THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Applied Machine Intelligence and Reinforcement Learning

Professor Hamza F. Alsarhan

SEAS 8505

Lecture 4

July 6, 2024

Welcome to SEAS Online at George Washington University

Class will begin shortly

Audio: To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (Raise Hand). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, *be sure to mute yourself again*.

Chat: Please type your questions in Chat.

Recordings: As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class for their own private use. Releasing these recordings is strictly prohibited.

Agenda

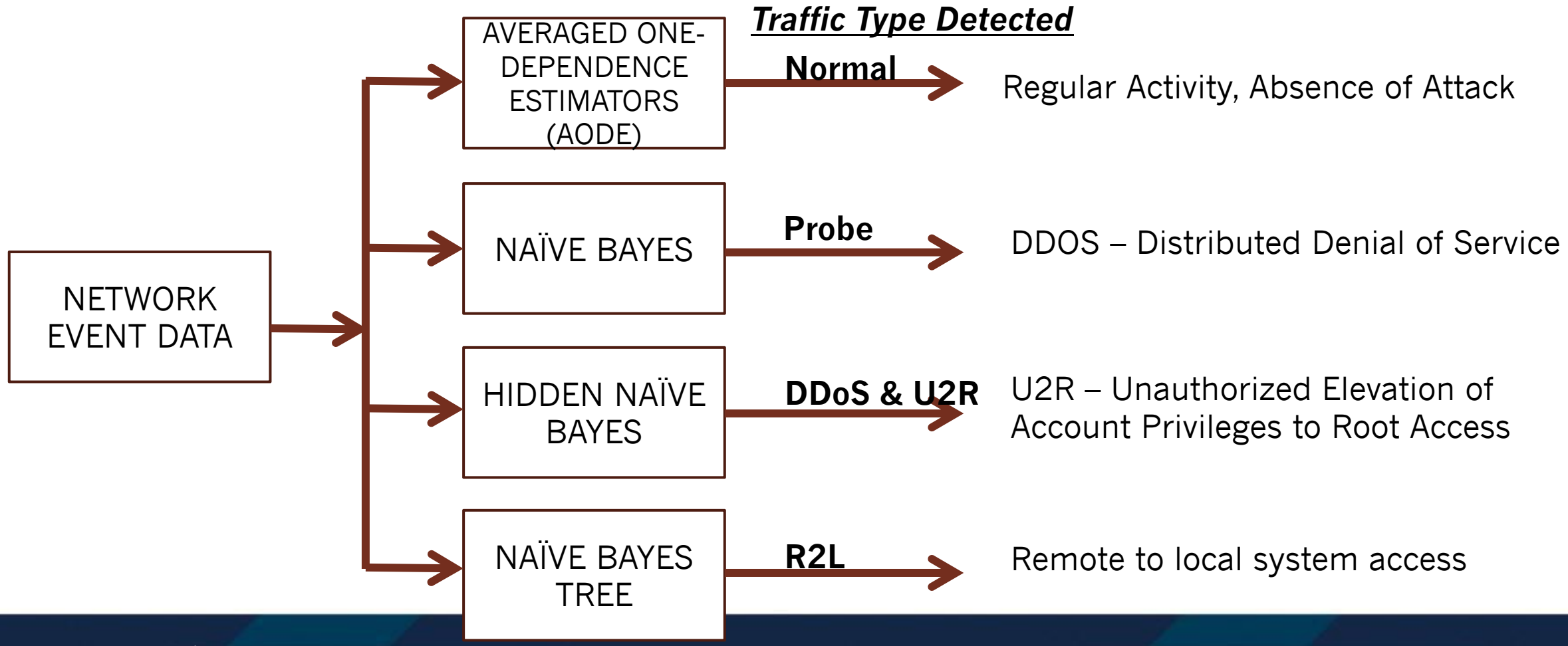
- Ensemble Learning
- AdaBoost
- Random Forests
- What to Expect for the Midterm
- Homework Overview

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Ensemble Learning

Professor Hamza F. Alsarhan
SEAS 8505

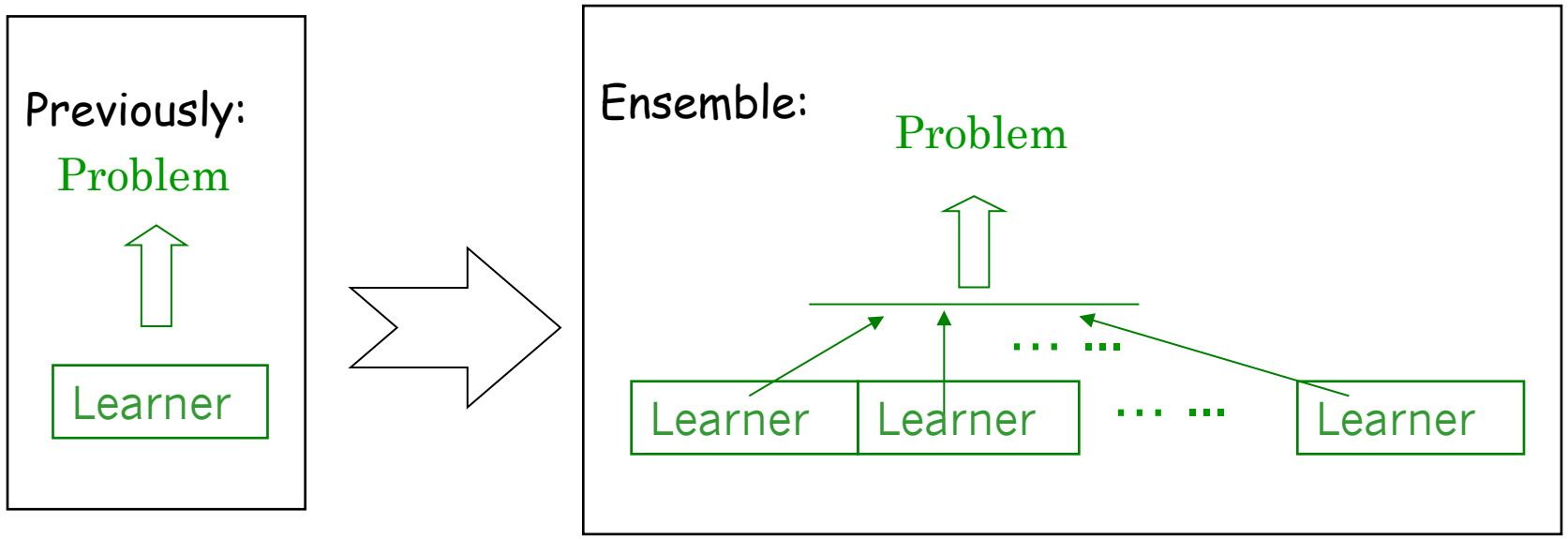
Multiple Classifier Models for Intrusion Detection



(Koc, Dissertation, 2013)

Ensemble Learning

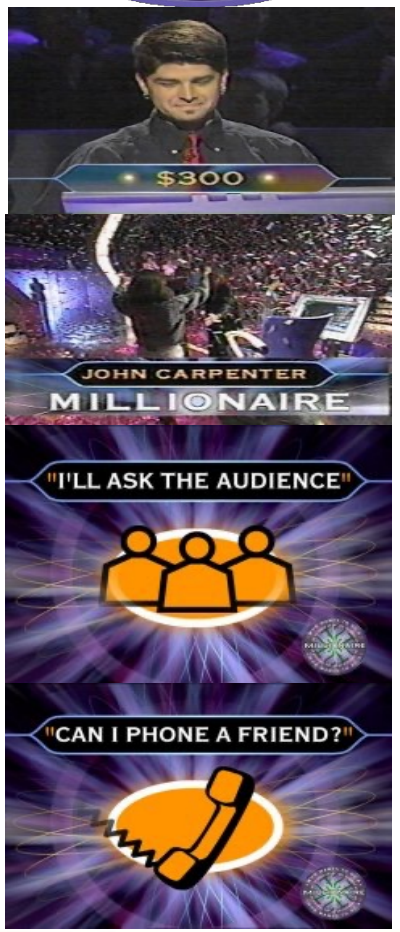
A machine learning paradigm where multiple learners are used to solve the problem



- The generalization ability of the ensemble is usually significantly better than that of an individual learner
- Boosting is one of the most important families of ensemble methods

Based on Rachlin

Decision Making with Ensembles



- **Intuitive basis** for why Ensembles should provide a better decision than other approaches
- **Good performance** on training data does not predict good **generalization** performance (i.e. performance of the classifier on data not seen during training)
- **Too much data:** training different classifiers with different partitions of data, and combining their outputs using an intelligent combination rule is often a more efficient approach
- **Too little data:** resampling techniques resampling techniques can be used for drawing overlapping random subsets of the available data, each of which can be used to train a different classifier, creating the ensemble
- **Data fusion:** data from different sources are combined to make a more informed decision

Based on Polikar

Wisdom of the Crowd

- Suppose you pose a complex question to thousands of random people, then aggregate their answers.
- In many cases you will find that this aggregated answer is better than an expert's answer. This is called the **wisdom of the crowd**.
- Similarly, if you aggregate the predictions of a group of predictors (such as classifiers or regressors), you will often get better predictions than with the best individual predictor.
- **A group of predictors is called an ensemble**; thus, this technique is called Ensemble Learning, and an Ensemble Learning algorithm is called an Ensemble method.

Combining Multiple Models

- Basic idea:
  - Build different “experts”, let them vote
- Advantage:
  - Often improves predictive performance
- Disadvantage:
  - Usually produces output that is very hard to analyze
  - What else?
- But: there are approaches that aim to produce a single comprehensible structure

Rationale

- No Free Lunch Theorem: There is no algorithm that is always the most accurate
- Generate a group of base-learners which when combined has higher accuracy
- Different learners use different:
  - Algorithms
  - Hyperparameters
  - Representations / Modalities / Views
  - Training sets
  - Subproblems
- Diversity vs accuracy
- ***Learning is an ill-posed problem*** and with finite data, each algorithm converges to a different solution and ***fails under different circumstances***

Generating Diverse Learners

- **Different Algorithms -** make different assumptions about the data and lead to different classifiers

- **Different Hyperparameters -** train multiple base-learners with different hyperparameter values, we average over this factor and reduce variance (bias does not change much)

Generating Diverse Learners

- Different Input Representations - to integrate different types of sensors/measurements/modalities. Different representations make different characteristics explicit allowing better identification

- Multiview Learning – example → in image retrieval where in addition to the image itself, we may also have text annotation in the form of keywords. In such a case, we want to be able to combine both of these sources to find the right set of images

Generating Diverse Learners

- **Different Training Sets –** use different subsets of the training set randomly by drawing random training sets from the given sample; (e.g. bagging).
  - Or, the learners can be trained serially so that instances on which the preceding base-learners are not accurate are given more emphasis in training later base-learners (boosting).
  - Examples are boosting and cascading, which actively try to generate complementary learners, instead of leaving this to chance.
- **Diversity vs. Accuracy –** learners that are accurate on different instances, specializing in subdomains of the problem. What we care for is the final accuracy when the base-learners are combined, rather than the accuracies of the base-learners we start from.

Model Combination Schemes

- Multi-expert Combination
  - Gobal or learner fusion approach: Examples: Voting and Stacking.
  - Local or learner selection approach: Example: Gating model with mixture of experts, chooses one (or very few) of the learners

- Multistage Combination
  - Serial approach next combination base-learner is trained with or tested on only the instances where the previous base-learners are not accurate enough. An example is cascading.

Example

- Suppose you have 2,000 different models with their predictions and want to ensemble predictions of best x models. Now, which of the following can be a possible method to select the best x models for an ensemble?

- You can apply stepwise forward selection or stepwise backward elimination.
  - In stepwise forward selection, you will start with empty predictions and will add the predictions of models one at a time if they improve the accuracy of an ensemble.
  - In stepwise backward elimination, you will start with full set of features and remove model predictions one by one if after removing the predictions of model give an improvement in accuracy.

Applying Stepwise Forward Selection

- Suppose you want to apply a stepwise forward selection method for choosing the best models for an ensemble model. What is the first step?
- Note: You have more than 1,000 models predictions

1. Add the models predictions (or in another term take the average) one by one in the ensemble which improves the metrics in the validation set.
2. Start with empty ensemble
3. Return the ensemble from the nested set of ensembles that has maximum performance on the validation set

Ensemble Learning Methods

- Voting
  - Simple majority or weighted
- Stacking
  - Combining disparate methods
- Bagging
  - Bias-variance decomposition, bagging with costs
- Boosting
  - AdaBoost, the power of boosting
- Randomization
  - Random forests, rotation forests

Voting

Linear combination

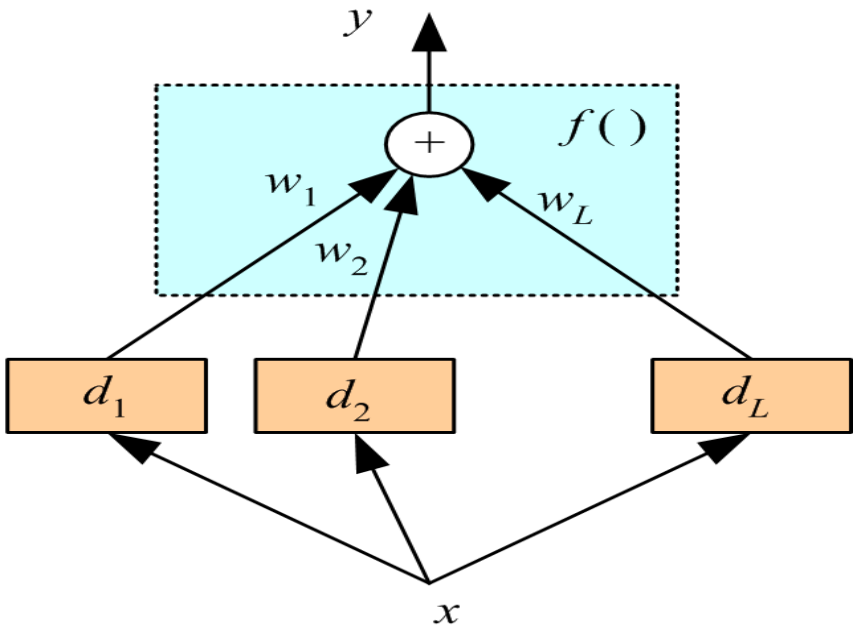
$$y = \sum_{j=1}^{L} w_j d_j$$

$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$

Classification

$$y_i = \sum_{j=1}^{L} w_j d_{ji}$$

In weighted sum, d_{ji} is the vote of learner j for class C_i and w_j is the weight of its vote. Simple voting is a special case where all voters have equal weight, namely, $w_j = 1/L$.



Base-learners are d_j and their outputs are combined using $f(\cdot)$. This is for a single output; in the case of classification, each base-learner has K outputs that are separately used to calculate y_i, and then we choose the maximum. Note that here all learners observe the same input; it may be the case that different learners observe different representations of the same input object or event.

Voting Classifiers

- Aggregate the predictions of each classifier and predict the class that gets the most votes.

