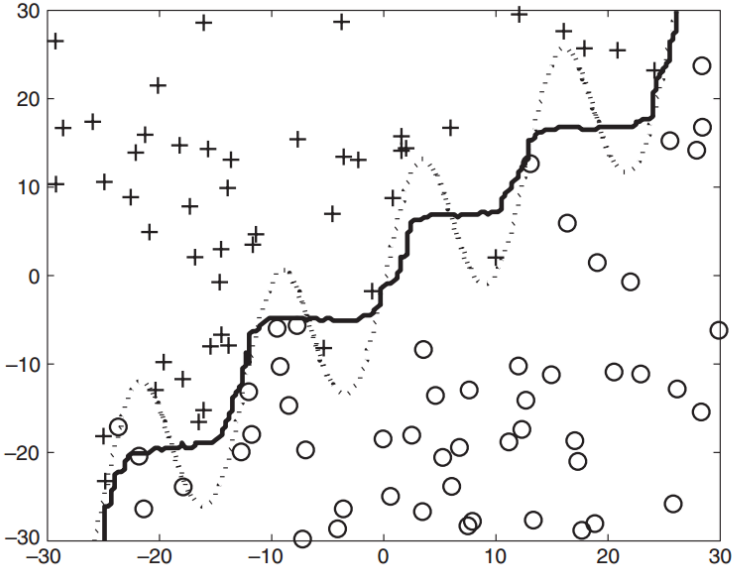


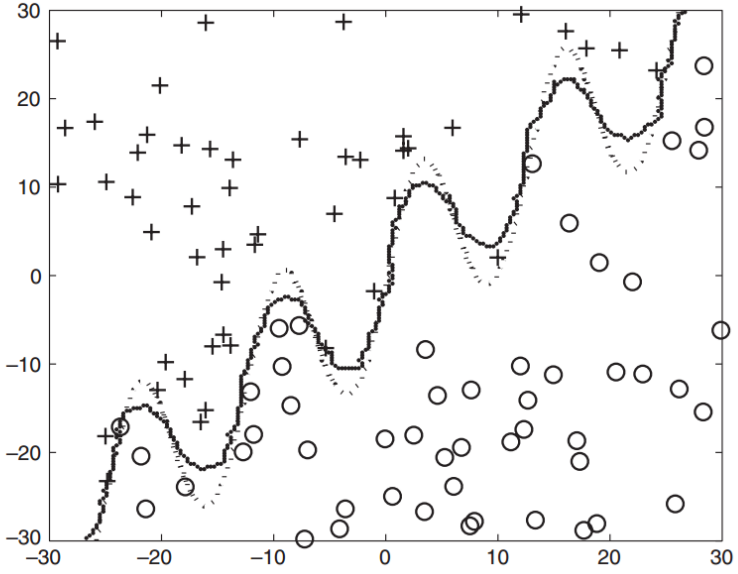
Figure 5.33. Two decision trees with different complexities induced from the same training data.

- The expected error of a classifier can be affected by different compositions of the training set leading to different decision boundaries. This is analogous to the **variance** when different amounts of force are applied to the projectile.
- The third component of the expected error (i.e., **noise**) is associated with the intrinsic noise in the class. That is, some instances with the same attributes may have different classes.

Bias-Variance in Classifiers



(a) Decision boundary for decision tree.



(b) Decision boundary for 1-nearest neighbor.

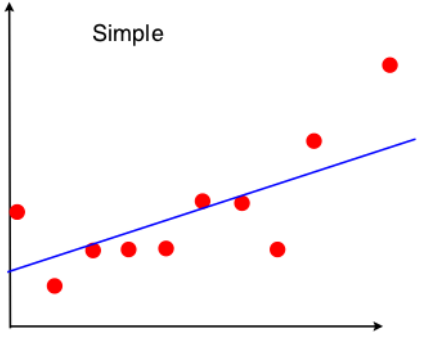
Figure 5.34. Bias of decision tree and 1-nearest neighbor classifiers.

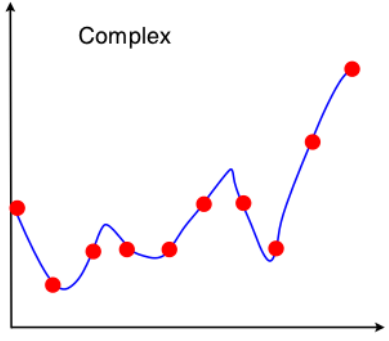
Generalization

- Components of generalization error
  - **Bias:** how much the average model over all training sets differ from the true model?
    - Error due to inaccurate assumptions/simplifications made by the model
  - **Variance:** how much models estimated from different training sets differ from each other
- **Underfitting:** model is too "simple" to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- **Overfitting:** model is too "complex" and fits irrelevant characteristics (noise) in the data
  - Low bias and high variance
  - Low training error and high test error

Slide credit: L. Lazebnik

Bias-Variance Tradeoff





- Models with too few parameters are inaccurate because of a large bias (not enough flexibility).
- Models with too many parameters are inaccurate because of a large variance (too much sensitivity to the sample).