- Even if each classifier is a weak learner (meaning it does only slightly better than random guessing), the ensemble can still be a strong learner (achieving high accuracy), provided there are a sufficient number of weak learners and they are ***sufficiently diverse.***

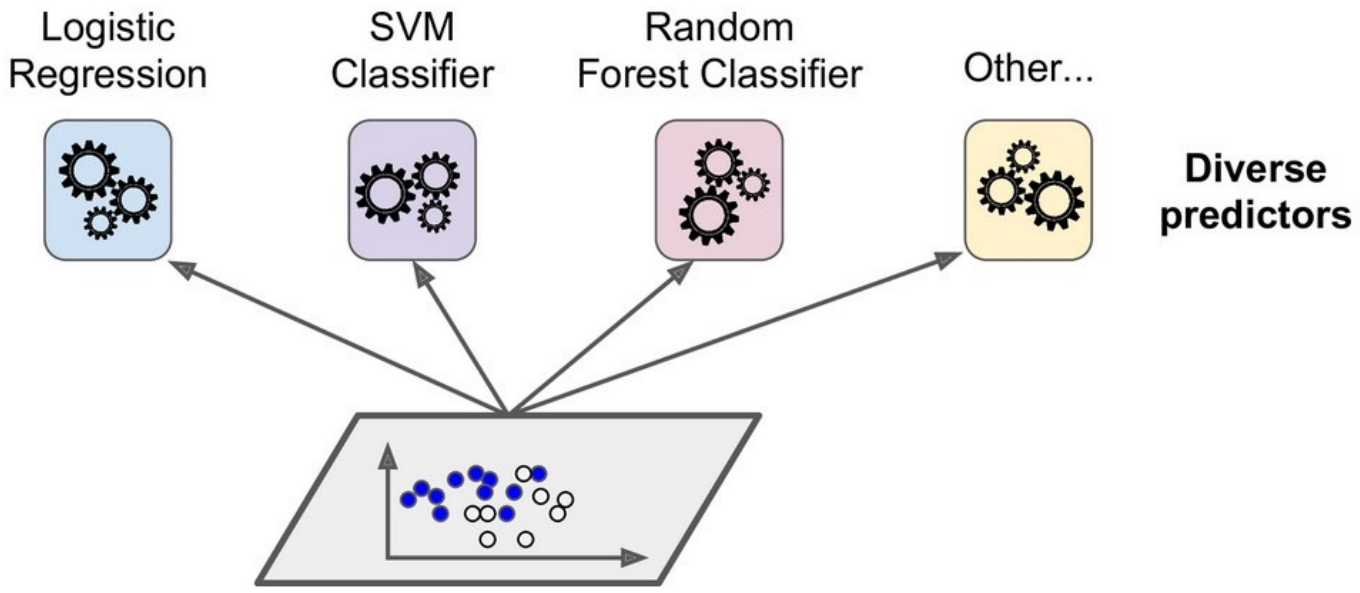


Figure 7-1. Training diverse classifiers

Hard Voting (i.e. majority-vote classifier)

- A simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes. This majority-vote classifier is called a **hard voting classifier.**

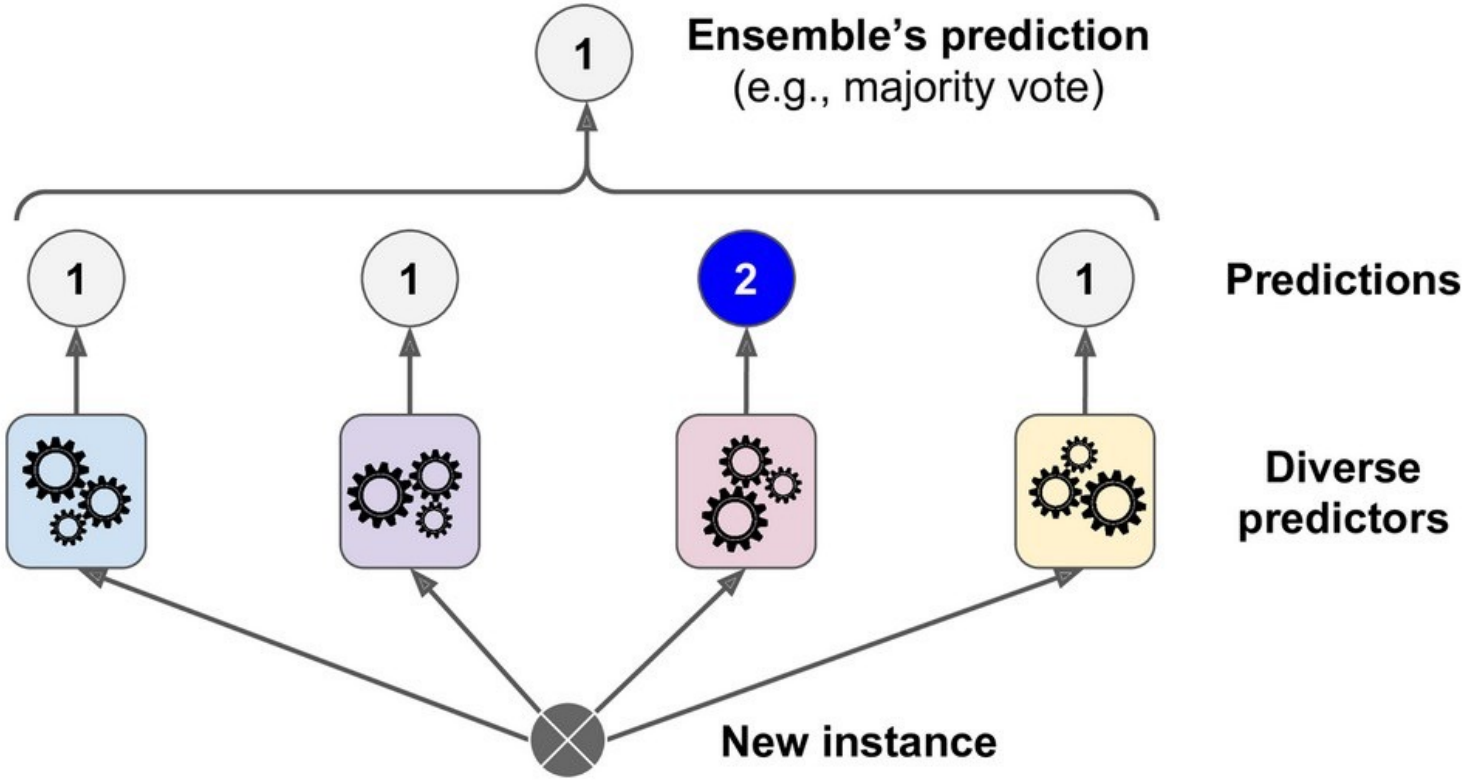


Figure 7-2. Hard voting classifier predictions

An Example of the Law of Large Numbers

- Suppose you have a ***slightly biased*** coin that has a **51%** chance of coming up heads and **49%** chance of coming up tails.
  - Toss it 1,000 times, you will generally get more or less 510 heads and 490 tails, and hence a majority of heads.
  - The probability of obtaining a majority of heads after 1,000 tosses is close to 75%.
  - The more you toss the coin, the higher the probability (e.g., with 10,000 tosses, the probability climbs over 97%).
- The ***law of large numbers***:

As you keep tossing the coin, the ratio of heads gets closer and closer to the probability of heads (51%).

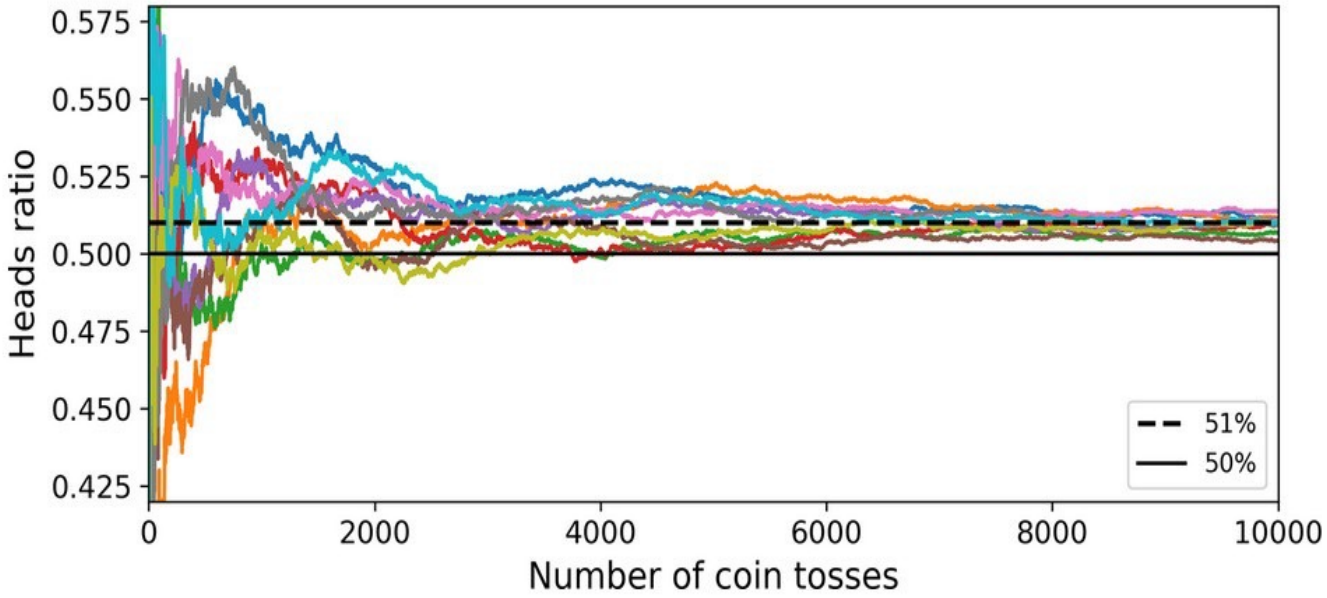


Figure 7-3. The law of large numbers

- Ensemble methods thrive when the predictors exhibit a high degree of independence from one another. Achieving diversity among classifiers can be accomplished by training them using distinctly different algorithms.
- This approach enhances the likelihood that they will produce different types of errors, thereby enhancing the accuracy of the ensemble.

Fixed Combination Rules

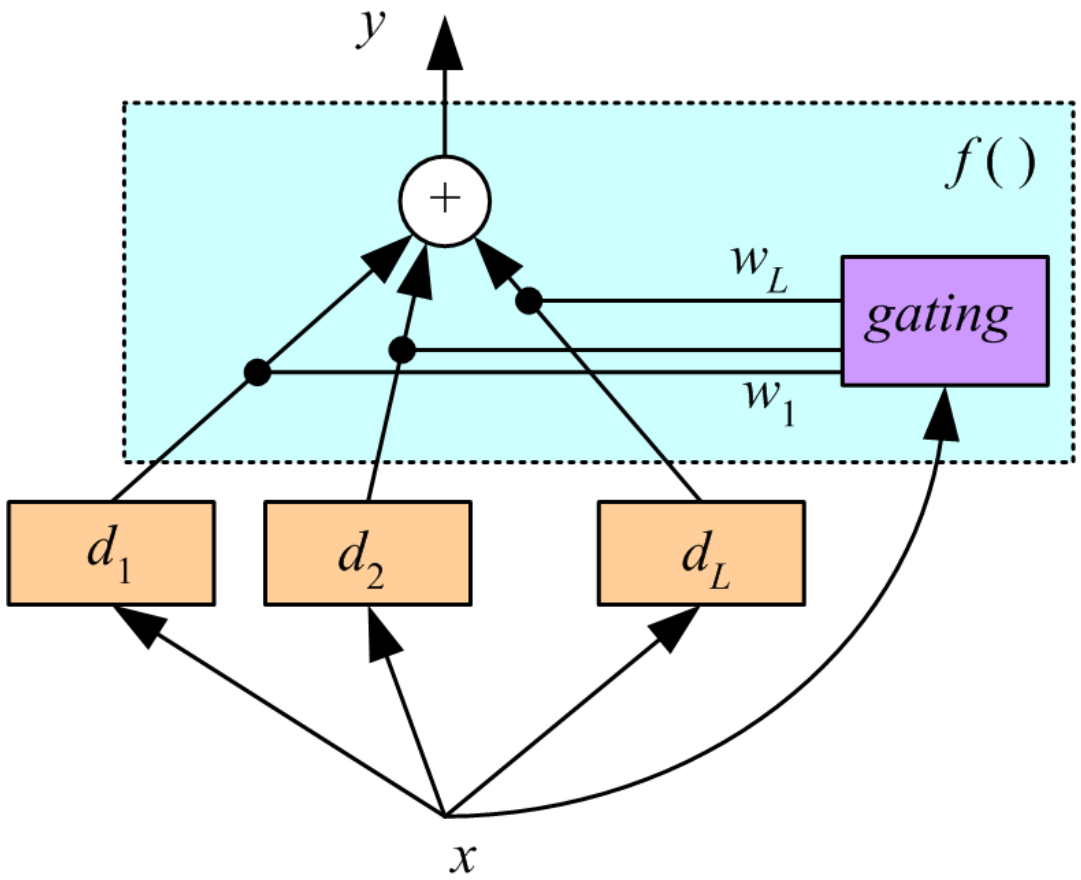
Rule	Fusion function $f(\cdot)$
Sum	$y_i = \frac{1}{L}\sum_{j=1}^{L} d_{ji}$
Weighted sum	$y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$
Median	$y_i = \text{median}_j\, d_{ji}$
Minimum	$y_i = \min_j d_{ji}$
Maximum	$y_i = \max_j d_{ji}$
Product	$y_i = \prod_j d_{ji}$

	C_1	C_2	C_3
d_1	0.2	0.5	0.3
d_2	0.0	0.6	0.4
d_3	0.4	0.4	0.2
Sum	0.2	**0.5**	0.3
Median	0.2	**0.5**	0.4
Minimum	0.0	**0.4**	0.2
Maximum	0.4	**0.6**	0.4
Product	0.0	**0.12**	0.032

Mixture of Experts

- Voting where weights are input dependent (gating)
- Mixture of experts is a voting method where the votes, as given by the gating system, are a function of the input. The combiner system f also includes this gating system.

$$y = \sum_{j=1}^{L} w_j d_j$$

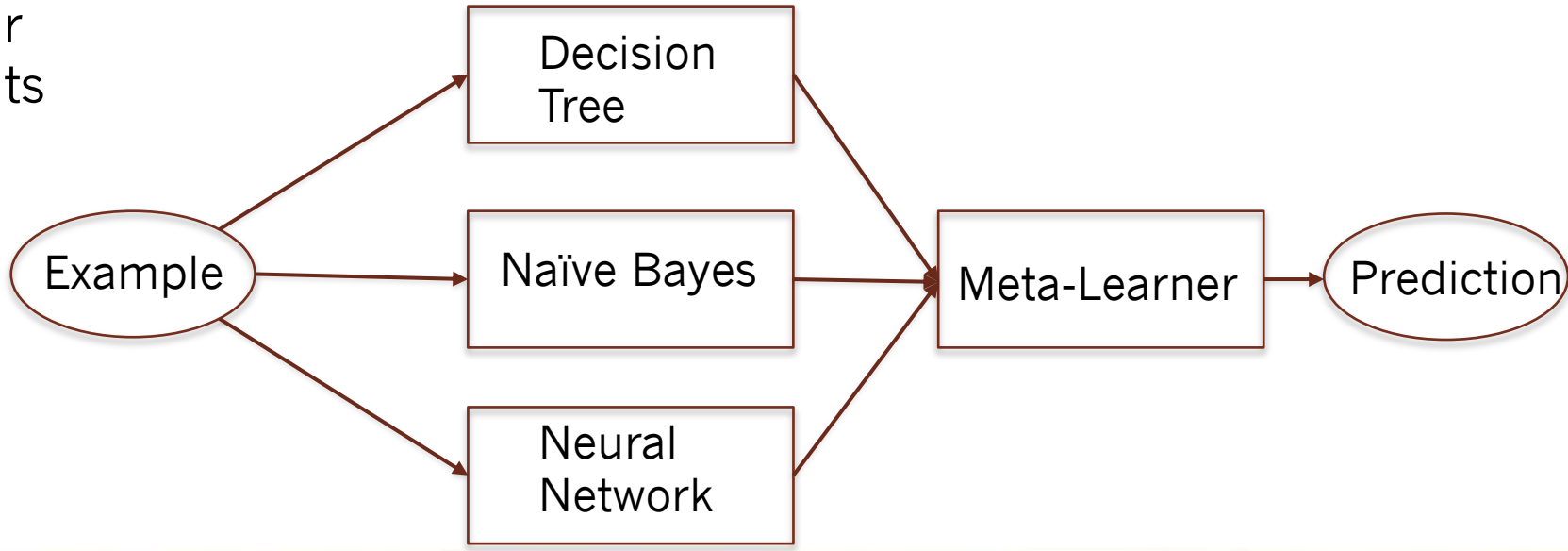


Based on Alpaydin & Jacobs et al.

Stacking (Stacking Generalization)

- Apply Multiple Base Learners

  (e.g. Decision Trees, naïve Bayes, Neural Networks)
- Meta-Learner (relatively simple learner – often linear or logistic regression): Inputs = Base learner prediction
- Base learners are usually different learning schemes
- Meta-Learner Inputs = Predictions on left-out examples



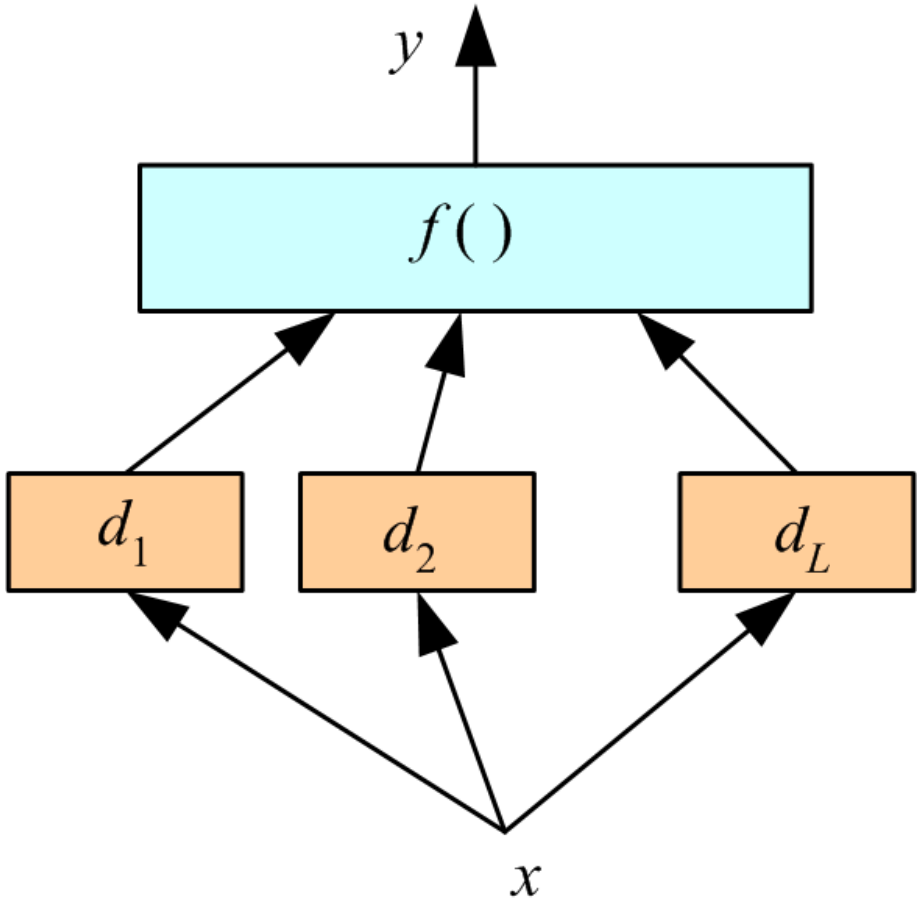
Based on Domingos

Stacking

- Combiner f () is another learner:

  $y = f(d1, d2, \ldots, d_L \mid \Phi)$

- When comparing trained combiners as we have in stacking, with a fixed rule such as in voting, we see that a trained rule is more flexible and may have less bias, but adds extra parameters, risks introducing variance, and needs extra time and data for training. Note also that there is no need to normalize classifier outputs before stacking.



In stacked generalization, the combiner is another learner and is not restricted to being a linear combination as in voting.

Stacking

- Instead of using trivial functions (such as hard voting) to aggregate the predictions of all predictors in an ensemble, we can train a model to perform the aggregation.
- In the figure, we have an ensemble performing a regression task on a new instance.
  - Each of the bottom three predictors predicts a different value (3.1, 2.7, and 2.9).
  - The final predictor (called a *blender*, or a *meta learner*) takes these predictions as inputs and makes the final prediction (3.0).

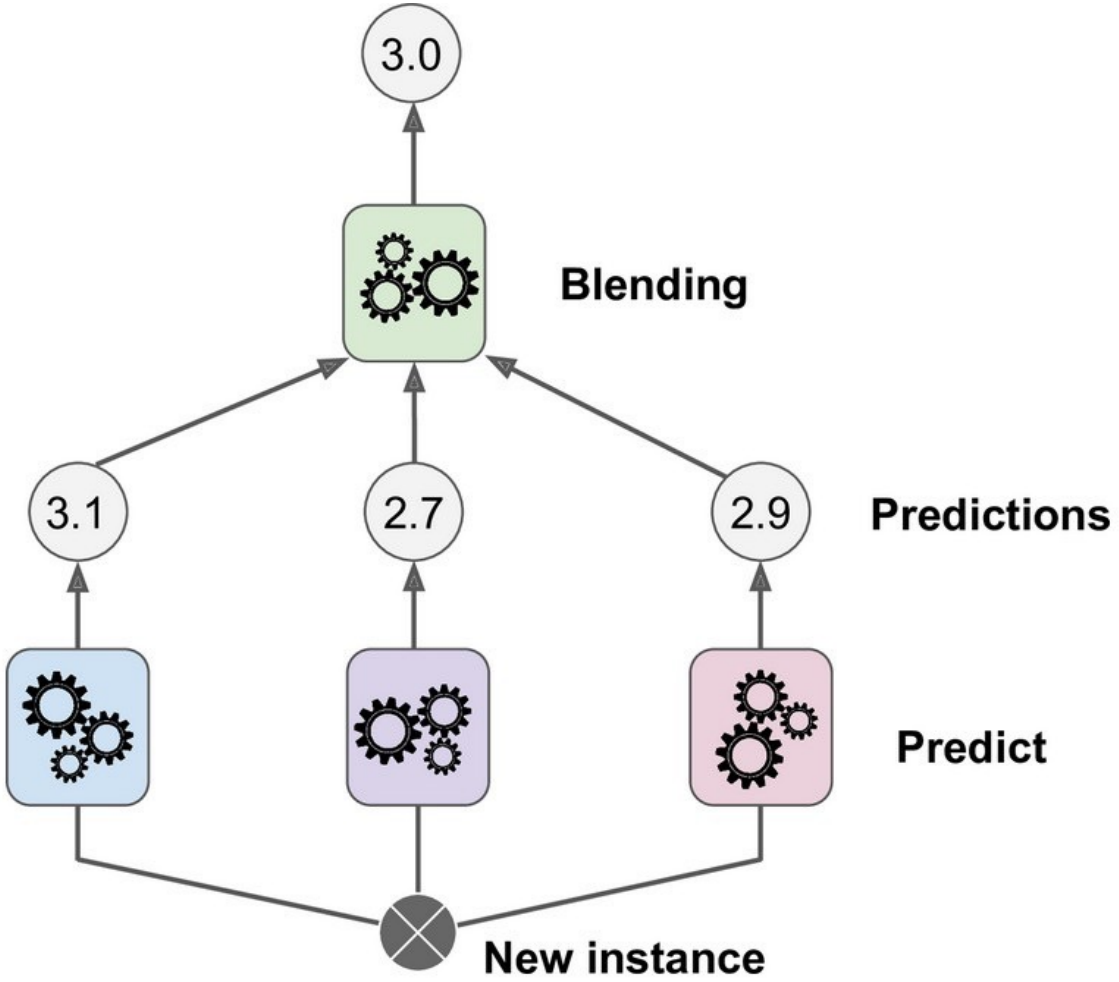


Figure 7-12. Aggregating predictions using a blending predictor

Train the Blender – Train the First Layer First

- A common approach is to use a hold-out set.
- First, the training set is split into two subsets.
  - The first subset is used to train the predictors in the first layer.
  - The first layer's predictions are used to make predictions on the second (hold-out) set.

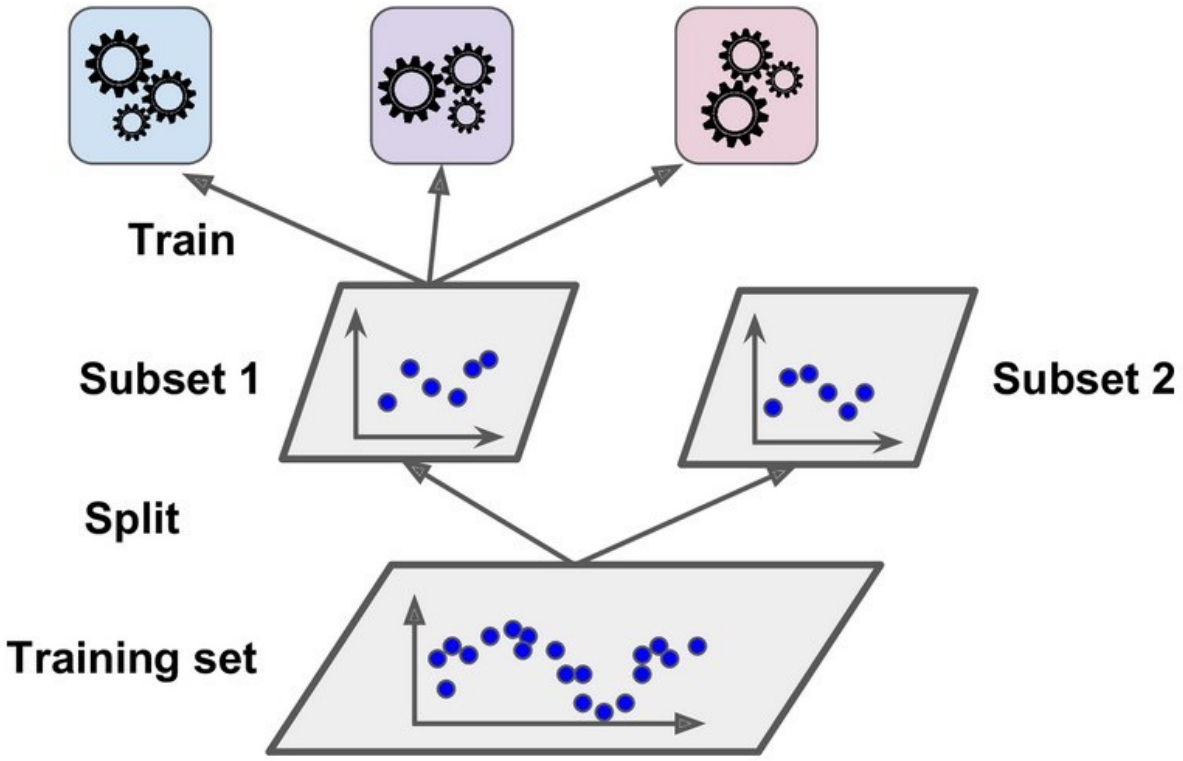


Figure 7-13. Training the first layer

Train the Blender – Create a Blending Training Set

- Next, create a new training set using the first layer's predicted values as features, keep the target values.

- The blender is trained on this new dataset, so it learns to predict the target value, given the first layer's predictions.

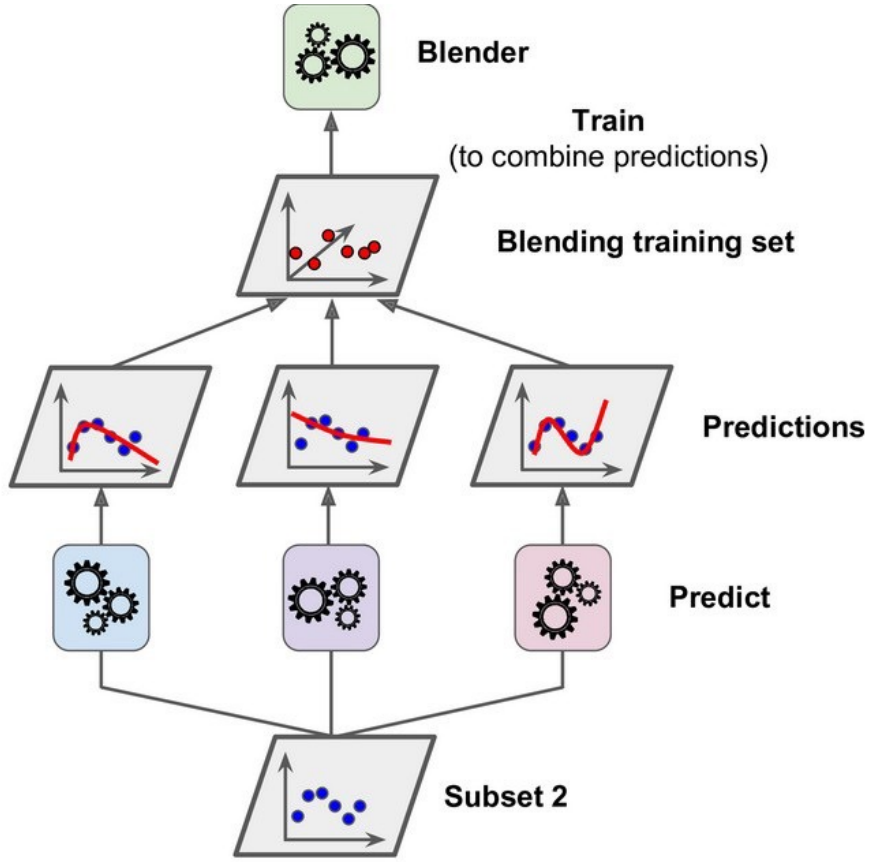


Figure 7-14. Training the blender

Extending this to Multilayer Stacking Ensembles

- We can also train several different blenders this way (e.g., one using Linear Regression, another using Random Forest Regression), to get a whole layer of blenders.
- The trick is to split the training set into three subsets:
  - The first one is used to train the first layer.
  - The second one is used to create the training set used to train the second layer (using predictions made by the predictors of the first layer).
  - The third one is used to create the training set to train the third layer (using predictions made by the predictors of the second layer).
- Once this is done, we can make a prediction for a new instance by going through each layer sequentially.

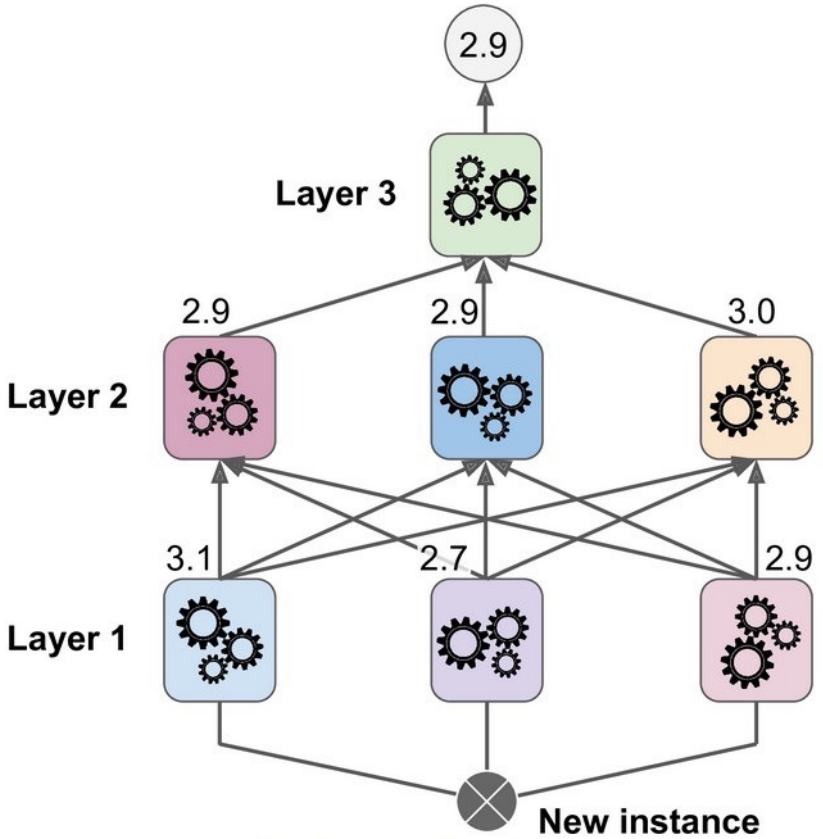


Figure 7-15. Predictions in a multilayer stacking ensemble

Stacking

- Question: how to build a heterogeneous ensemble consisting of different types of models (e.g., decision tree and neural network)
  - Problem: models can be vastly different in accuracy
- Idea: to combine predictions of base learners, do not just vote, instead, use meta learner
  - In stacking, the base learners are also called level-0 models
  - Meta learner is called level-1 model
  - Predictions of base learners are input to meta learner
- Base learners are usually different learning schemes
- Caveat: cannot use predictions on training data to generate data for level-1 model!
  - Instead use scheme based on cross-validation

Based on Witten

More on Stacking

- Stacking is hard to analyze theoretically: “black magic”
- If the base learners can output class probabilities, use those as input to meta learner instead of plain classifications
  - Makes more information available to the level-1 learner
- Important question: which algorithm to use as the meta learner (aka level-1 learner)?
  - In principle, any learning scheme
  - In practice, prefer “relatively global, smooth” models because
  - Base learners do most of the work and
  - This reduces the risk of overfitting
- Note that stacking can be trivially applied to numeric prediction too

Fine-Tuning an Ensemble

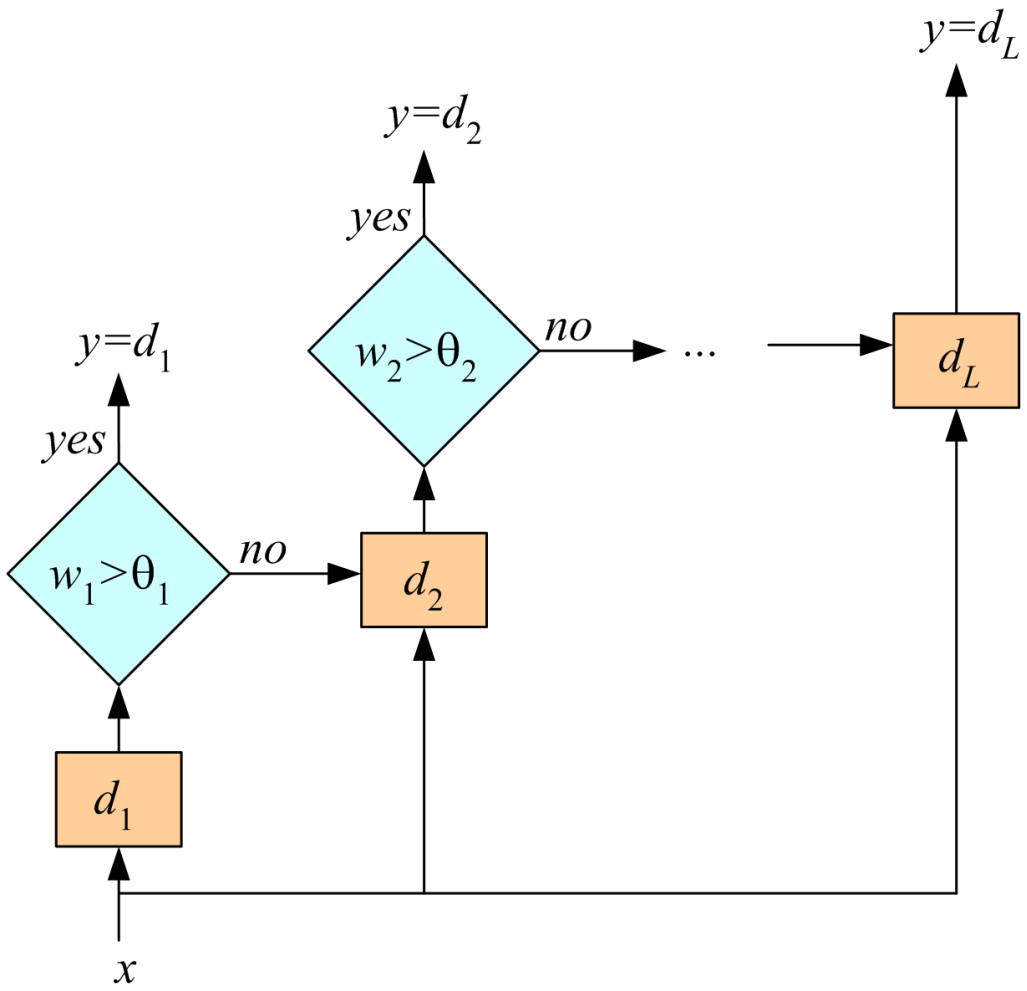
- Given an ensemble of dependent classifiers, do not use it as is, try to get independence
  1. Subset selection: Forward (growing)/Backward (pruning) approaches to improve accuracy/diversity/independence
  2. Train metaclassifiers: From the output of correlated classifiers, extract new combinations that are uncorrelated.