Slide credit: D. Hoiem

Bias-Variance Tradeoff

$$E(MSE) = \text{noise}^2 + \text{bias}^2 + \text{variance}$$

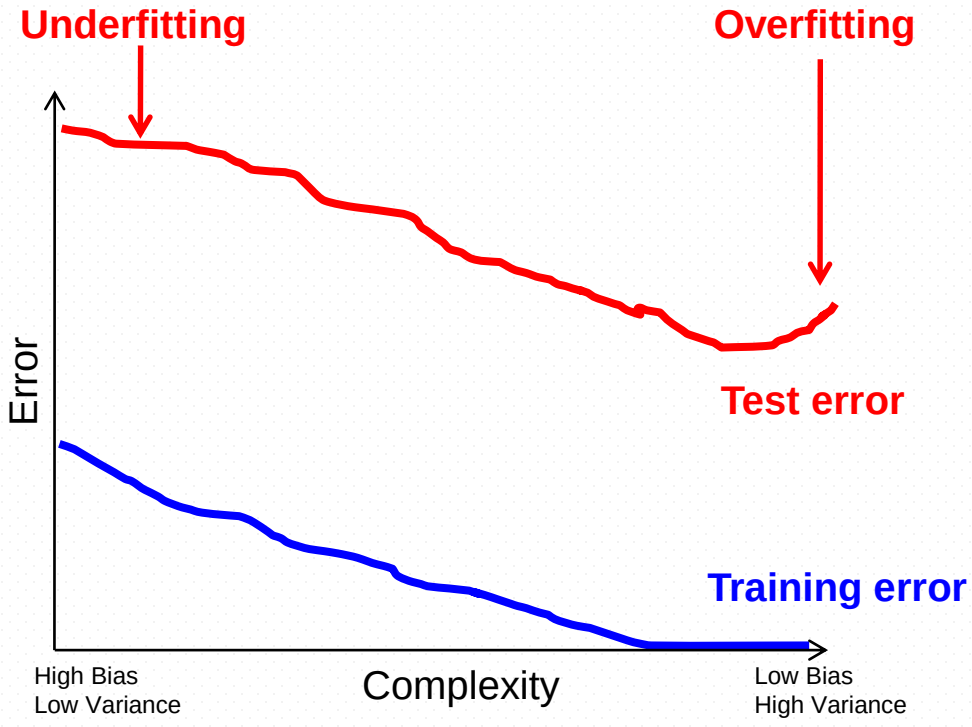
Unavoidable error

Error due to incorrect assumptions

Error due to variance of training samples

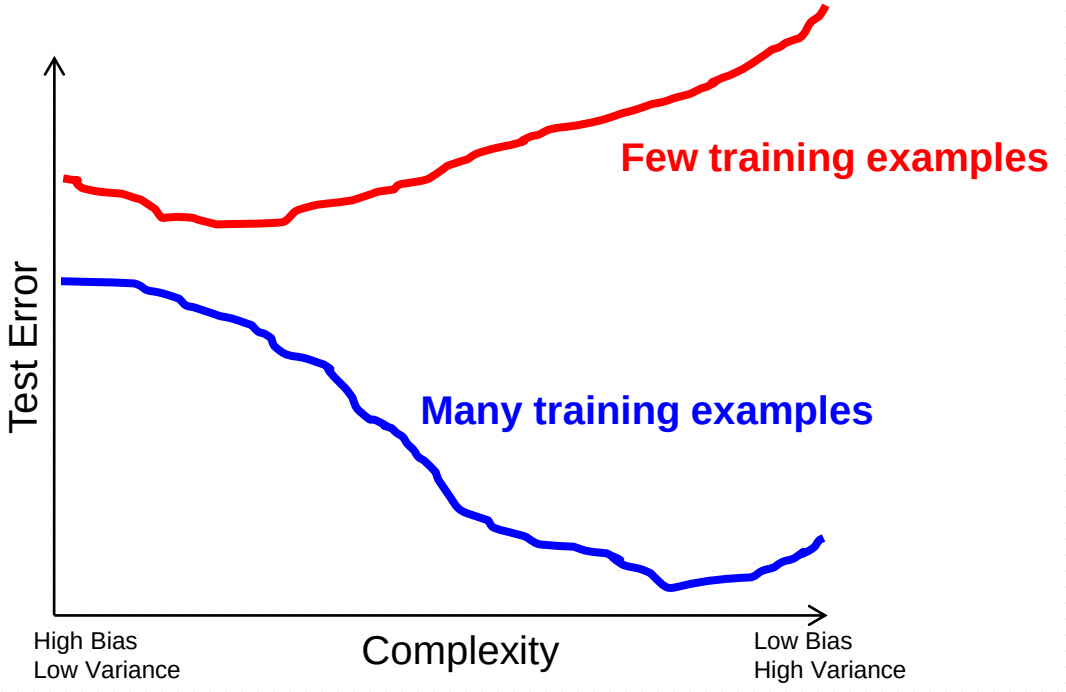
Slide credit: D. Hoiem

Bias-Variance Tradeoff



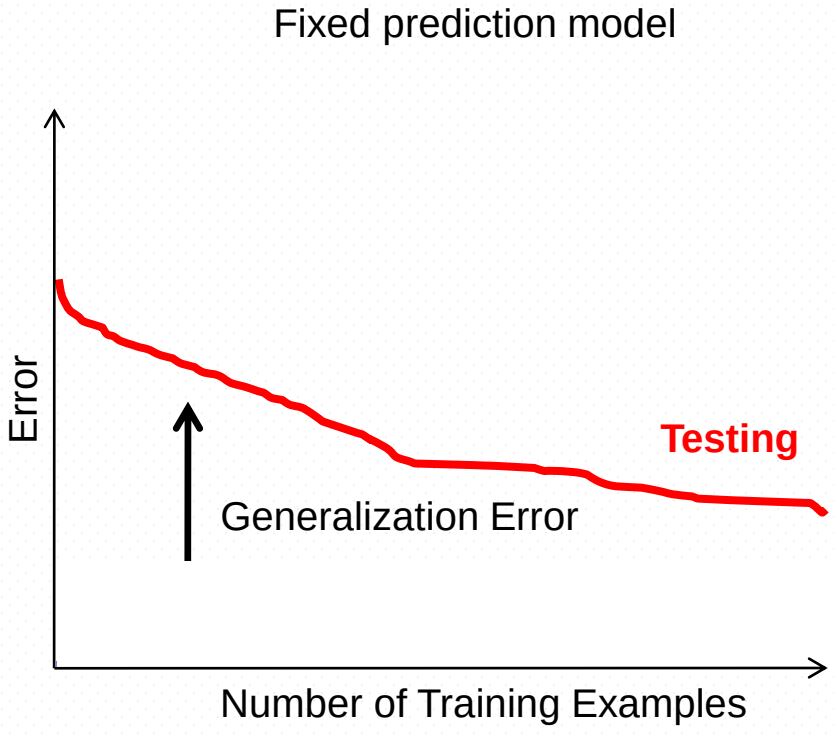
Slide credit: D. Hoiem

Bias-Variance Tradeoff



Slide credit: D. Hoiem

Effect of Training Size



Slide credit: D. Hoiem

Illustration (1)

Low variance, high bias method ⇒ underfitting

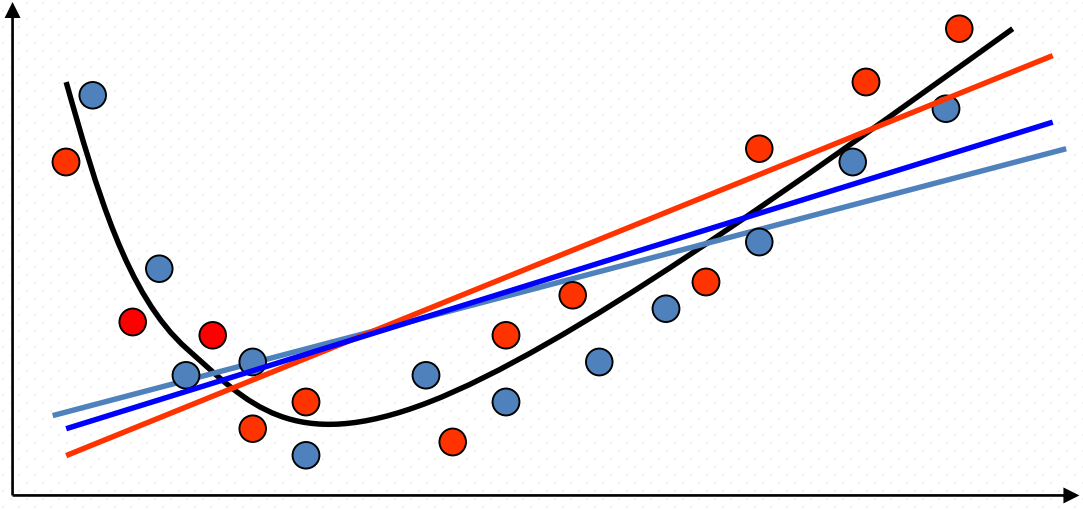


Illustration (2)

Low bias, high variance method ⇒ overfitting

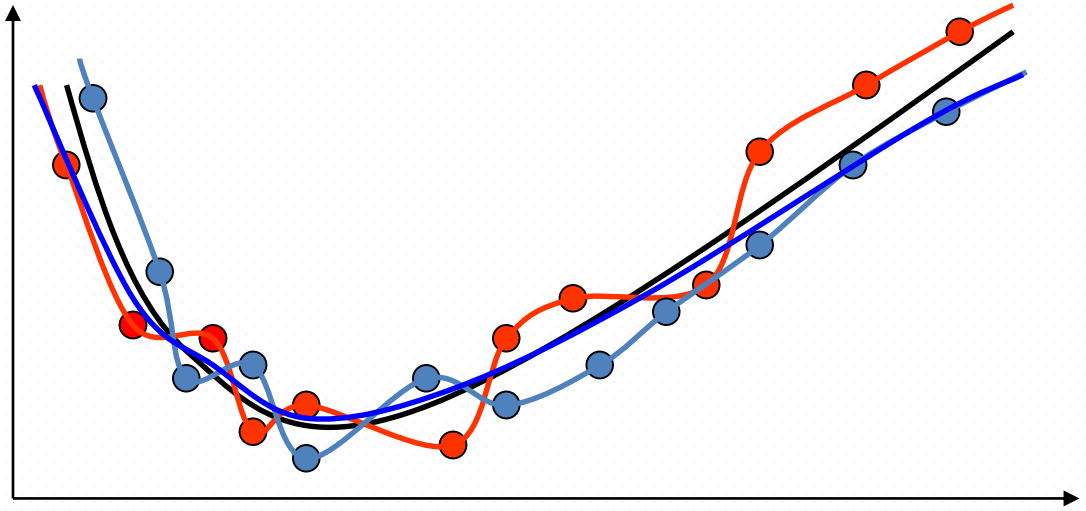
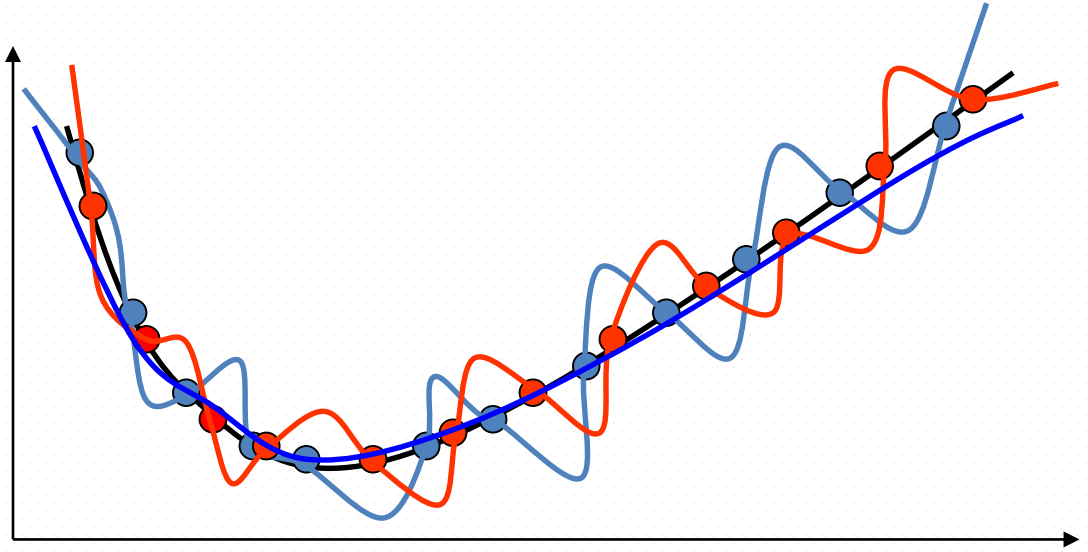


Illustration (3)

No noise doesn't imply no variance (but less variance)



Remember…

- No classifier is inherently better than any other: you need to make assumptions to generalize

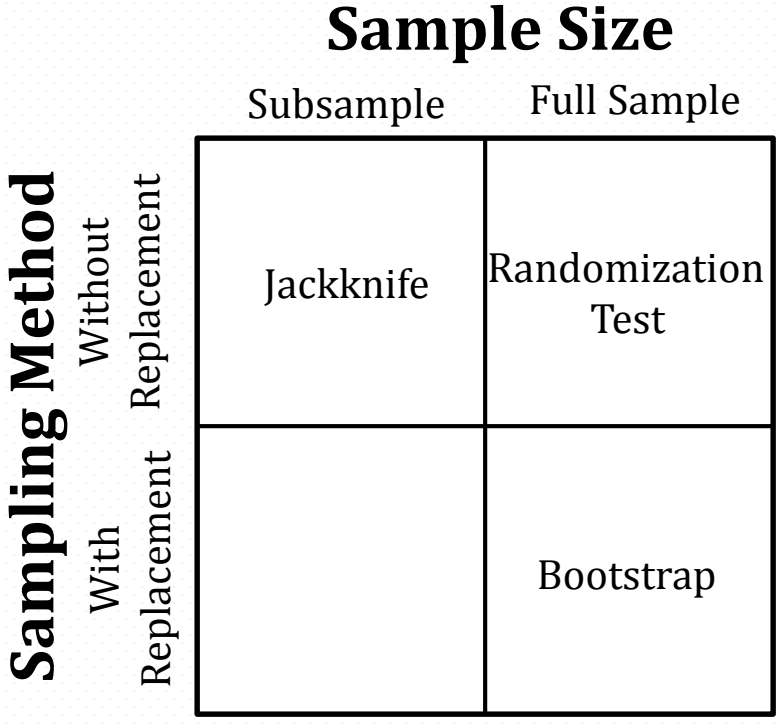
- Three kinds of error
  - **Inherent:** unavoidable
  - **Bias:** due to over-simplifications
  - **Variance:** due to inability to perfectly estimate parameters from limited data

How to Reduce Variance?