- Similar to feature selection vs feature extraction

Cascading

- Use d_j only if preceding ones are not confident
- Cascade learners in order of complexity (simple models first)

- Cascading is a multistage method where there is a sequence of classifiers, and the next one is used only when the preceding ones are not confident.



Combining Multiple Sources/Views

- **Early integration:** Concatenate all features and train a single learner
- **Late integration:** With each feature set, train one learner, then either use a fixed rule or stacking to combine decisions
- **Intermediate integration:** With each feature set, calculate a kernel, then use a single SVM with multiple kernels
- **Combining features** vs decisions vs kernels

Multiple Kernel Learning

- Considered to be “intermediate integration
- Single kernel machine classifier that uses multiple kernels for different inputs
- For different sources, there are different notions of similarity calculated by their kernels
- Alpaydin’s research:
  - Using multiple kernels instead of a single one is useful and combining kernels in a nonlinear or data-dependent way seems more promising than linear combination in fusing information provided by simple linear kernels.
  - However, linear methods are more reasonable when combining complex Gaussian kernels.

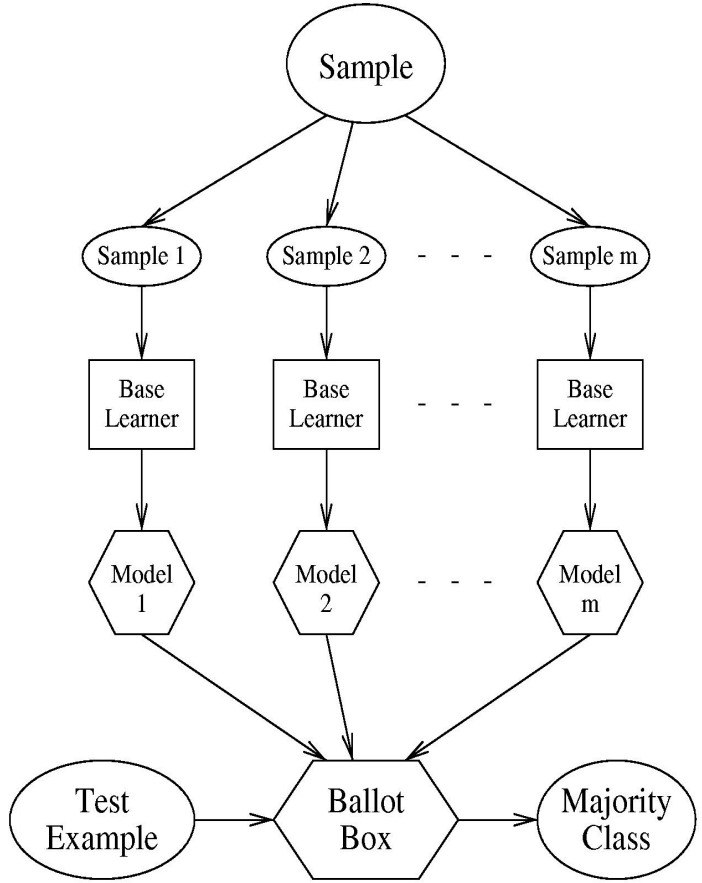
Bagging and Pasting

- Another way to get a diverse set of classifiers is to use the same training algorithm for every predictor and train them on different random subsets of the training set.
  - **Bagging (bootstrap aggregating)** - when sampling is performed ***with*** replacement (i.e. data instances can be sampled more than once to create a given training set)
  - **Pasting** - when sampling is performed ***without*** replacement (i.e. any particular data instance can only be sampled once)

- Both bagging and pasting allow training instances to be sampled several times across multiple predictors, but only bagging allows training instances to be sampled several times for the same predictor.

Bootstrapping as an Ensemble Method

- Bootstrap Aggregating (Bagging)
- Generate “bootstrap” samples of training set by sampling **with** replacement
- Learn one model on each replicate
- Combine with uniform voting
- Use Average or Median with Regression
- Unstable Algorithms improve with Bagging
- i.e. If learning scheme is unstable, Bagging almost always improves performance
- An unstable learner: a small change in training data can make a big change in model (e.g., when learning decision trees)



Based on Domingos

Properties of Bagging

- Parameters that can be tuned for finding a good ensemble model in bagging-based algorithms:
  - Max number of samples
  - Max features
  - Bootstrapping of samples
  - Bootstrapping of features

Nice Properties of Bagging Classifiers

- In bagging, individual learners are not dependent on each other so they can be used in parallel
- Bagging is suitable for high variance low bias models (for complex models).
- Therefore what can you say?

Out-of-Bag Evaluation

- With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all.
  - Only about 63% of the training instances are sampled on average for each predictor
  - The remaining 37% of the training instances that are not sampled are called out-of-bag (oob) instances (not the same 37% for all predictors)
- A bagging ensemble can be evaluated using oob instances, without the need for a separate validation set.
  - Indeed, if there are enough estimators, then each instance in the training set will likely be an oob instance of several estimators, so these estimators can be used to make a fair ensemble prediction for that instance.
  - Once you have a prediction for each instance, you can compute the ensemble's prediction accuracy (or any other metric).

Reducing Variance

- Each individual predictor has a higher bias than if it were trained on the original training set, but aggregation reduces both bias and variance.
- Generally, the net result is that the ensemble has a similar bias but a lower variance than a single predictor trained on the original training set.
- The ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors.

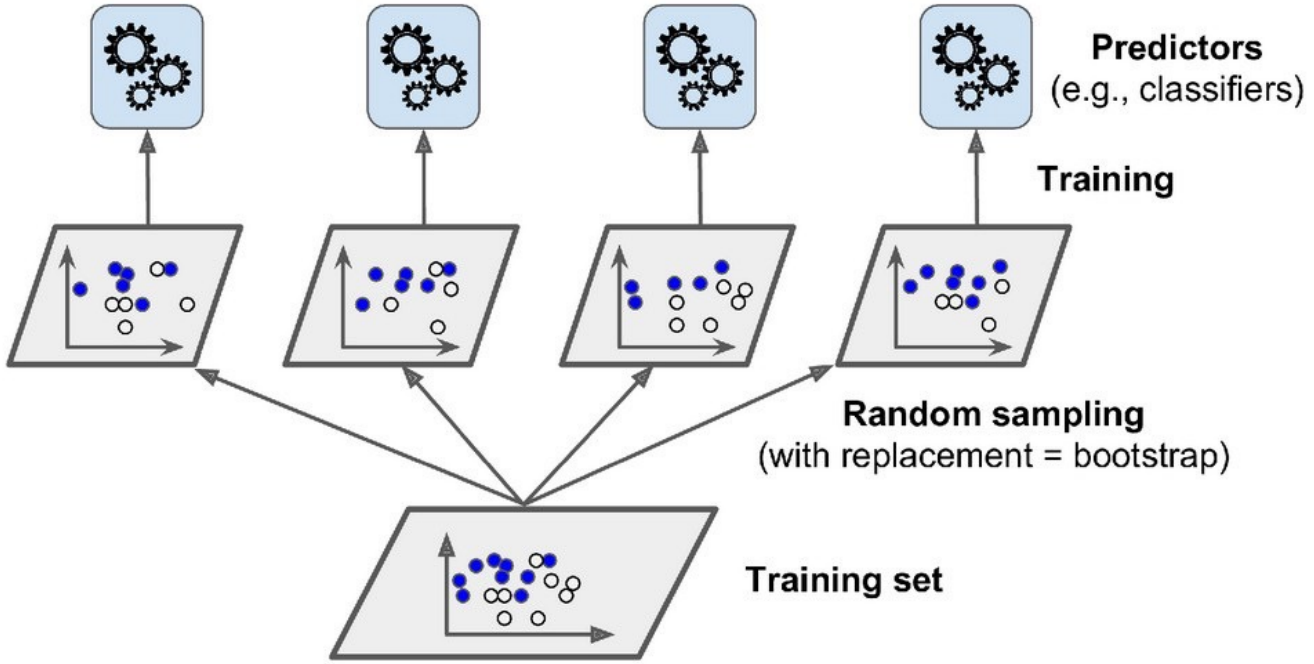


Figure 7-4. Bagging and pasting involves training several predictors on different random samples of the training set

Reduced Variance

- The ensemble's predictions will likely generalize much better than the single Decision Tree's predictions.
- The ensemble has a comparable bias but a smaller variance (it makes roughly the same number of errors on the training set, but the decision boundary is less irregular).

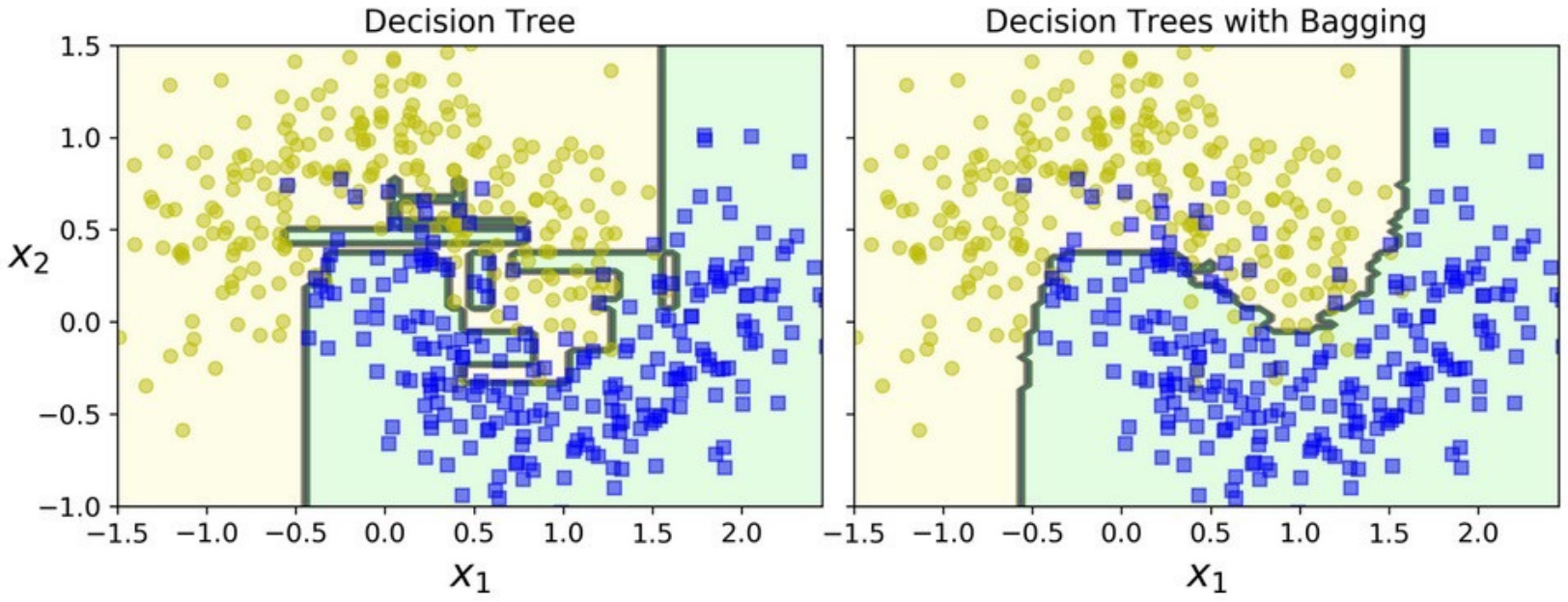


Figure 7-5. A single Decision Tree (left) versus a bagging ensemble of 500 trees (right)

Computational Considerations

- Ensemble approach benefits from parallelism.
  - Predictors can all be trained in parallel, via different CPU cores or even different servers.
  - Predictions can be made in parallel.
- This is one of the reasons bagging and pasting are such popular methods: they scale very well.

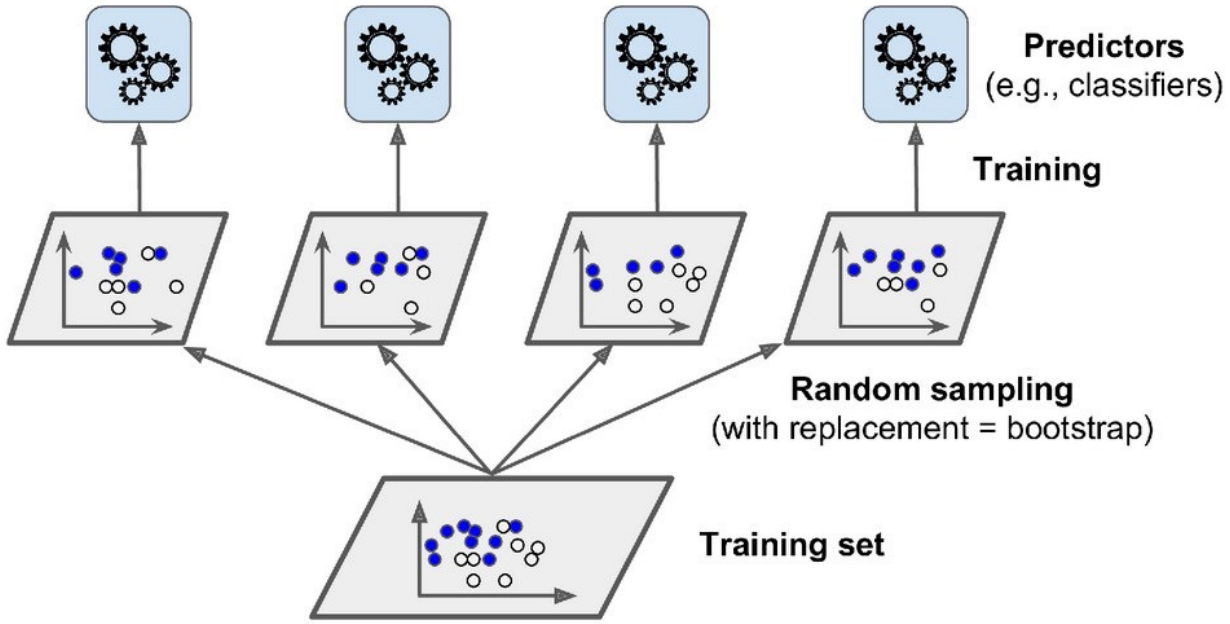


Figure 7-4. Bagging and pasting involves training several predictors on different random samples of the training set

Random Patches and Random Subspaces

- Similar to sampling instances from the dataset, there is a a concept of sampling the features, as well. With feature sampling, each predictor will be trained on a random subset of the input features.
- This technique is particularly useful when you are dealing with high-dimensional inputs (such as images).
  - Sampling both training instances and features is called the **Random Patches method**.
  - Keeping all training instances but sampling features is called the **Random Subspaces method**.
- Sampling features results in even more predictor diversity (main purpose), trading a bit more bias for a lower variance.

Bootstrapping vs Pasting

- Bootstrapping (resampling) introduces a bit more diversity in the subsets that each predictor is trained on, so bagging ends up with a *slightly higher bias* than pasting.
- The extra diversity which results from bagging means that the predictors end up being less correlated, so the ensemble's variance is reduced.
- Overall, bagging often results in better models, which explains why it is generally preferred.
- Use cross-validation to evaluate both bagging and pasting and select the one that works best.

Boosting

- Bagging can easily be parallelized because ensemble members are created independently
- Boosting is an alternative approach
- Also uses voting/averaging
- But: weights models according to performance
- Iterative: new models are influenced by performance of previously built ones
  - Encourage new model to become an “expert” for instances misclassified by earlier models
  - Intuitive justification: models should be experts that complement each other
- Many variants of boosting exist, we cover a couple

Boosting High Level Overview

- Maintain vector of weights for examples
- Initialize Uniform Weights
- Loop:
  - Apply learner to weighted examples (sample)
  - Increase weights of misclassified examples
- Combine models by weighted voting

Based on Domingos

Boosting

- In 1990 it was proven that a weak learner, an algorithm that generates classifiers that can merely do better than random guessing, can be turned into a strong learner that generates a classifier that can correctly classify all but an arbitrarily small fraction of the instances
- Creates an ensemble of classifiers by resampling the data, which are then combined by majority voting.
- **Basic Algorithm:**
  - Classifier C1 is trained with a random subset of the available training data.
  - The training data subset for the second classifier C2 is chosen as the most informative subset, given C1. That is, C2 is trained on a training data only half of which is correctly classified by C1, and the other half is misclassified.
  - The third classifier C3 is trained with instances on which C1 and C2 disagree (tie breaker).
  - The three classifiers are combined through a three-way majority vote.
- AdaBoost: is a more general version of the original boosting algorithm (1997 – AT&T Bell Labs Research)

Based on Polikar

A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting

A gambler, frustrated by persistent horse-racing losses and envious of his friends' winnings, decides to allow a group of his fellow gamblers to make bets on his behalf. He decides he will wager a fixed sum of money in every race, but that he will apportion his money among his friends based on how well they are doing. Certainly, if he knew psychically ahead of time which of his friends would win the most, he would naturally have that friend handle all his wagers. Lacking such clairvoyance, however, he attempts to allocate each race's wager in such a way that his total winnings for the season will be reasonably close to what he would have won had he bet everything with the luckiest of his friends. In this paper, we describe a simple algorithm for solving such dynamic allocation problems, and we show that our solution can be applied to a great assortment of learning problems. Perhaps the most surprising of these applications is the derivation of a new algorithm for ``boosting,'' i.e., for converting a ``weak'' PAC learning algorithm that performs just slightly better than random guessing into one with arbitrarily high accuracy.

Freund, Yoav, and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." Journal of computer and system sciences 55.1 (1997): 119-139. (Cited 16,860 times!)

AdaBoost

- When training an AdaBoost classifier:
  - The algorithm first trains a base classifier (such as a Decision Tree) and uses it to make predictions on the training set.
  - The algorithm then increases the relative weight of misclassified training instances.
  - Then it trains a second classifier, using the updated weights, and again makes predictions on the training set, updates the instance weights, and so on.

- The new classifier pays a bit more attention to the training instances that its predecessor *underfitted (e.g. "got wrong")*
- This results in the new classifiers focusing more and more on the hard cases.

Gradually Getting Better

- The first classifier gets many instances wrong, so their weights get boosted.
- The second classifier therefore does a better job on these instances, and so on.
- AdaBoost adds predictors to the ensemble, gradually making it better.

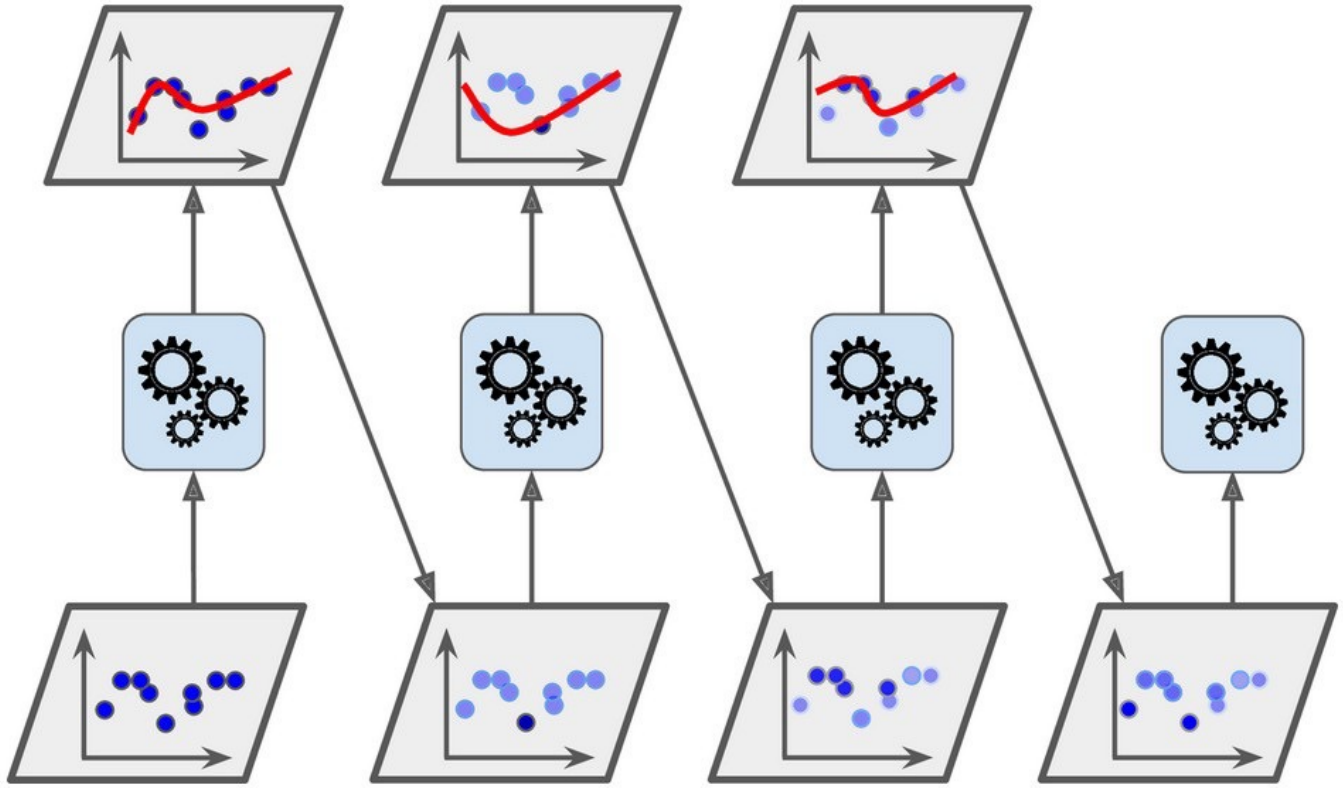
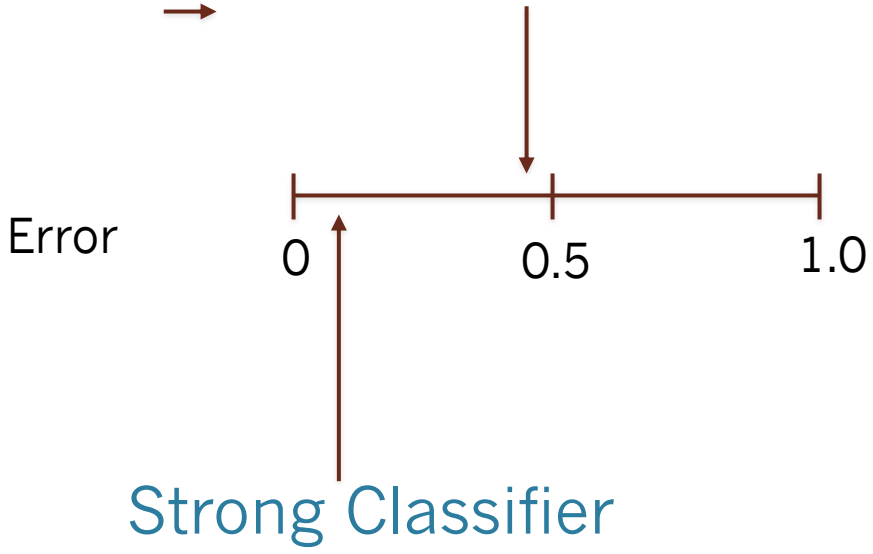


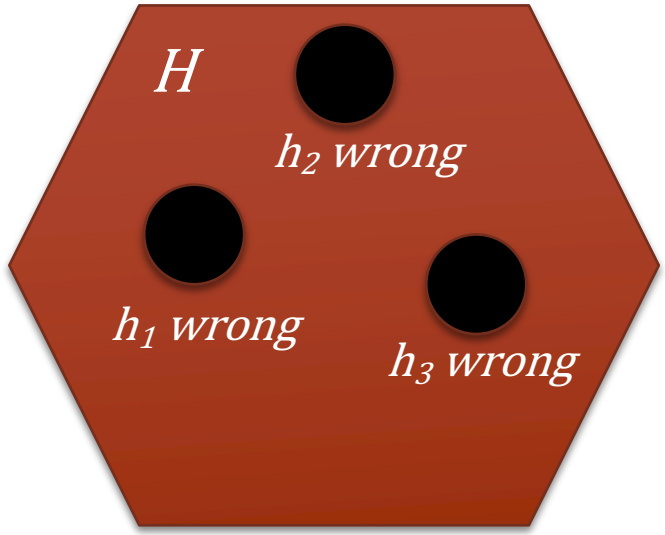
Figure 7-7. AdaBoost sequential training with instance weight updates

Deriving AdaBoost

Weak Classifier (Just slightly better than flipping a coin)

$h \quad [-1, +1]$





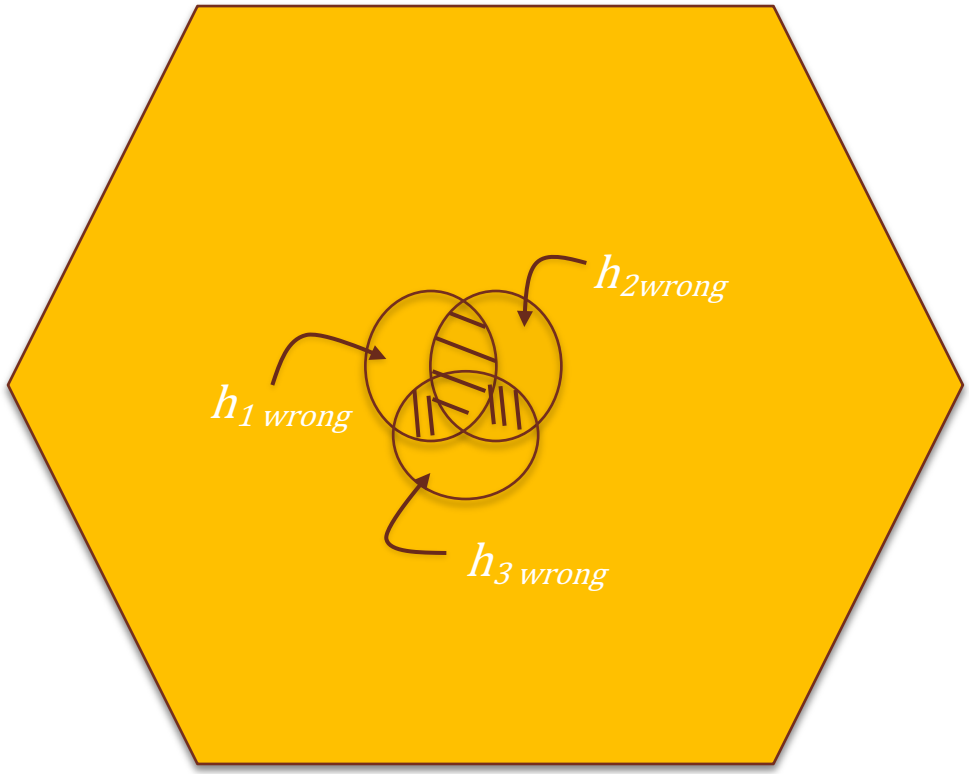
Voting Scheme:

$$H(x) = sign\left(h^1(x) + h^2(x) + h^3(x)\right)$$

This is done per sample

Based on Winston

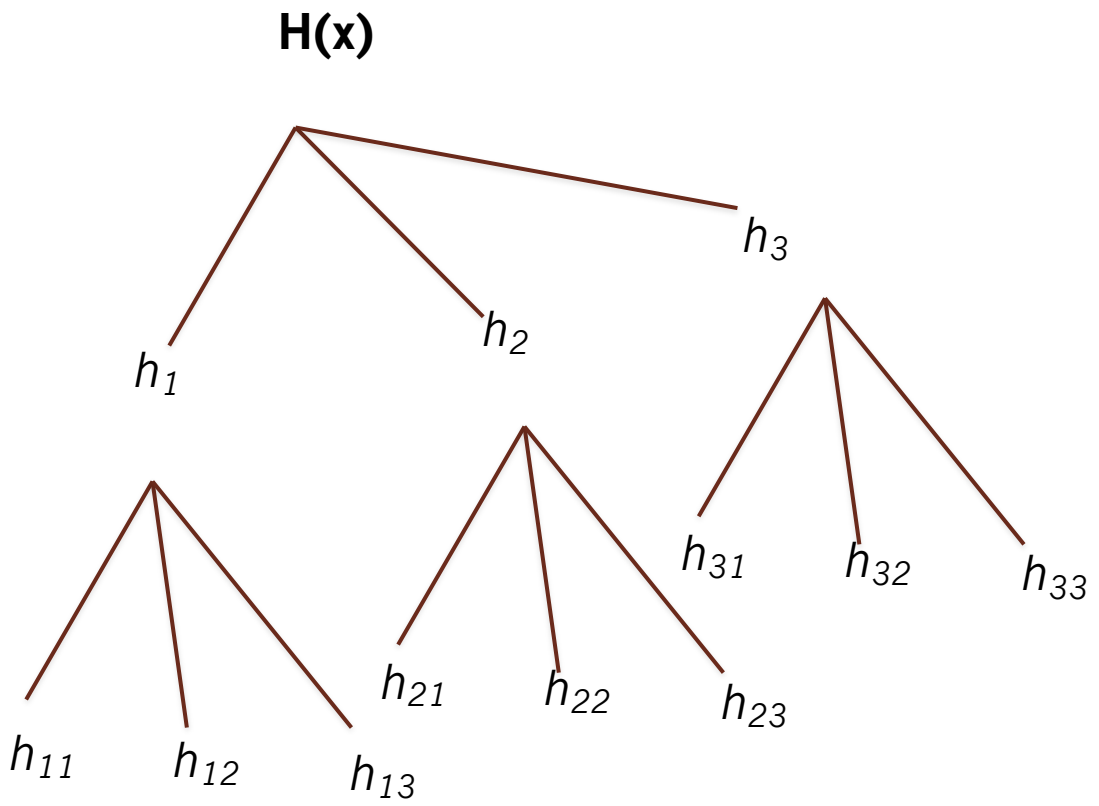
Overlapping Errors



$$Data \rightarrow h^1$$
$$More\ Heavily\ Weight\ h^1\ errors \rightarrow h^2$$
$$More\ Heavily\ Weight\ h^1 \neq h^2 \rightarrow h^3$$

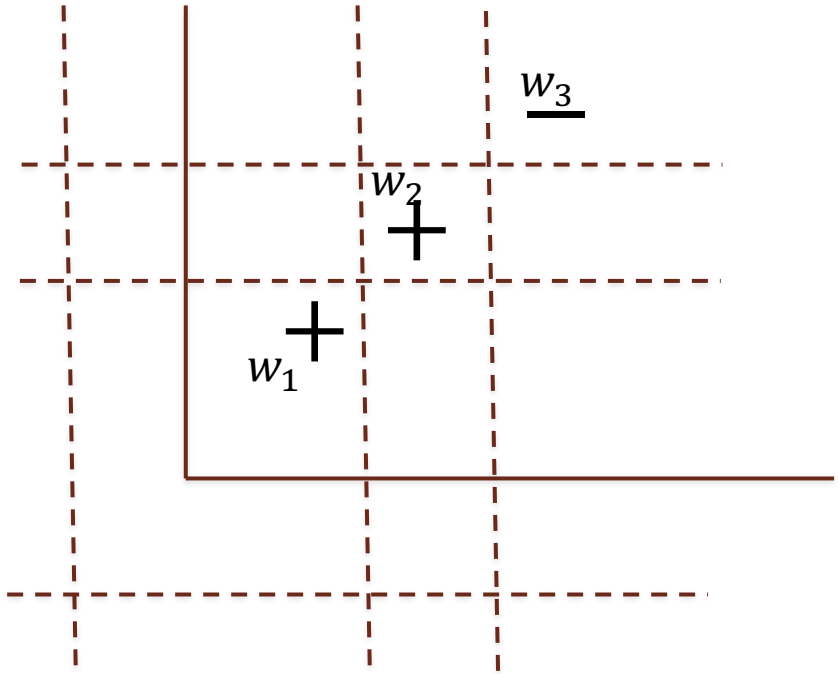
Based on Winston

Voting Scheme



Based on Winston

Decision Tree Stumps



How many tests can we get out of this?

$$\epsilon = \sum_{wrong} \frac{1}{N} \; where \; N \; is \; the \; number \; of \; samples$$

w_i at step 1 is $\frac{1}{N}$

And we also know that:

$$\sum_i w_i = 1 \; which \; enforces \; \; a \; distribution$$

Ultimately we have:

$$\epsilon = \sum_{wrong} w_i$$

Based on Winston

Combining Classifiers

$$H(x) = sign(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x) + \ \ldots)$$