- Choose a simpler classifier
- Regularize the parameters
- Get more training data

The Jackknife

- Sampling technique somewhat similar to Bootstrap



Recall the Bootstrap

- The bootstrap uses sampling with replacement to form the training set.
  - Sample a dataset of n instances n times with replacement to form a new dataset of n instances.
  - Use this data as the training set.
  - Use the instances from the original dataset that don't occur in the new training set for testing.

The Jackknife

- Somewhat similar to bootstrap
- For single-elimination jackknife:
  - Create n samples of size $n - 1$
  - The i^{th} instance is eliminated in the i^{th} sample
  - Compute the mean (or wanted quantity) of each sample
- Can be used to estimate/reduce bias

The Jackknife

- Estimating a parameter θ:

$$\bar{\theta}_{\text{Jack}} = \frac{1}{n}\sum_{i=1}^{n}(\bar{\theta}_i)$$

- Estimating variance:

$$Var(\theta) = \sigma^2 = \frac{n-1}{n}\sum_{i=1}^{n}\left(\bar{\theta}_i - \bar{\theta}_{\text{Jack}}\right)^2$$

The Jackknife

- Estimating and correcting bias:

$$\bar{\theta}_{\text{BiasCorrected}} = N\bar{\theta} - (N-1)\bar{\theta}_{\text{Jack}}$$

  - This reduces bias from $O(N^{-1})$ to $O(N^{-2})$

Boostrap aggregating (Bagging)

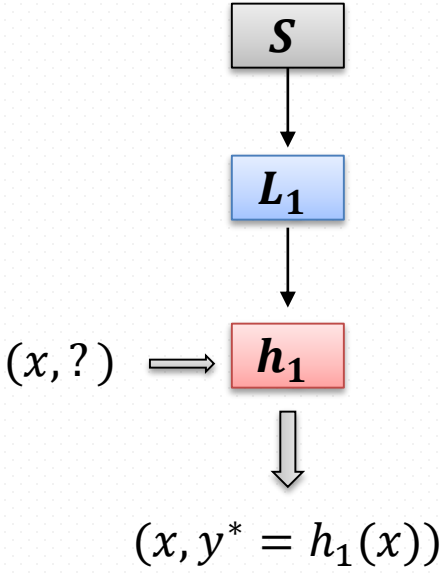
- An ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms
- Can be used in both regression and classification
- Reduces variance and helps to avoid overfitting
- Usually applied to decision trees, though it can be used with any type of method

An Aside: Ensemble Methods

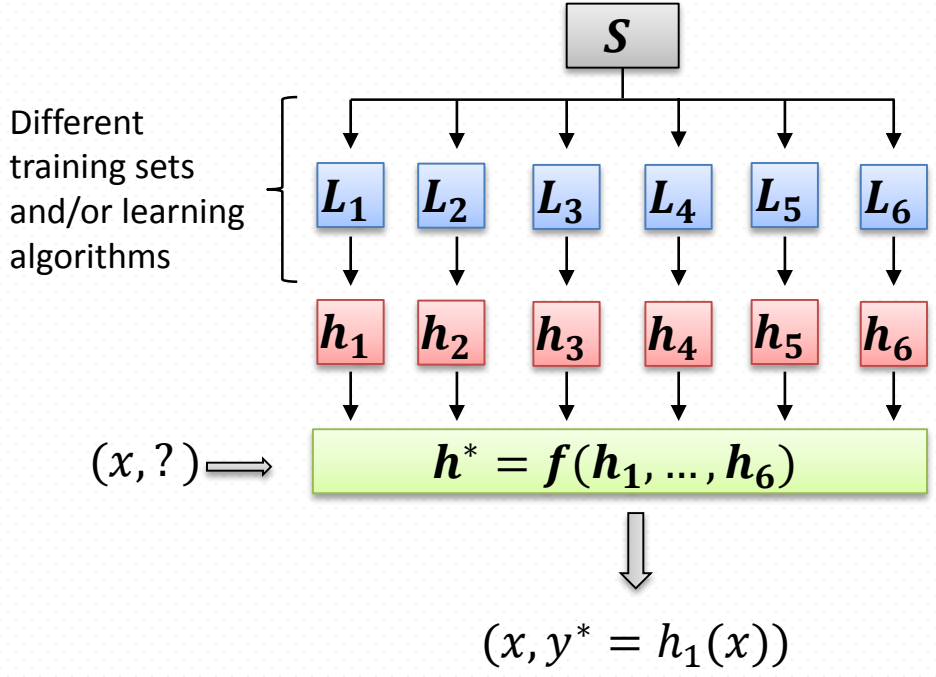
- In a nutshell:
  - A combination of multiple learning algorithms with the goal of achieving better predictive performance than could be obtained from any of these classifiers alone
  - A meta-algorithm that can be considered to be, in itself, a supervised learning algorithm since it produces a single hypothesis
  - Tend to work better when there is diversity among the models
  - Examples:
    - Bagging
    - Boosting
    - Bucket
    - Stacking

An Aside: Ensemble Methods

Traditional:



Ensemble Method:



Back to Bagging

- The idea:
  1. Create N boostrap samples $\{S_1, \dots, S_N\}$ of S as follows:
     - For each S_i, randomly draw $|S|$ examples from S with replacement
  2. For each $i = 1, \dots, N$

     $h_i = \text{Learn}(S_i)$
  1. Output $H =< \{h_1, \dots, h_N\}, majorityVote >$

Most notable benefits

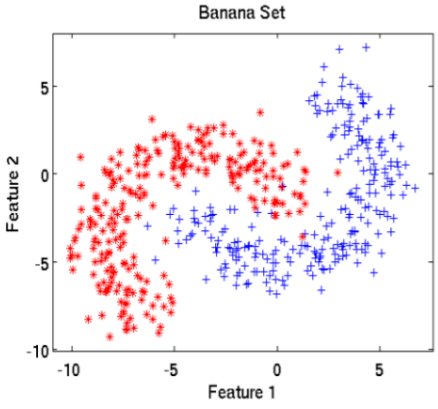
1. Surprisingly competitive performance & rarely overfits
2. Is capable of reducing variance of constituent models
3. Improves ability to ignore irrelevant features

Remember:

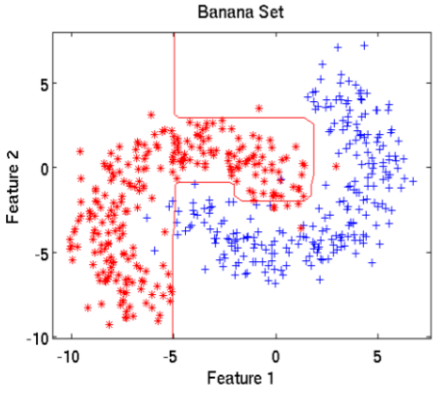
$$\boldsymbol{error}(\boldsymbol{x}) = \mathbf{noise}(\mathbf{x}) + \mathbf{bias}(\mathbf{x}) + \mathbf{variance}(\mathbf{x})$$

Variance: how much does prediction change if we change the training set?

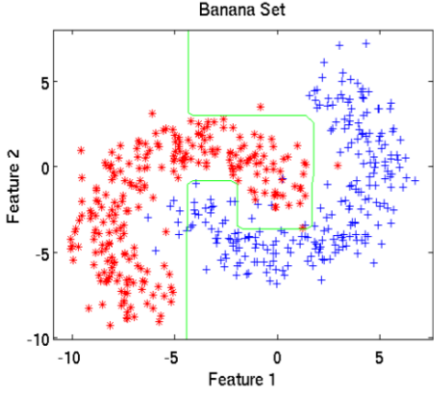
Bagging Example 1



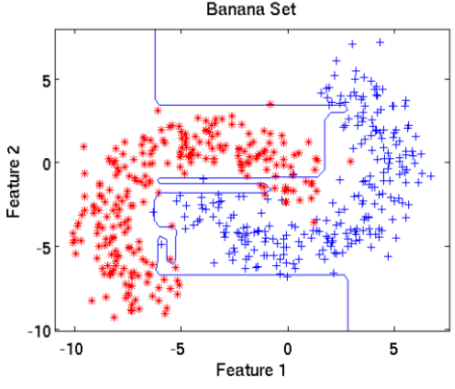
Training data



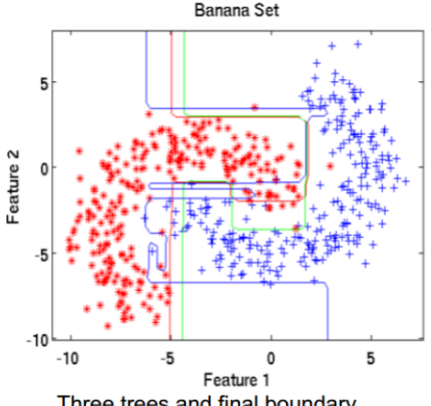
Decision boundary produced by one tree



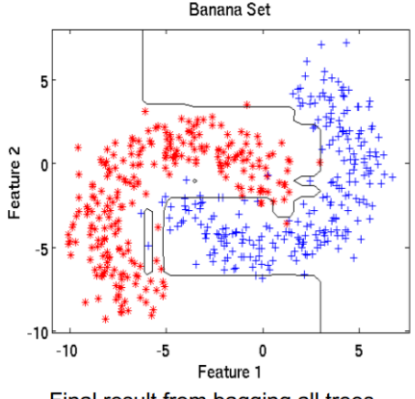
Decision boundary produced by a second tree