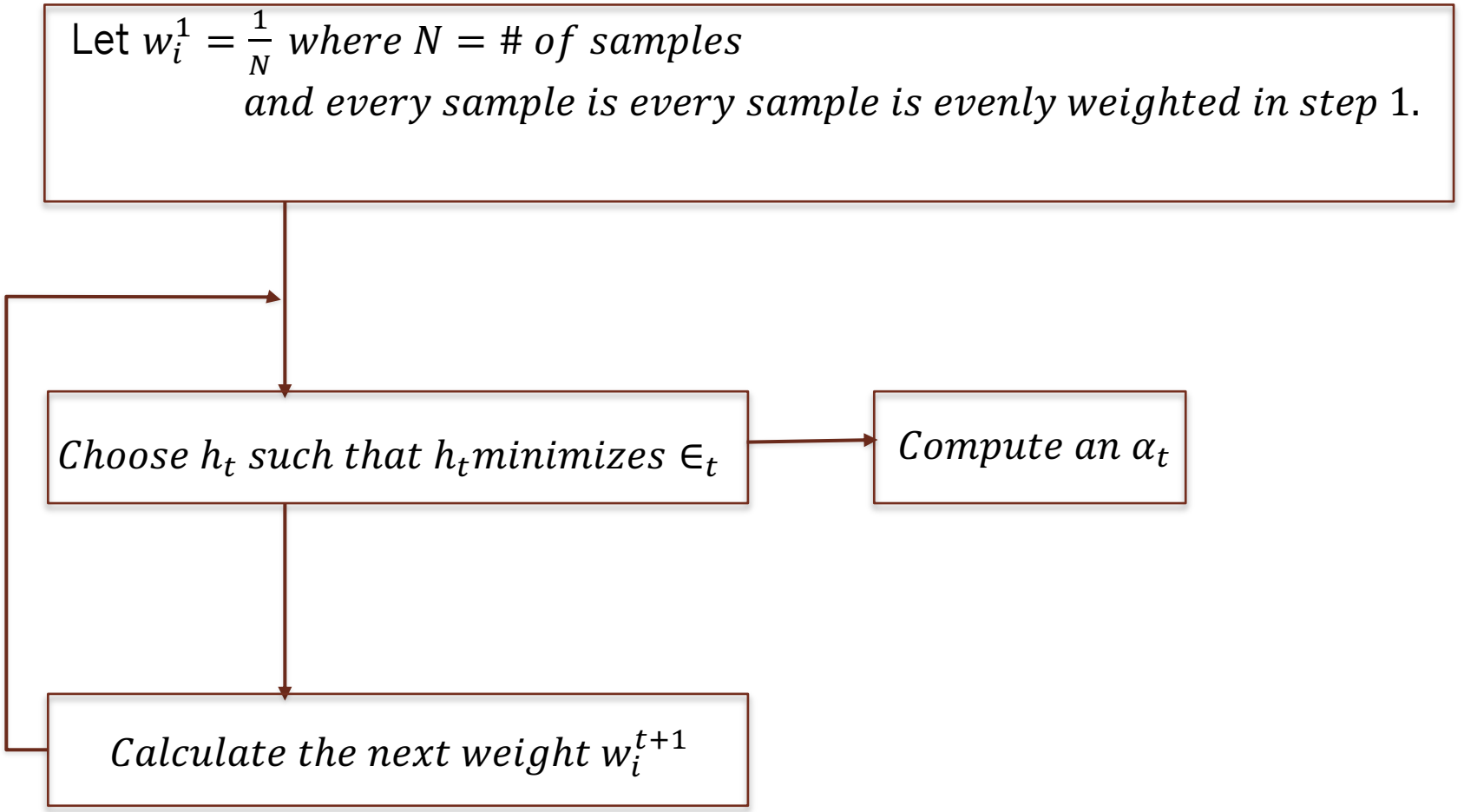
The wisdom of a crowd

OR

"The wisdom of a weighted crowd of experts"

Based on Winston

High-Level Sketch of the Boosting Algorithm



Based on Winston

But How Do We Calculate the Next Weight: w_i^{t+1} (weight of instance I at iteration t)?

$$w_i^{t+1} = \frac{w_i^t}{z} e^{-\alpha^t h^t(x) y(x)} \qquad Note: \ z \ is \ a \ normalizing \ factor$$

$$y(x) is \ the \ function \ we \ saw \ with \ SVMs \ [+1. -1]$$

h(x)	y(x)	h(x) * y(x)	Agreement
-1	-1	1	Yes
1	1	1	Yes
1	-1	-1	No
-1	1	-1	No

$$The \ \alpha's \ are \ from \ the \ voting \ equation:$$
$$H(x) = sign(\alpha^1 h^1(x) + \alpha^2 h^2(x) + \alpha^3 h^3(x))$$

It turns out that to get the minimum error, ϵ_t : $\alpha^t = \frac{1}{2} ln\left(\frac{1-\epsilon^t}{\epsilon^t}\right)$

Based on Winston

Minimum Error Bound

The minimum error **bound** for the voting function

$$H(x) = sign(\alpha^1 h^1(x) + \alpha^2 h^2(x) + \alpha^3 h^3(x))$$

is:

$$\alpha^t = \frac{1}{2} ln\left(\frac{1-\epsilon^t}{\epsilon^t}\right)$$

Based on Winston

Simplifications

Let's plug in the minimum error bound alphas:

$$\alpha^t = \frac{1}{2} ln\left(\frac{1-\epsilon^t}{\epsilon^t}\right)$$

into our weight update formula:

$$w_i^{t+1} = \frac{w_i^t}{z} e^{-\alpha^t h^t(x) y(x)}$$

$$w_i^{t+1} = \frac{w_i^t}{z} \begin{cases} \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} \; if \; correct \\ \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} \; if \; not \; correct \end{cases}$$

Based on Winston

More Simplifications

We need to figure out what z is going be (normalizing factor)

So we should add up all of the elements to find z:

$$Z = \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} \sum_{Correct} w_i^t + \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} \sum_{Incorrect} w_i^t$$

Insight: The sum over the incorrect samples is ϵ^t and the sum over the correct samples $1-\epsilon^t$ therefore:

$$Z = \sqrt{\frac{\epsilon^t}{1-\epsilon^t}}(1-\epsilon^t) + \sqrt{\frac{1-\epsilon^t}{\epsilon^t}}(\epsilon^t) = 2\sqrt{\epsilon^t(1-\epsilon^t)}$$

Based on Winston

Plugging in Z …

$$Z = 2\sqrt{\epsilon^t(1-\epsilon^t)}$$

$$w_i^{t+1} = \frac{w_i^t}{z} \begin{cases} \sqrt{\frac{\epsilon^t}{1-\epsilon^t}} \; if \; correct \\ \sqrt{\frac{1-\epsilon^t}{\epsilon^t}} \; if \; not \; correct \end{cases}$$

$$w_i^{t+1} = \frac{w_i^t}{2} \frac{1}{1-\epsilon^t} \; if \; correct \rightarrow \frac{1}{2}\frac{1}{1-\epsilon^t} \sum_{correct} w_i^t = \frac{1}{2}$$

$$= \frac{w_i^t}{2} \frac{1}{\epsilon^t} \; if \; not \; correct \rightarrow \frac{1}{2}\frac{1}{\epsilon^t} \sum_{incorrect} w_i^t = \frac{1}{2}$$

$1-\epsilon^t$

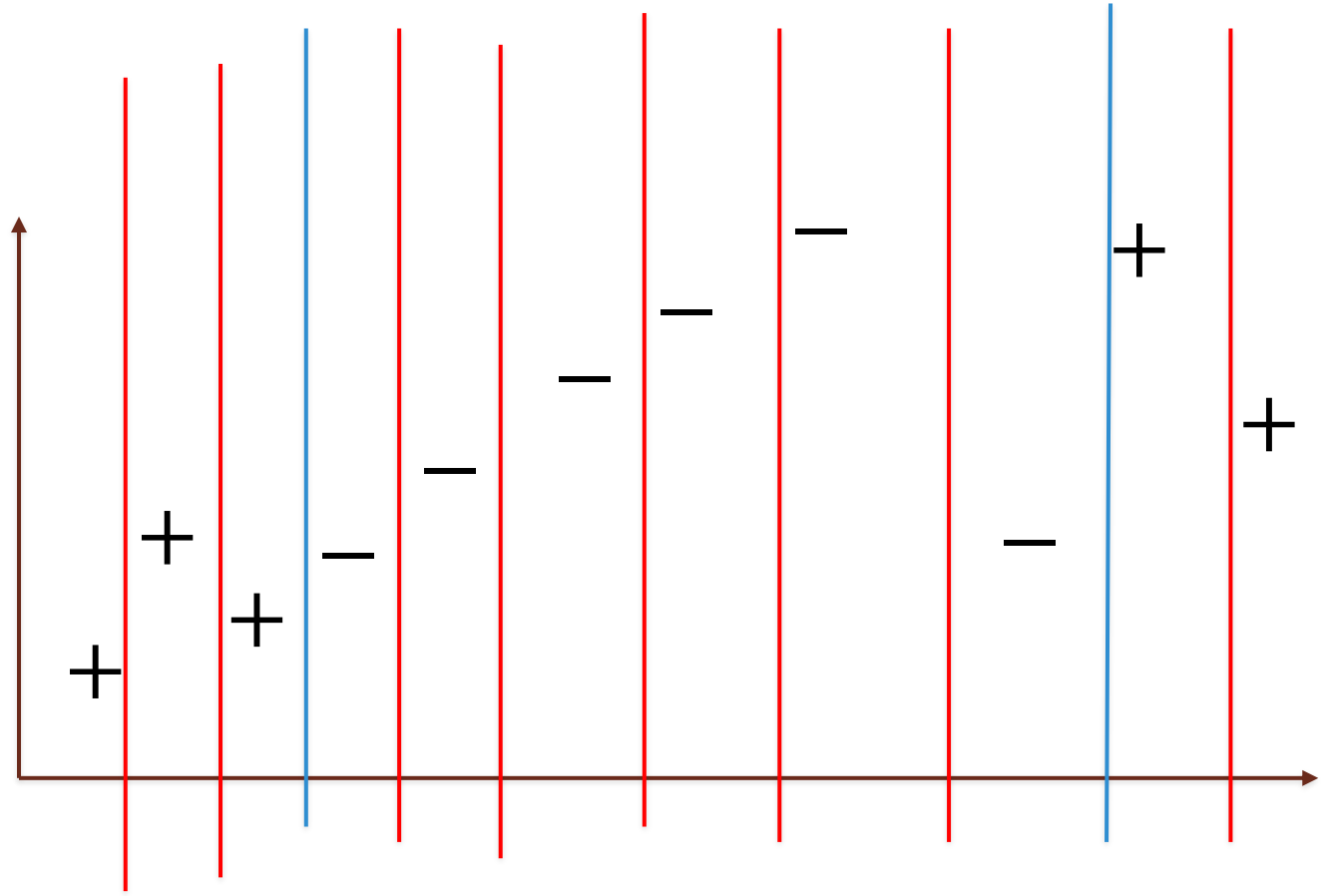
ϵ^t

It turns out that Adaboost is just scaling the weights for the answers we got correct by ½ and also scaling the weights For the answers we go wrong by ½ and then normalizing so that all the weights add to One to enforce a distribution!

Based on Winston

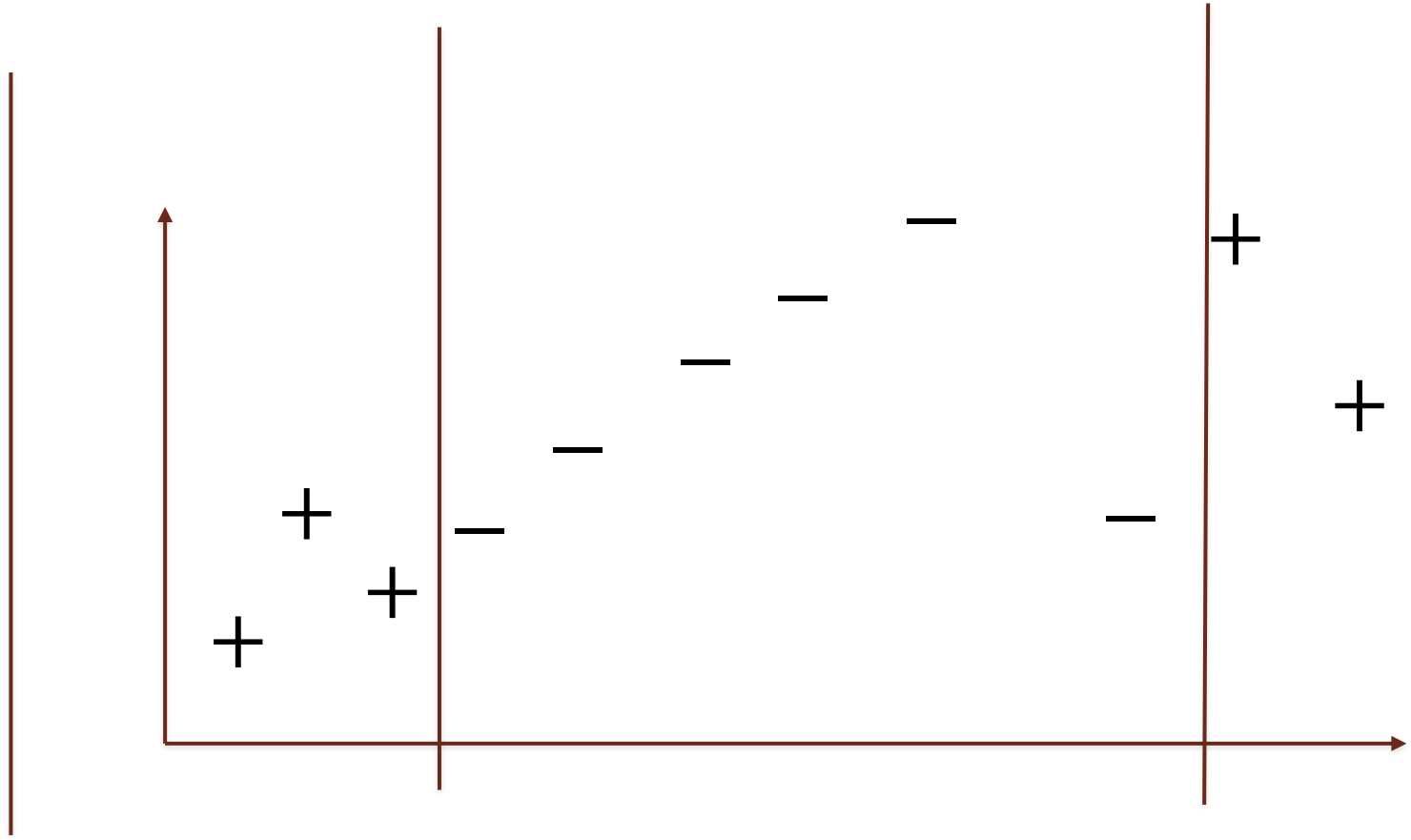
How Many Decisions Are Required Over the Sample Set?

No test that lies between two correctly classified samples will ever provide new information.



Required Decision Tree Stumps

No test that lies between two correctly classified samples will ever provide new information.

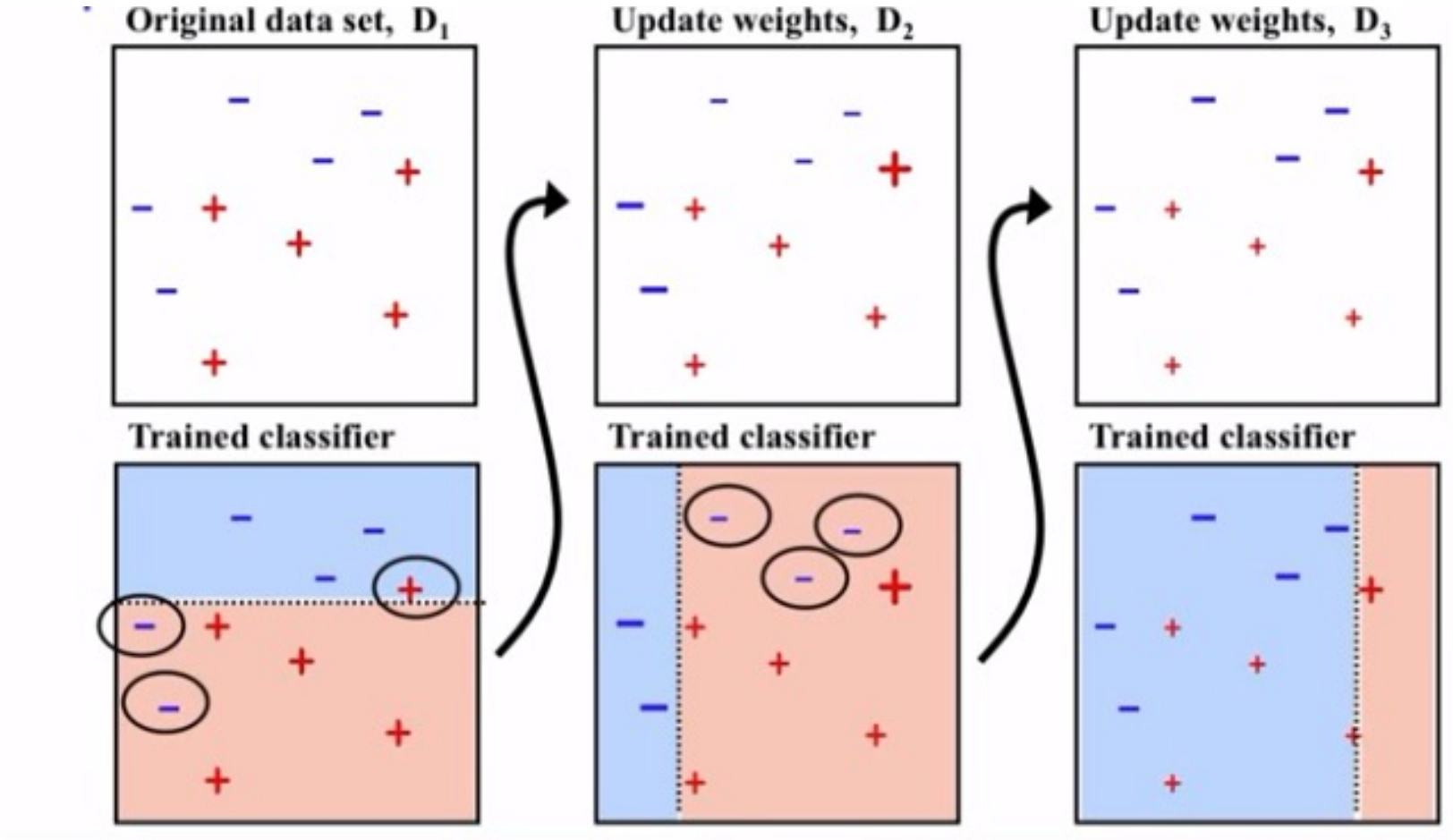


High-Level Description of AdaBoost

- Weighted combinations of classifiers
- “Committee” decisions
  - Trivial example
  - Equal weights (majority vote)
  - Might want to weight unevenly – up-weight good experts
- Boosting
  - Focus new experts on examples that others get wrong
  - Train experts sequentially
  - Errors of early experts indicate the “hard” examples
  - Focus later classifiers on getting these examples right
  - Combine the whole set in the end
  - Convert many “weak” learners into a complex classifier