Decision boundary produced by a third tree

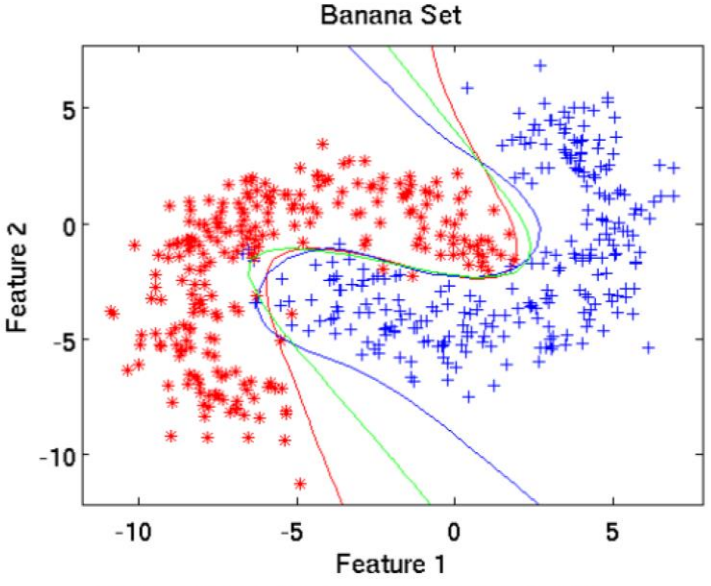


Three trees and final boundary overlaid

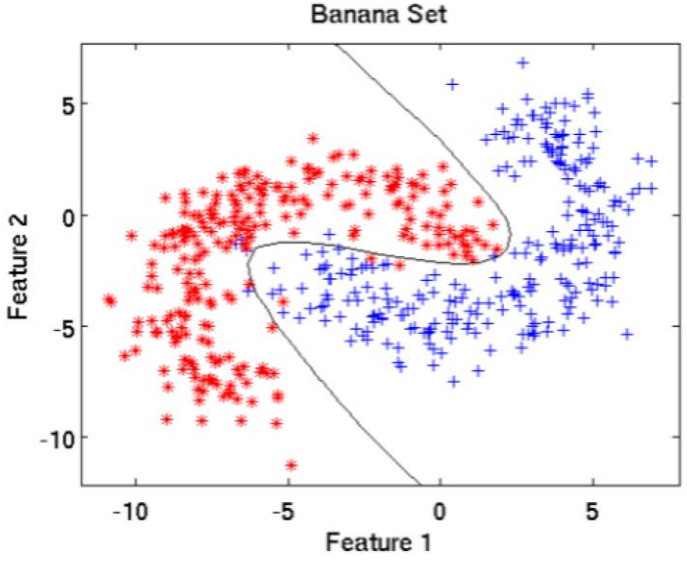


Final result from bagging all trees.

Bagging Example 2



Three neural nets generated with default settings [bpxnc]



Final output from bagging 10 neural nets

Bagging: Neural Net

Bagging Example 3 (1)

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9	x <= 0.35 ==> y = 1
y	1	1	1	1	-1	-1	-1	-1	1	1	x > 0.35 ==> y = -1

Bagging Round 2:

x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1	x <= 0.65 ==> y = 1
y	1	1	1	-1	-1	1	1	1	1	1	x > 0.65 ==> y = 1

Bagging Round 3:

x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9	x <= 0.35 ==> y = 1
y	1	1	1	-1	-1	-1	-1	-1	1	1	x > 0.35 ==> y = -1

Bagging Round 4:

x	0.1	0.1	0.2	0.4	0.4	0.5	0.5	0.7	0.8	0.9	x <= 0.3 ==> y = 1
y	1	1	1	-1	-1	-1	-1	-1	1	1	x > 0.3 ==> y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1	x <= 0.35 ==> y = 1
y	1	1	1	-1	-1	-1	-1	1	1	1	x > 0.35 ==> y = -1

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1	x <= 0.75 ==> y = -1
y	1	-1	-1	-1	-1	-1	-1	1	1	1	x > 0.75 ==> y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1	x <= 0.75 ==> y = -1
y	1	-1	-1	-1	-1	1	1	1	1	1	x > 0.75 ==> y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1	x <= 0.75 ==> y = -1
y	1	1	-1	-1	-1	-1	-1	1	1	1	x > 0.75 ==> y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1	x <= 0.75 ==> y = -1
y	1	1	-1	-1	-1	-1	-1	1	1	1	x > 0.75 ==> y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9	x <= 0.05 ==> y = -1
y	1	1	1	1	1	1	1	1	1	1	x > 0.05 ==> y = 1

Figure 5.35. Example of bagging.

Bagging Example 3 (2)

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Sign	1	1	1	-1	-1	-1	-1	1	1	1
True Class	1	1	1	-1	-1	-1	-1	1	1	1

Figure 5.36. Example of combining classifiers constructed using the bagging approach.

Accuracy: 100%

How does bagging minimize error?

- Ensemble reduces the overall variance
- Let $f(x)$ be the target value of x, h_1 to h_n be the set of base hypothesis, and h_{avg} be the prediction of the base hypotheses
- $Error(h, x) = \big(f(x) - h(x)\big)^2$ → Squared error
- Is there any relation between h_{avg} and variance?
  - Yes

How does bagging minimize error?

- $Error(h, x) = \big(f(x) - h(x)\big)^2$
- $Error\big(h_{avg}, x\big) = \frac{\sum_1^n Error(h_i, x)}{n} = \frac{\sum_1^n \big(h_i(x) - h_{avg}(x)\big)^2}{n}$
- By the above, we see that the squared error of the average hypothesis equals the average squared error of the base hypotheses minus the variance of the base hypotheses

Stability of Learn

- A learning algorithm is **unstable** if small changes in the training data can produce large changes in the output hypothesis (otherwise stable)
- Clearly bagging will have little benefit when used with stable base learning algorithms (i.e., most ensemble members will be very similar)
- Bagging generally works best when used with unstable yet relatively accurate base learners

Bagging Summary

- Works well if the base classifiers are unstable (complement each other)
- Increased accuracy because it ***reduces the variance*** of the individual classifier
- Does not focus on any particular instance of the training data
  - Therefore, less susceptible to model over-fitting when applied to noisy data

Boosting

- Key differences with respect to bagging:
  - It is iterative:
    - **Bagging:** Each individual classifier is independent
    - **Boosting:**
      - Looks at the errors from previous classifiers to decide what to focus on for the next iteration
      - Successive classifiers depend on their predecessors
      - Key idea: place more weight on “hard” examples (i.e., instances that were misclassified on previous iterations)

Historical Notes

- The idea of boosting began with a learning theory question first asked in the late 80's
- The question was answered in 1989 by Robert Shapire resulting in the first theoretical boosting algorithm
- Shapire and Freund later developed a practical boosting algorithm called Adaboost
- Many empirical studies show that Adaboost is highly effective(very often they outperform ensembles produced by bagging)

Boosting

- An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of a boosting round
  - Different implementations vary in terms of (1) how the weights of the training examples are updated and (2) how the predictions are combined

Boosting

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Boosting

- Equal weights are assigned to each training instance (1/N for round 1) at first round
- After a classifier $\boldsymbol{C_i}$ is learned, the weights are adjusted to allow the subsequent classifier $\boldsymbol{C_{i+1}}$ to "pay more attention" to data that were misclassified by $\boldsymbol{C_i}$.
- Final boosted classifier $\boldsymbol{C^*}$ combines the votes of each individual classifier
  - Weight of each classifier's vote is a function of its accuracy
- **Adaboost** – popular boosting algorithm

Adaboost (Adaptive Boost)

- Input:
  - Training set D containing ***N*** instances
  - *T* rounds
  - A classification learning scheme
- Output:
  - A composite model

Adaboost: Training Phase

- Training data D contain N labeled data :
  - $(X_1, y_1), (X_2, y_2), (X_3, y_3), \ldots . (X_N, y_N)$
- Initially assign equal weight $1/N$ to each data
- To generate T base classifiers, we need T rounds or iterations
- Round i:
  - data from D are sampled with replacement , to form D_i (size N)
- Each data's chance of being selected in the next rounds depends on its weight
  - Each time the new sample is generated directly from the training data D with different sampling probability according to the weights; these weights are not zero

Adaboost: Training Phase

- Base classifier C_i, is derived from training data of D_i
- Error of C_i is tested using D_i
- Weights of training data are adjusted depending on how they were classified
  - Correctly classified: Decrease weight
  - Incorrectly classified: Increase weight
- Weight of a data indicates how hard it is to classify it (directly proportional)

Adaboost: Testing Phase

- The lower a classifier error rate, the more accurate it is, and therefore, the higher its weight for voting should be
- Weight of a classifier C_i's vote is

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$

- Testing:
  - For each class c, sum the weights of each classifier that assigned class c to X (unseen data)
  - The class with the highest sum is the WINNER!

$$C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

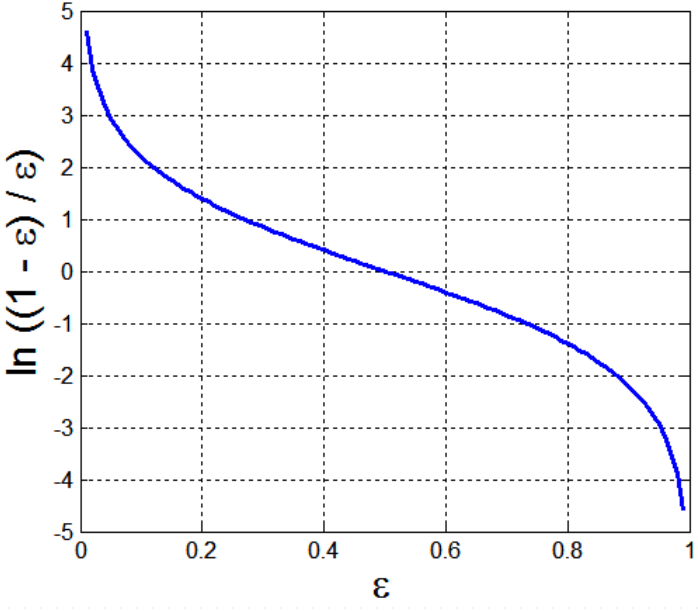
Example: Error and Classifier Weight in AdaBoost

- Base classifiers: $C_1, C_2, \ldots, C_T$
- Error rate:
  - i = index of classifier
  - j=index of instance

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



Example: Data Instance Weight in AdaBoost

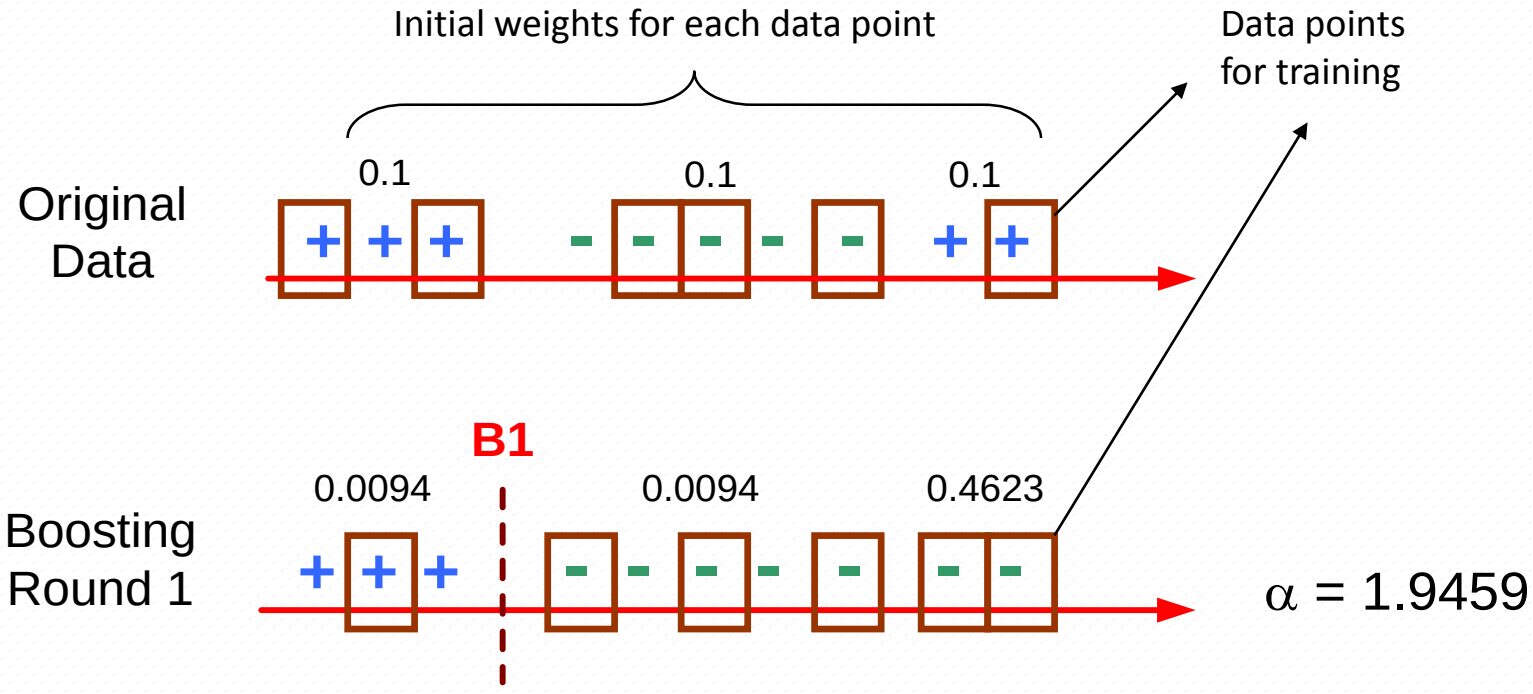
- Assume: N training data in D, T rounds, (x_j, y_j) are the training data, C_i, a_i are the classifier and weight of the i^{th} round, respectively
- Weight update on all training data in D:

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \begin{cases} \exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ \exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

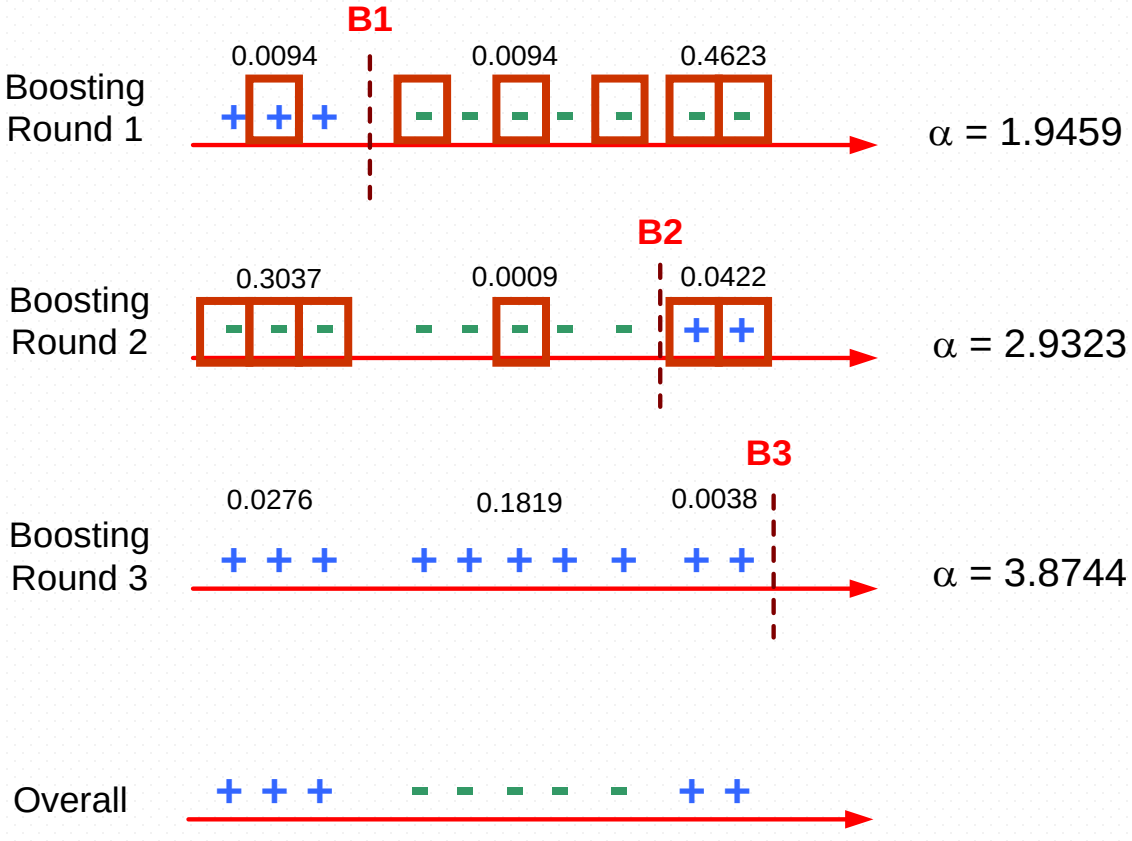
where Z_i is the normalization factor

$$C^*(x_{test}) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta\left(C_i(x_{test}) = y\right)$$

Illustrating AdaBoost



Illustrating AdaBoost



Bagging and Boosting Summary

- **Bagging:**
  - Resample data points
  - Weight of each classifier is the same
  - Only variance reduction
  - Robust to noise and outliers
- **Boosting:**
  - Reweight data points (modify data distribution)
  - Weight of classifier vary depending on accuracy
  - Reduces both bias and variance
  - Can hurt performance with noise and outliers

Ensemble Methods

- An ensemble is a set of classifiers that learn a target function, and their individual predictions are combined to classify new examples.
- Ensembles generally improve the generalization performance of a set of classifiers on a domain.

Typical Ensemble Procedure

General procedure for ensemble method.

Let D denote the original training data, k denote the number of base classifiers, and T be the test data.

for $i = 1$ to k **do**

Create training set D_i from D.

Build a base classifier C_i from D.

end for

for each test record $x \in T$ **do**

$C^*(x) = Vote\big(C_1(\mathbf{x}), C_2(\mathbf{x}), \dots, C_k(\mathbf{x})\big)$

end for

Rationale for Ensemble Methods

- Statistical reasons
  - A set of classifiers with similar training performances may have different generalization performances.
  - Combining outputs of several classifiers *reduces the risk of selecting a poorly performing classifier*.

Rationale for Ensemble Methods

- Large volumes of data
  - If the amount of data to be analyzed is too large, a single classifier may not be able to handle it; train different classifiers on *different partitions of data*.
- Too little data
  - Ensemble systems can also be used when there is too little data; *resampling techniques*.