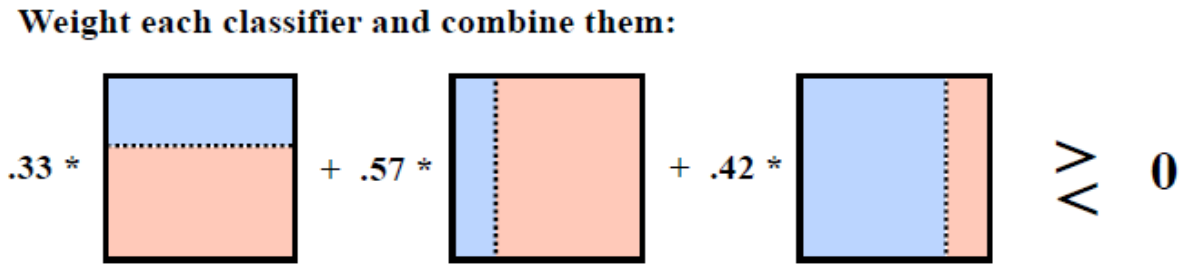
Based on Ihler

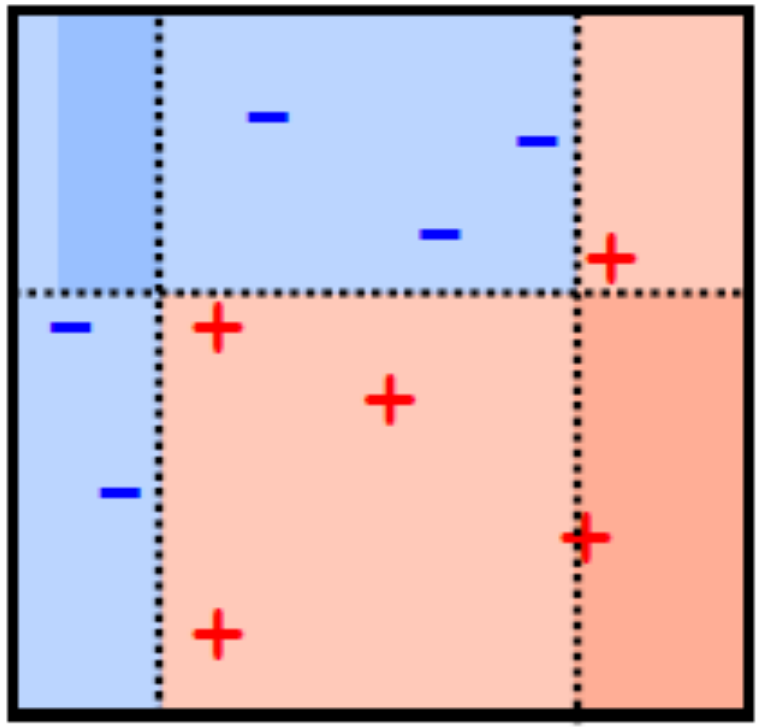
Boosting Example



Based on Ihler

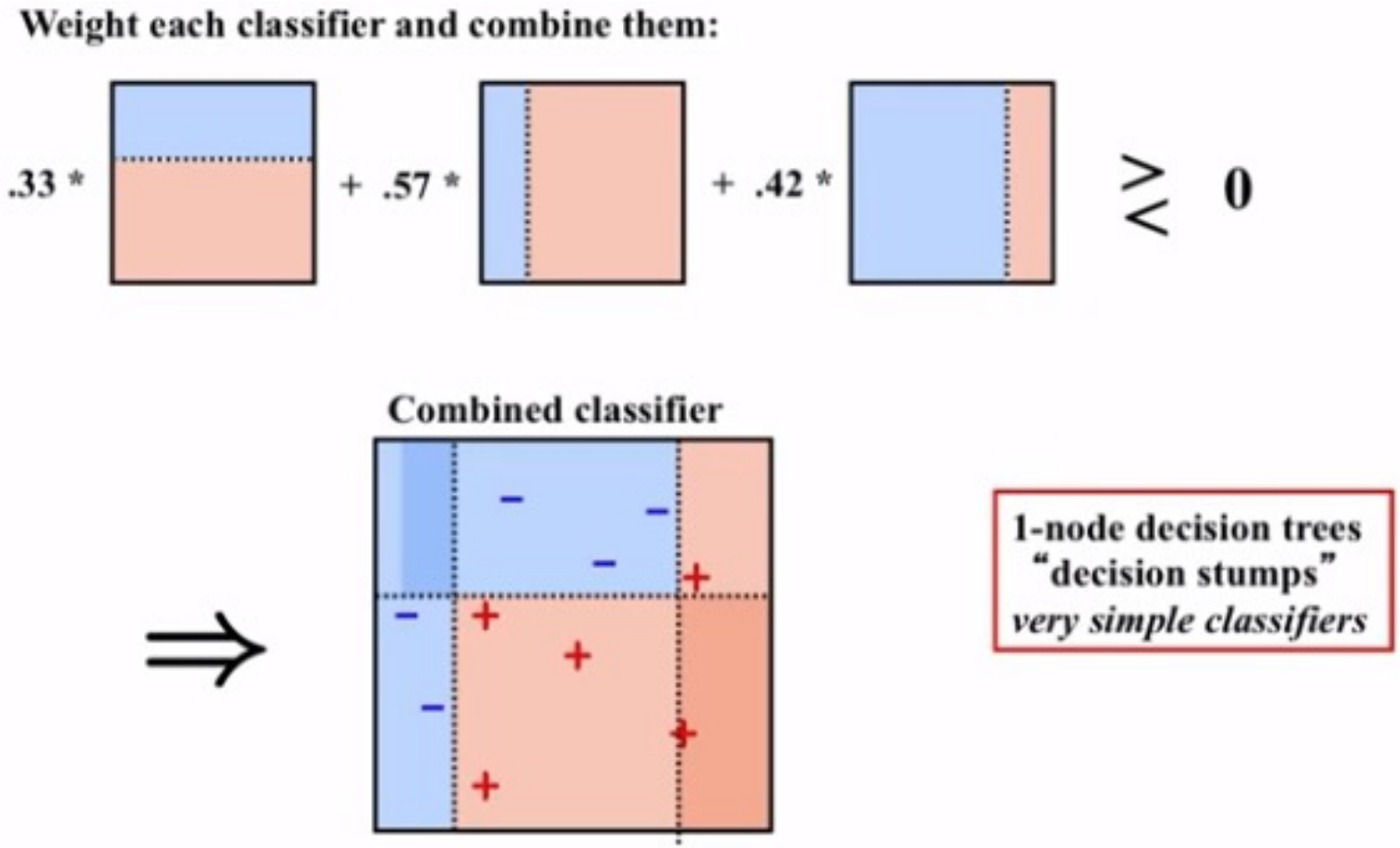
Combined Classifier





Based on Ihler

High-Level Description of AdaBoost



Based on Ihler

Properties

- Question: True or False – In boosting, individual base learners can be parallel.
  A. True
  B. False

Comments on the Application of AdaBoost.M1

- The AdaBoost.M1 boosting algorithm stems from work in *computational learning theory*
- Theoretical result:
  - Training error decreases exponentially as iterations are performed
- Other theoretical and empirical results:
  - Works well if base classifiers are not too complex and
  - Their error does not become too large too quickly as more iterations are performed

The AdaBoost Algorithm

TrainAdaBoost(D, BaseLearn)
For each example d_i in D let its weight w_i=1/|D|
Let H be an empty set of hypotheses
For t from 1 to T do:
Learn a hypothesis, h_t, from the weighted examples: h_t=BaseLearn(D)
Add h_t to H
Calculate the error, ε_t, of the hypothesis h_t as the total sum weight of the examples that it classifies incorrectly.
If $\varepsilon_t > 0.5$ then exit loop, else continue.
Let $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$
Multiply the weights of the examples that h_t classifies correctly by β_t
Rescale the weights of all of the examples so the total sum weight remains 1.
Return H

TestAdaBoost(ex, H)
Let each hypothesis, h_t, in H vote for ex's classification with weight $\log(1/\beta_t)$
Return the class with the highest weighted vote total.

Based on Mooney

More Comments on the Application of Boosting

- Continue boosting after training error = 0?
- Puzzling fact: generalization error continues to decrease!
  - Seems to contradict Occam's Razor
- Possible explanation: Consider margin (confidence), not just error
  - A possible definition of margin: difference between estimated probability for true class and nearest other class (between –1 and 1)
  - Margin continues to increase with more iterations
- AdaBoost.M1 works well with so-called weak learners; only condition: error does not exceed 0.5
  - Example of weak learner: decision stump
- In practice, boosting sometimes overfits if too many iterations are performed (in contrast to bagging)

Computational Consideration – Practical Tips

- The sequential learning technique cannot be parallelized (or only partially) since each predictor can only be trained after the previous predictor has been trained and evaluated.
  - Does not scale as well as bagging or pasting.

- If your AdaBoost ensemble is overfitting the training set, you can try reducing the number of estimators or more strongly regularizing the base estimator.

Computational Consideration – Practical Tips

- Simplicity
  - Simple and easy to program
  - No parameters to tune (except T – number of iterations)
  - Flexible — can combine with any learning algorithm
  - No prior knowledge needed about weak learner
  - Provably effective, provided can consistently find rough rules of thumb

  → shift in mind set — goal now is merely to find classifiers barely better than random guessing
  - Versatile
  - Can be used with data that is textual, numeric, discrete, etc.
  - Has been extended to learning problems well beyond binary classification
- Caveats Caveats Caveats
  - Performance of AdaBoost depends on data and weak learner
  - Consistent with theory, AdaBoost can fail if:
    - weak classifiers are too complex → overfitting
    - weak classifiers are too weak ($\gamma_t \to 0$ too quickly) → underfitting → low margins → overfitting
    - empirically, AdaBoost seems especially susceptible to uniform noise

Based on Witten

Similar to Gradient Descent

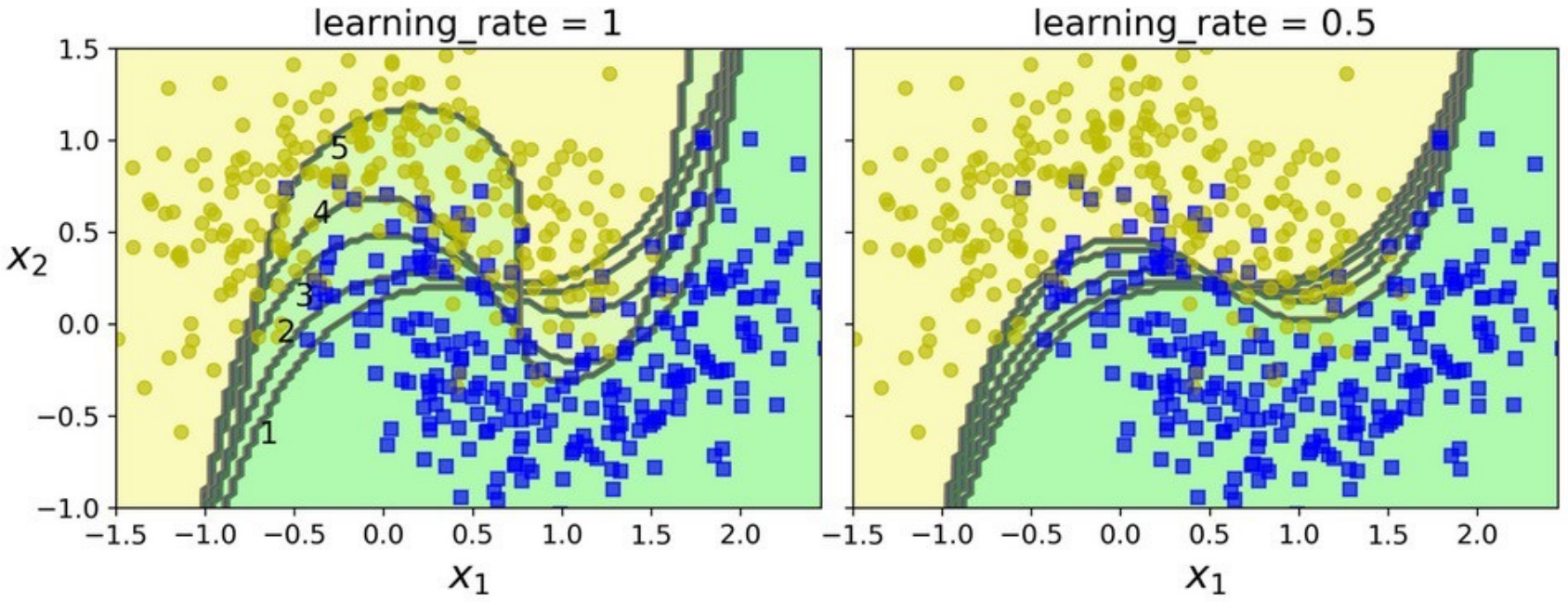


Figure 7-8. Decision boundaries of consecutive predictors

- Once all predictors are trained, the ensemble makes predictions very much like bagging or pasting, except that predictors have different weights depending on their overall accuracy on the weighted training set.
- As you can see, this sequential learning technique has some similarities with Gradient Descent, except that instead of tweaking a single predictor's parameters to minimize a cost function, AdaBoost adds predictors to the ensemble, gradually making it better.

Gradient Boosting

- Just like AdaBoost, Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor.
- However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method ***tries to fit the new predictor to the residual errors*** made by the previous predictor.
- The figure to the right represents the predictions of these three trees in the left column, and the ensemble's predictions in the right column.

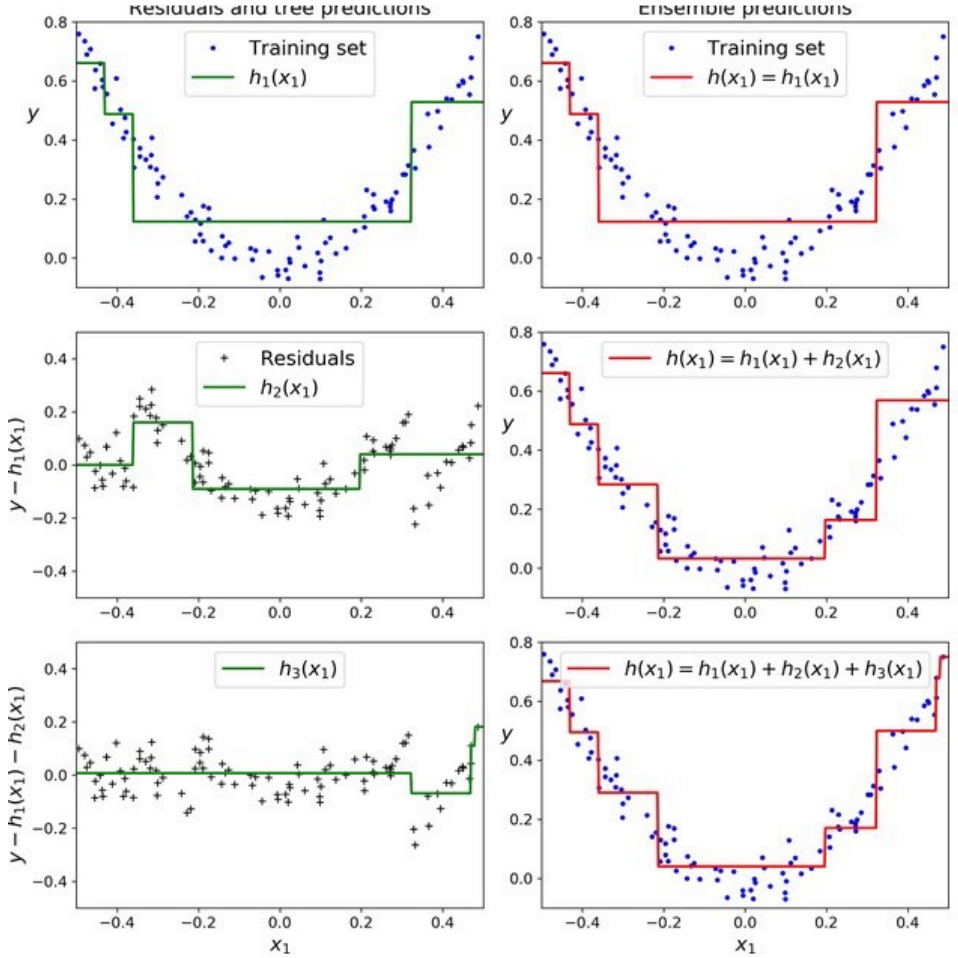


Figure 7-9. In this depiction of Gradient Boosting, the first predictor (top left) is trained normally, then each consecutive predictor (middle left and lower left) is trained on the previous predictor's residuals; the right column shows the resulting ensemble's predictions

Gradient Boosting

- First row: the ensemble has just one tree, so its predictions are exactly the same as the first tree's predictions.
- Second row: a new tree is trained on the residual errors of the first tree. On the right you can see that the ensemble's predictions are equal to the sum of the predictions of the first two trees.
- Third row: another tree is trained on the residual errors of the second tree. You can see that the ensemble's predictions gradually get better as trees are added to the ensemble.