Rationale for Ensemble Methods

- Divide-and-conquer

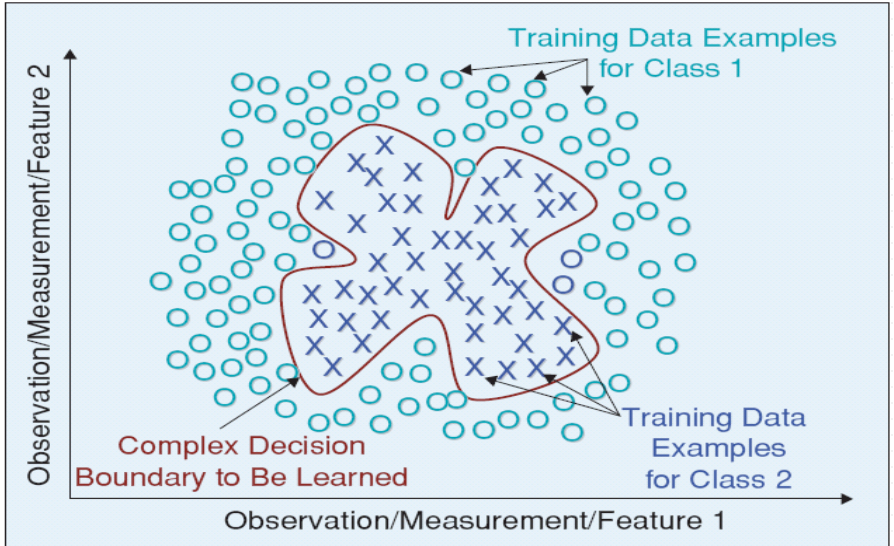


Figure 1. Complex decision boundary that cannot be learned by linear or circular classifiers.

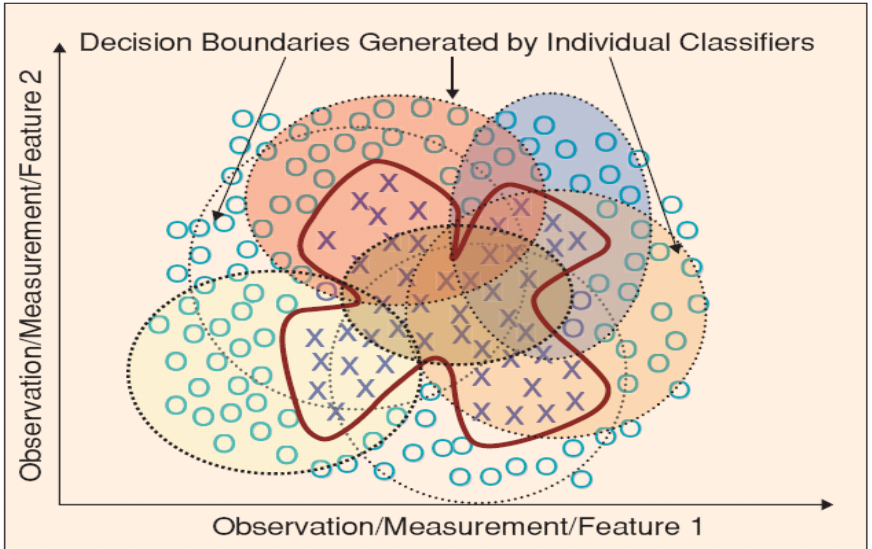


Figure 2. Ensemble of classifiers spanning the decision space.

Why Ensemble Methods Work

Consider an ensemble of twenty-five binary classifiers, each of which has an error rate of $\epsilon = 0.35$. If the base classifiers are independent—i.e., their errors are uncorrelated—then the error rate of the ensemble classifier is

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

which is considerably lower than the base classifiers.

Requirements for Ensembles

There are two necessary and sufficient conditions for an ensemble of classifiers to be more accurate than any of its individual members. Namely:

1. The classifiers must be accurate.
2. The classifiers must be diverse.

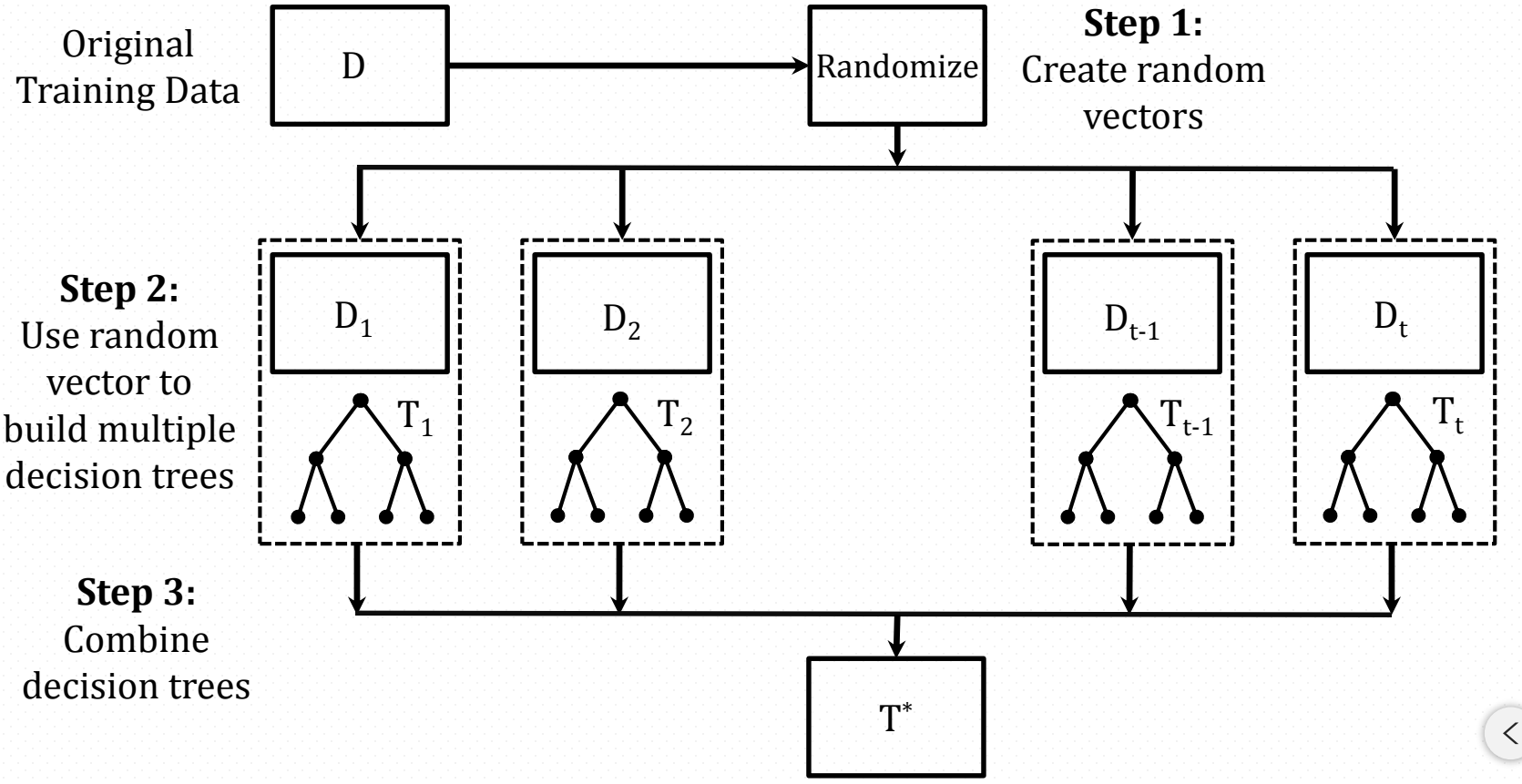
Methods for Constructing Ensembles

1. **By manipulating the training set.** Create multiple training sets by resampling the original data according to some sampling distribution.
2. **By manipulating the input features.** Choose a subset of input features to form each training set.
3. **By manipulating the class labels.** Transform the training data into a binary class problem by randomly partitioning the class labels into two disjoint subsets.
4. **By manipulating the learning algorithm.** Manipulate the learning algorithm to generate different models.

Random Forests Method

- A class of ensemble methods specifically designed for decision tree classifiers.
- Combines predictions made by many decision trees.
- Each tree is generated based on a bootstrap sample and the values of an independent set of random vectors.
- The random vectors are generated from a fixed probability distribution.

Random Forests: A Visual Explanation



Randomizing Random Forests

Each decision tree uses a random vector that is generated from some fixed probability distribution. This randomness can be incorporated in many ways:

1. Randomly select F input features to split at each node (Forest-RI).
2. Create linear combinations of the input features to split at each node (Forest-RC).
3. Randomly select one of the F best splits at each node.

Why use Random Vectors?

- Recall that an underlying assumption of the ensemble process is that the base learners are *independent*.
- As the trees become more correlated (less independent), the generalization error bound tends to increase.
- Randomization helps to reduce the correlation among decision trees.

Random Forests Error Bounds

The upper bound for generalization error of random forests converges to the following expression, when the number of trees is sufficiently large

$$\text{Generalization error} \leq \frac{\bar{\rho}\left(1 - s^2\right)}{s^2},$$

where $\bar{\rho}$ is the average correlation among the trees and s is a quantity the measures the "strength" (or average performance) of the tree classifiers.

Out of Bag Error

- Assume a method for constructing a classifier from any training set. Given a specific training set T, form bootstrap training sets T_k, construct classifiers, and let these vote to form the bagged predictor. For each $y, \mathbf{x}$ in the training set, aggregate the votes only over those classifiers for which T_k does not contain $y, \mathbf{x}$. Call this the out-of-bag classifier.

Using Random Features

- Random split selection does better than bagging; introduction of random noise into the outputs also does better.
- To improve accuracy, the randomness injected has to minimize the correlation $\bar{\rho}$ while maintaining strength.

Random Forests Advantages

- Empirically comparable classification accuracies to AdaBoost.
- More robust to outliers and noise than AdaBoost.
- Runs much faster than AdaBoost.
- Produces useful internal estimates of error, strength, correlation, and variable importance.
- Simple and easily parallelized.

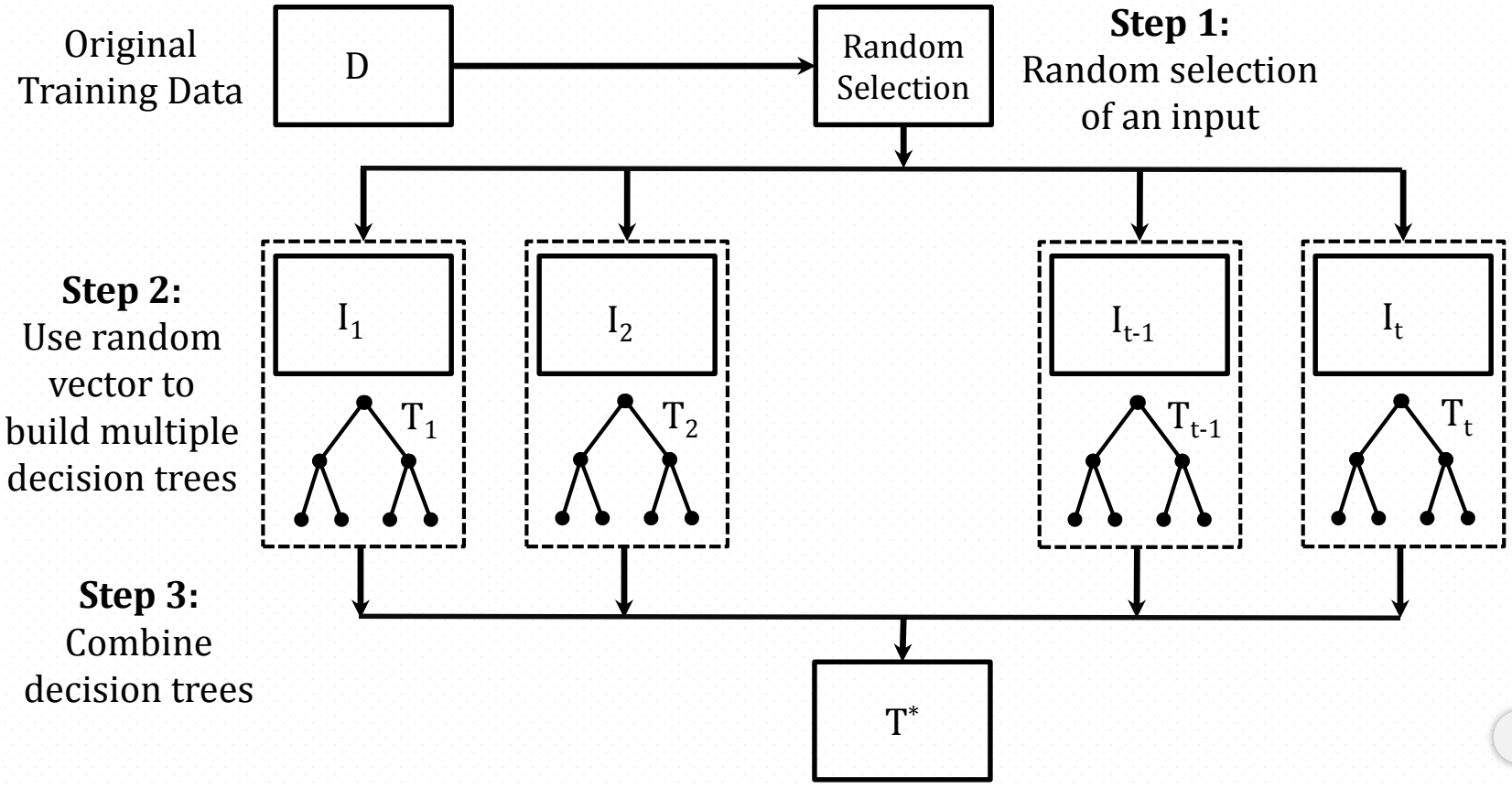
Random Subspace Method

- An ensemble classifier that consists of several classifiers trained on the same dataset using only a (random) subset of the available features.
  - A generalization of the random forest algorithm.
  - Can be composed from any underlying classifiers.

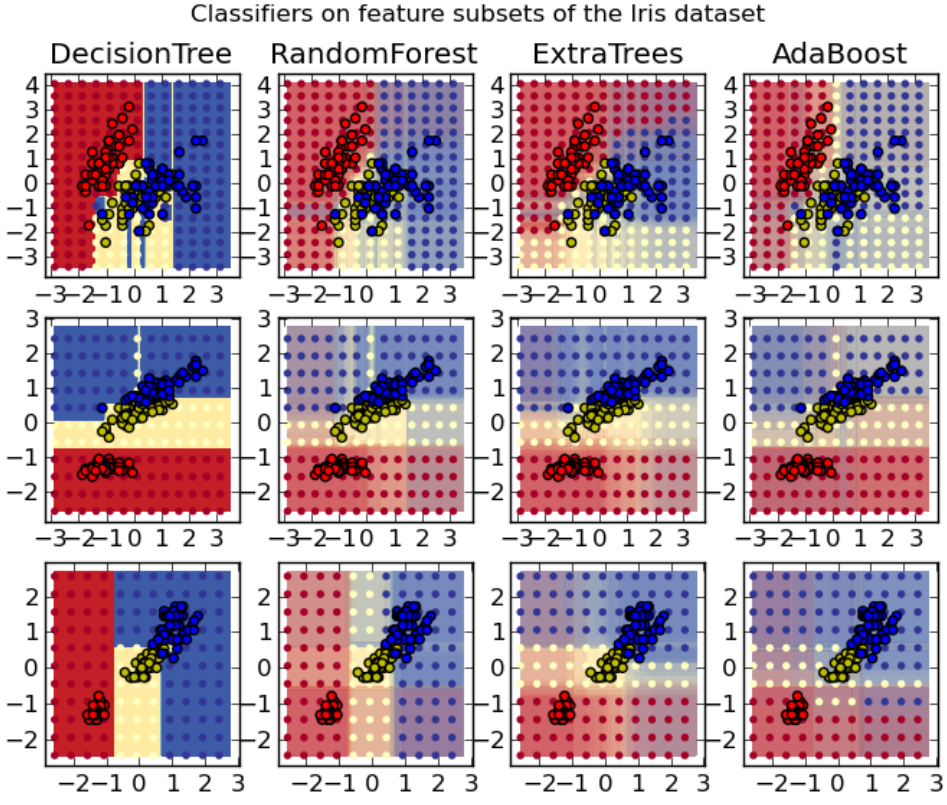
Extra-Trees Method

- **Extremely randomized trees (extra-trees)** are another class of ensemble methods specifically designed for decision tree classifiers.
- Each tree is built from the original learning sample.
- At each test node, the best split is determined among K random splits, and each one is determined by a random selection of an input (without replacement) and a threshold.

Extra-Trees: A Visual Explanation



Ensembles of Trees: A Comparison



Summarizing Ensembles of Trees

- **Random forests** can be thought of as combining bagging and random subspaces.
  - Bootstrap aggregated (bagged) trees.
  - Random subset of the input space (random subspaces).
- **Extra-trees** combine classifiers without bagging by using random splits to generate different trees.
- Decision tree ensembles are **effective because decision trees are diverse** (display high variance).

Ensemble Methods

- An ensemble is a set of classifiers that learn a target function, and their individual predictions are combined to classify new examples.
- Covered so far:
  - Bagging
  - Boosting
  - Random Forests
  - Extra Trees
- Today:
  - Stacking
  - Blending

Stacked Generalization (Stacking)

- **Underlying idea:** *Learn whether training data have been properly learned*
- 2 tiers of classifiers
  - If a particular base-classifier incorrectly learned a certain region of the feature space, the second tier (meta) classifier may be able to detect this undesired behavior
  - Along with the learned behaviors of other classifiers, it can correct such improper training

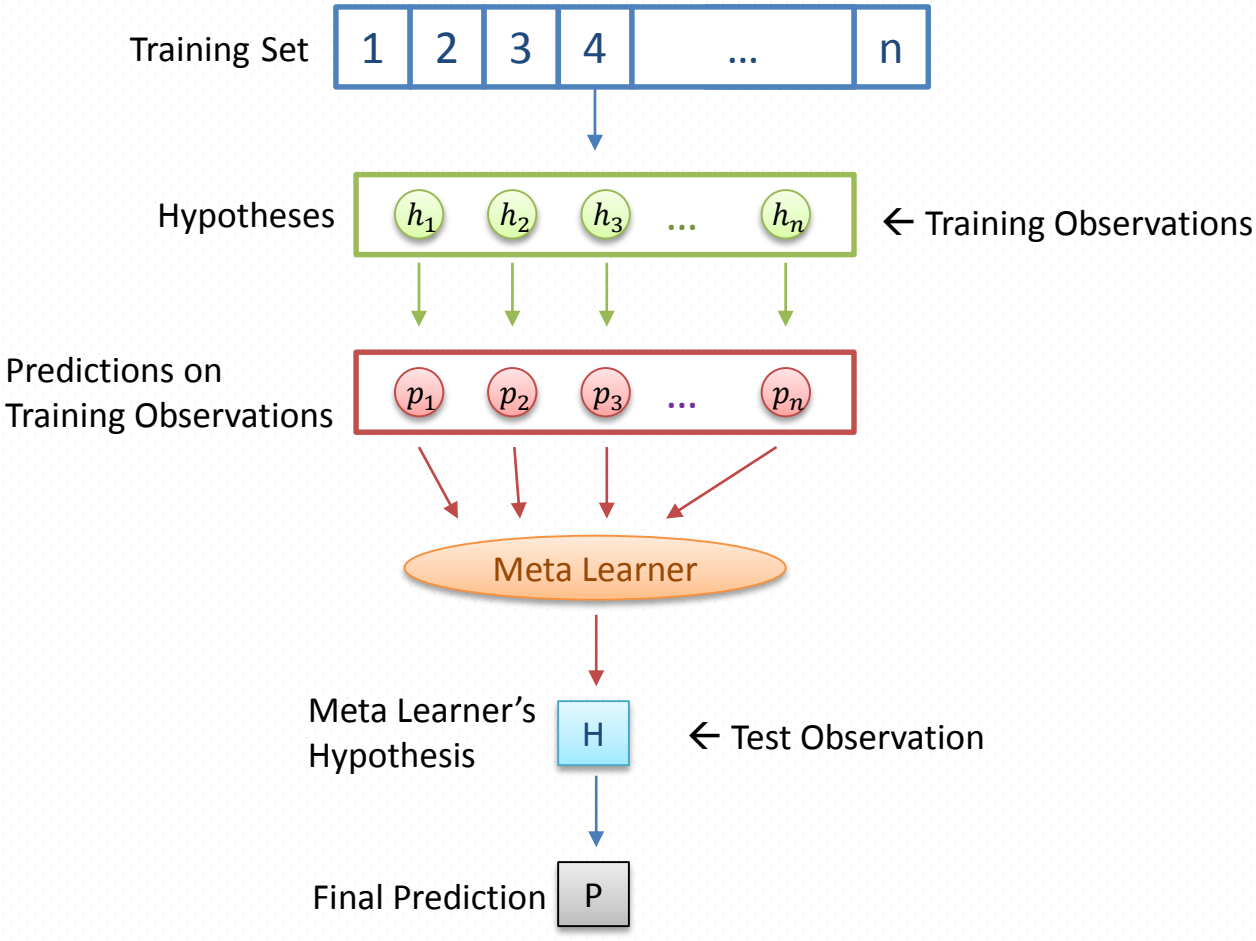
The Stacking Framework

- There are two approaches for combining models: **voting** and **stacking**
- Difference between the stacking framework and those previously discussed:
  - In contrast to stacking, no learning takes place at the meta-level when combining classifiers by a voting scheme
  - Label that is most often assigned to a particular instance is chosen as the correct prediction when using voting

The Stacking Framework

- Stacking is concerned with combining multiple classifiers generated by different learning algorithms $L_1, \ldots, L_N$ on a single dataset S, which is composed by a feature vector $s_i = (x_i, y_i)$.
- The stacking process can be broken into two phases:
  1. Generate a set of base-level classifiers $C_1, \ldots, C_N$
     - Where $C_i = L_i(S)$
  2. Train a meta-level classifier to combine the outputs of the base-level classifiers

The Stacking Framework



The Stacking Framework

- The training set for the meta-level classifier is generated through a leave-one-out cross validation process.

$$\forall \boldsymbol{i} = \mathbf{1}, \ldots \boldsymbol{n} \textbf{ and } \forall \boldsymbol{k} = \mathbf{1}, \ldots, \boldsymbol{N}: \boldsymbol{C_k^i} = \boldsymbol{L_k}(\boldsymbol{S} - \boldsymbol{s_i})$$

- The learned classifiers are then used to generate predictions for $\boldsymbol{s_i}: \hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\boldsymbol{k}} = \boldsymbol{C_k^i}(\boldsymbol{x_i})$
- The meta-level dataset consists of examples of the form $\left(\left(\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\mathbf{1}}, \ldots, \hat{\boldsymbol{y}}_{\boldsymbol{i}}^{\boldsymbol{n}}\right), \boldsymbol{y_i}\right)$, where the features are the predictions of the base-level classifiers and the class is the correct class of the example in hand.

Stacking with Probability Distributions

- Proposed by Ting and Witten (1999)
- The idea:
  - Stack base-level classifiers output probability distributions (PDs) over the set of class values, instead of single class values – *somewhat similar to logistic regression*
  - The meta-level classifiers can then use these for training.
  - PDs represent the confidence of the base-level classifiers, and hence the authors argue that using this information is more appropriate

Stacking with Probability Distributions

- The prediction of a base-level classifier C applied to x:

$$p^C(x) = (p^C(c_1|x), p^C(c_2|x), \ldots, p^C(c_n|x))$$

  - $\{c_1, c_2, .., c_n\}$ is the set of possible class values
  - $p^C(c_i|x)$ denotes the probability that example x belongs to class c_i as estimated by classifier C
- The class c_j with highest probability is predicted by classifier C.

Stacking with Probability Distributions

- Multi-response linear regression (MLR) is recommended for meta-level learning (Ting & Witten, 1999)
  - For a classification problem with m class values $\{c_1, \dots, c_m\}$, m regression problems are formulated
    - For each class c_j, a linear equation LR_j is constructed to predict a binary value (1: class value is c_j, 0: otherwise)
  - Given a new x, $LR_j(x)$ is calculated for all j and the class k is predicted with maximum $LR_k(x)$

Mathematical Insight Into Stacking

- If an ensemble has M base models having an error rate $\boldsymbol{e < 1/2}$ and if the errors are independent, then the probability that the ensemble makes an error is the probability that more than $M/2$ base models misclassify that instance
- In essence, the meta-classifier is trained to learn the error of the base classifiers
- Adding the estimated errors to the output of the base classifiers can improve prediction

The Stacking Algorithm

Input: Data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$;
First-level learning algorithms $\mathcal{L}_1, \cdots, \mathcal{L}_T$;
Second-level learning algorithm $\mathcal{L}$.

Process:

for $t = 1, \cdots, T$:
  $h_t = \mathcal{L}_t(\mathcal{D})$ % Train a first-level individual learner h_t by applying the first-level
end; % learning algorithm $\mathcal{L}_t$ to the original data set $\mathcal{D}$
$\mathcal{D}' = \emptyset$; % Generate a new data set
for $i = 1, \cdots, m$:
  for $t = 1, \cdots, T$:
    $z_{it} = h_t(\boldsymbol{x}_i)$ % Use h_t to classify the training example $\boldsymbol{x}_i$
  end;
  $\mathcal{D}' = \mathcal{D}' \cup \{((z_{i1}, z_{i2}, \cdots, z_{iT}), y_i)\}$
end;
$h' = \mathcal{L}(\mathcal{D}')$. % Train the second-level learner h' by applying the second-level
% learning algorithm $\mathcal{L}$ to the new data set $\mathcal{D}'$

Output: $H(\boldsymbol{x}) = h'(h_1(\boldsymbol{x}), \cdots, h_T(\boldsymbol{x}))$

A few more thoughts on ensembles

- There are two basic ways of aggregating classification methods:
  - **After the fact:** Combines existing solutions
    - Ex.: Netflix challenge teams merging / "blending"
  - **Before the fact:** Creates solutions to be combined
    - Ex.: Bagging
- Each of these methods can be more/less appropriate to particular problems

Before-the-fact aggregation

- Suppose we want to create a robust method to determine if a given image contains a face
- **Problem:** It may be very difficult and computationally intensive to create a classifier for that task
- **One idea:** detect eyes instead of faces
  - **Advantage:** A lot more efficient
  - **Problem #2:** this may produce models with low accuracy
  - **Solution:** Combine it with other similar classifiers

Before-the-fact aggregation – Detecting Faces

- Eyes detector
- Mouth detector
- Ears detector
- Skin detector

Meta Classifier

High accuracy!

Low individual accuracy
Computationally efficient

After-the-fact aggregation

- Netflix challenge teams merging
- Different models are already built. Find an intelligent way to combine them
- Given each hypothesis $h_i, \dots, h_N$ and a new instance x, we can use the previously discussed regression to compute the prediction $g(x)$

$$h_1, h_2, \dots, h_N \rightarrow g(x) = \sum_{t=1}^{T} \boldsymbol{\alpha_t}\, h_t(x)$$

- Choose $\boldsymbol{\alpha_t}$'s as to minimize the error on ***aggregation set***
  - *Can be done using min-squared-error*

After-the-fact aggregation

- One highly ranked team proposed

  Merge with us by giving us your solution. We will split the winnings according to your contribution.

  - To determine the value of each new team's hypothesis, they re-evaluated the aggregation set while stacking each new model sequentially
  - Each increase in performance denoted the contribution of that new model
  - Doing something weird (different than what other teams may have done), yielded higher contribution values.
    - Models with 3% accuracy often contributed more than their counterparts that achieved much higher accuracies alone
  - This team ended up ranking second overall and their model was ultimately adopted by Netflix

More on the Netflix Challenge

- **BellKor's Pragmatic Chaos** (1st place) was a hybrid team: KorBell (AT&T Research), which won the first Progress Prize milestone in the contest, combined with the Austrian team Big Chaos to improve their score
- **The Ensemble** (2nd place) was also a composite team
- BellKor's Pragmatic Chaos submitted their solution 10 minutes earlier than the second-place team, The Ensemble, while the two teams' algorithms were a perfect tie, score-wise
- The dataset is still available (if you want to beat them)

More on the Netflix Challenge

Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43