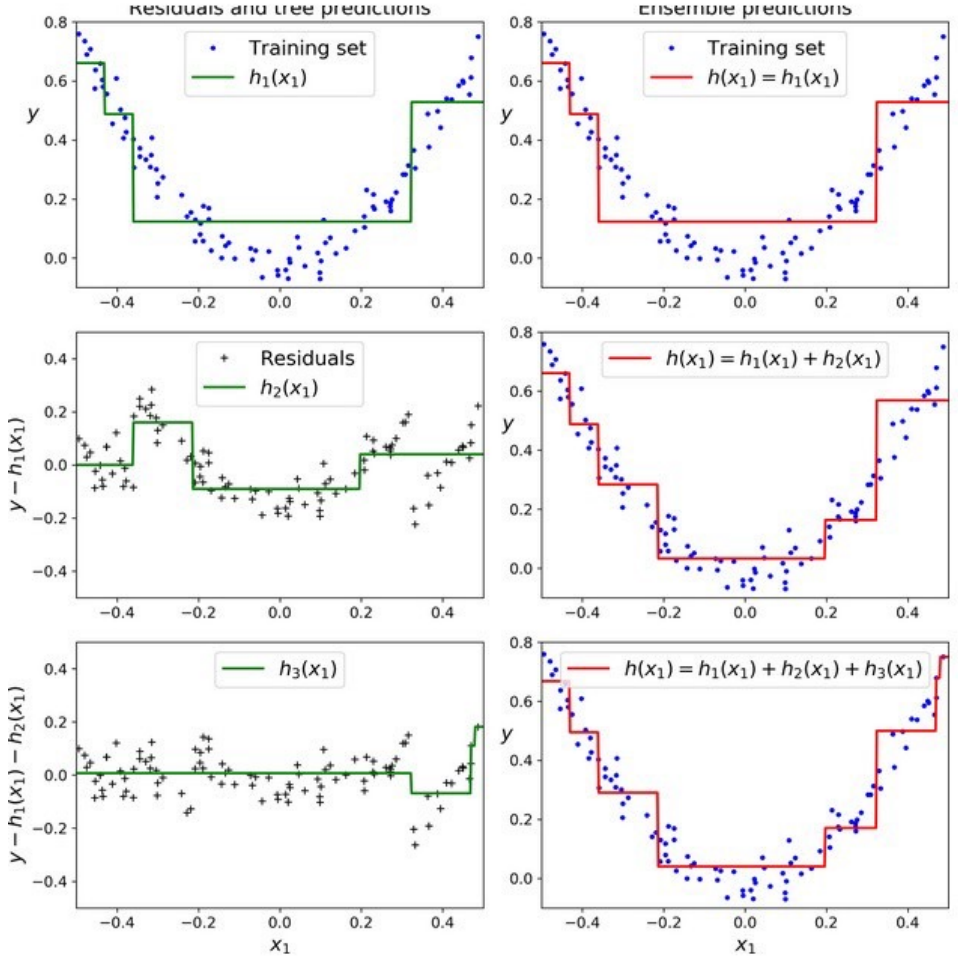


Figure 7-9. In this depiction of Gradient Boosting, the first predictor (top left) is trained normally, then each consecutive predictor (middle left and lower left) is trained on the previous predictor's residuals; the right column shows the resulting ensemble's predictions

Underfitting & Overfitting

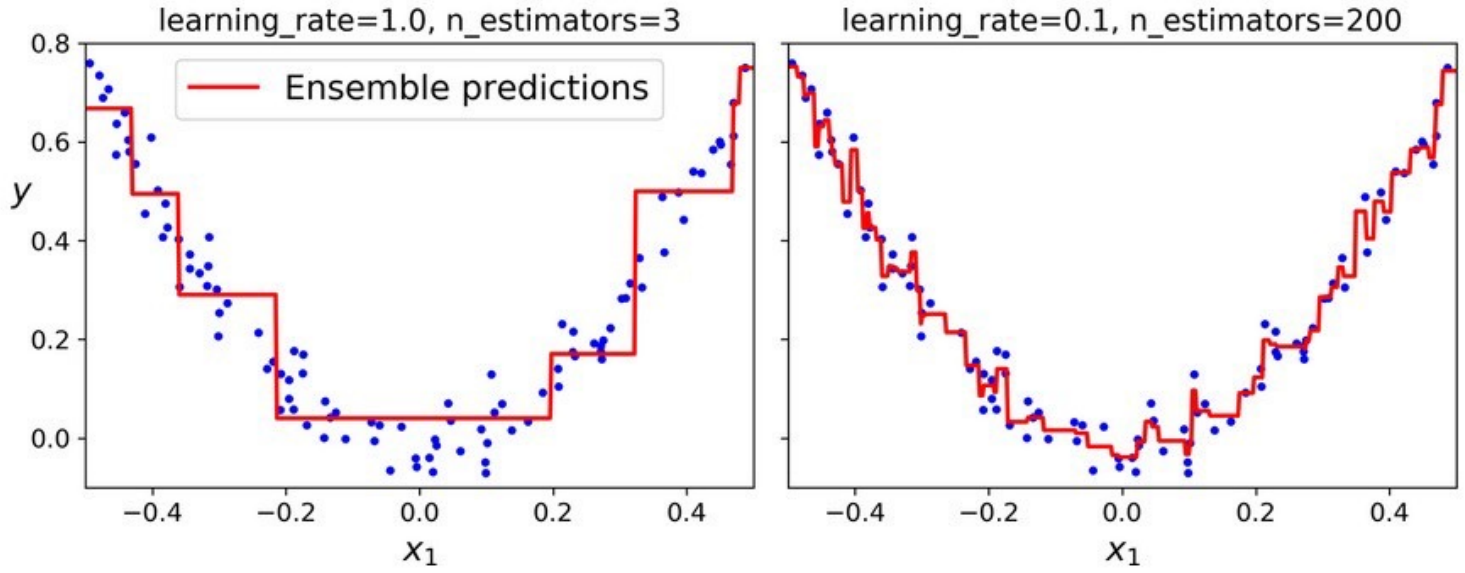


Figure 7-10. GBRT ensembles with not enough predictors (left) and too many (right)

- Two ensembles trained with a low learning rate: the one on the left does not have enough trees to fit the training set, while the one on the right has too many trees and overfits the training set.

Early Stopping

- A method that can be used to avoid overfitting is early stopping.

- With early stopping, you just stop training as soon as the validation error reaches the minimum. It is such a simple and efficient regularization technique that Geoffrey Hinton called it a “beautiful free lunch.”

- Early stopping measures the validation error at each stage of training to find the optimal number of trees, and finally trains another Gradient Boosted Regression Tree (GBRT) ensemble using the optimal number of trees.

Getting to the Best Model

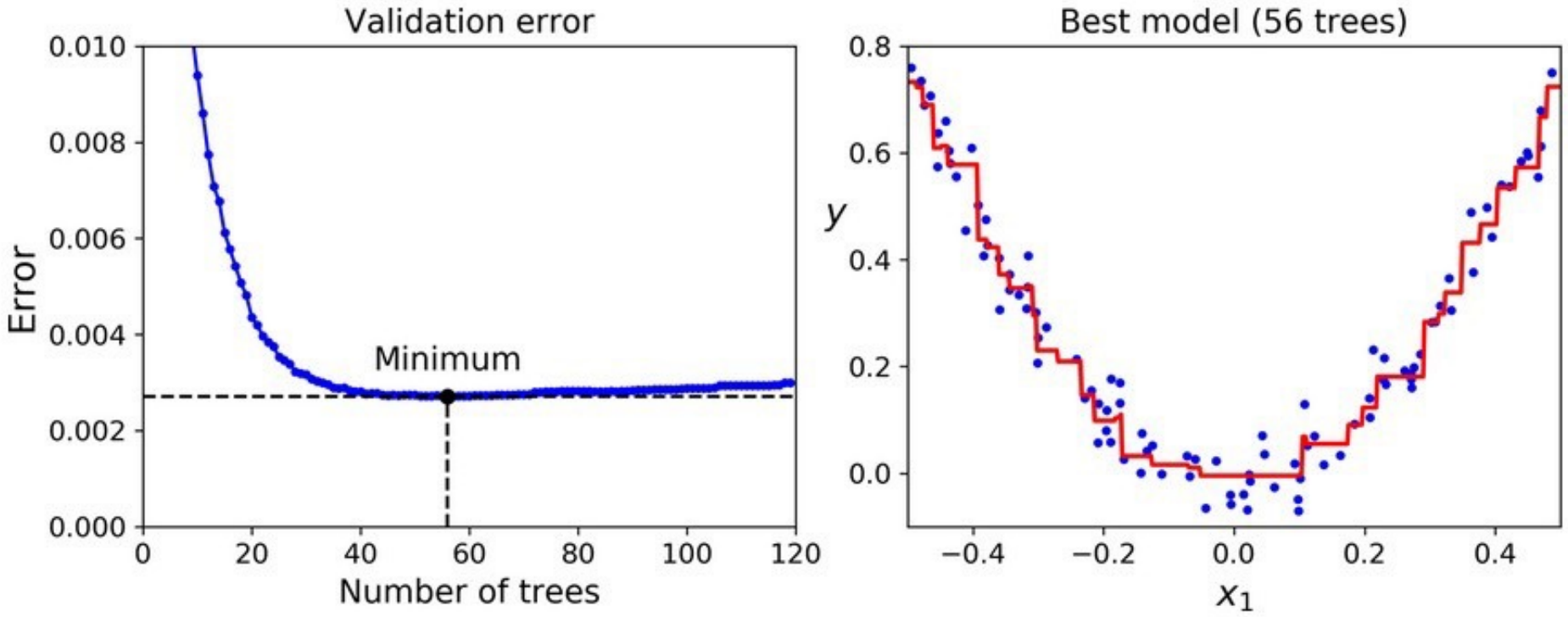


Figure 7-11. Tuning the number of trees using early stopping

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Random Forests

Professor Hamza F. Alsarhan
SEAS 8505

Random Forests

- A Random Forest is an **ensemble** of Decision Trees, generally trained via the bagging method (or sometimes pasting).
- When you are growing a tree in a Random Forest, at each node only a random subset of the features is considered for splitting.
- Possible to make trees even more random by also using random thresholds for each feature rather than searching for the best possible thresholds.
- Overview:
  - Bagging of Trees
  - Goal is to reduce correlation between trees
  - At every node, sample p features

Building a Random Forest

- Create a bootstrapped dataset
- Create a decision tree using the bootstrapped dataset, but only use a random subset of variables (or columns) at each step

Original Dataset

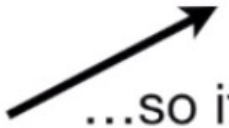
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Imagine that these 4 samples are the entire dataset that we are going to build a tree from…

Based on Josh Starmer, Statquest

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes



…so it's the first sample in our bootstrapped dataset.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes

Based on Josh Starmer, Statquest

Original Dataset

Chest Pain	Good Blood Circ	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No

…so it's the second sample in our bootstrapped dataset.

Based on Josh Starmer, Statquest

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

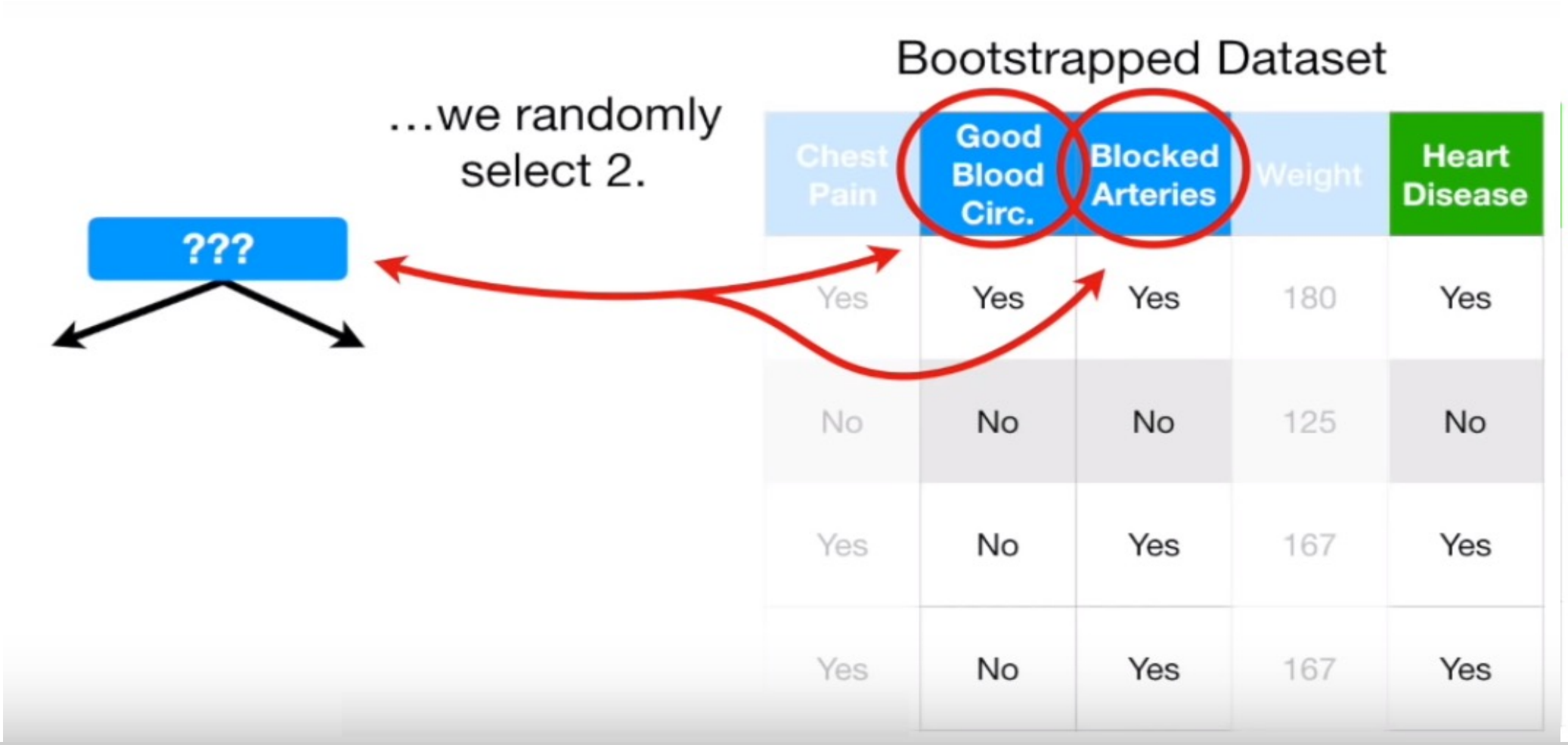
Based on Josh Starmer, Statquest

In this example, we will only consider 2 variables (columns) at each step.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Based on Josh Starmer, Statquest



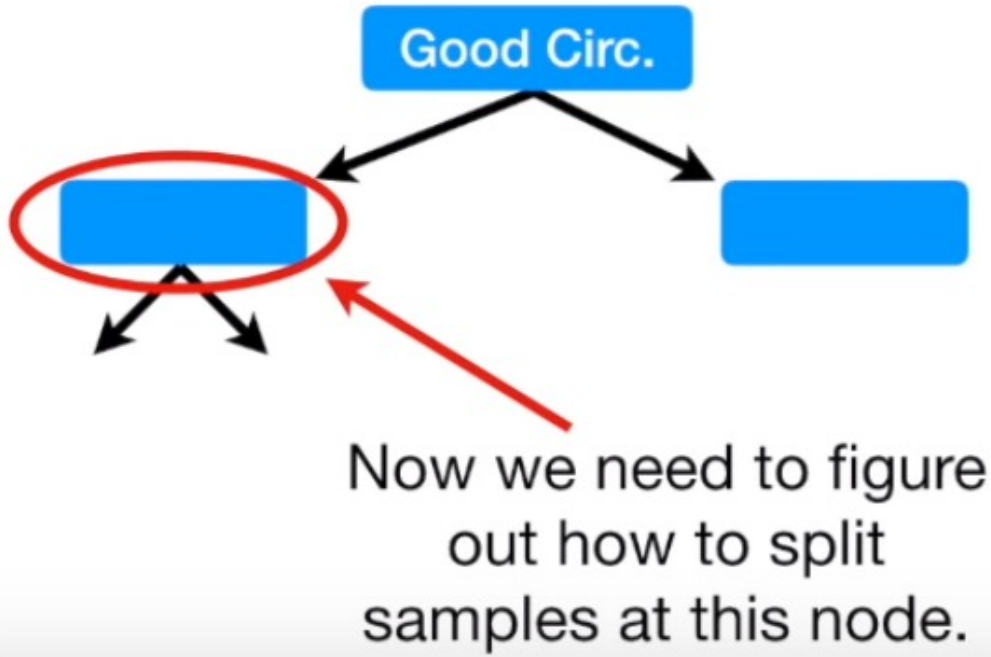
Based on Josh Starmer, Statquest

Just for the sake of the example, assume that **Good Blood Circulation** did the best job separating the samples.

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

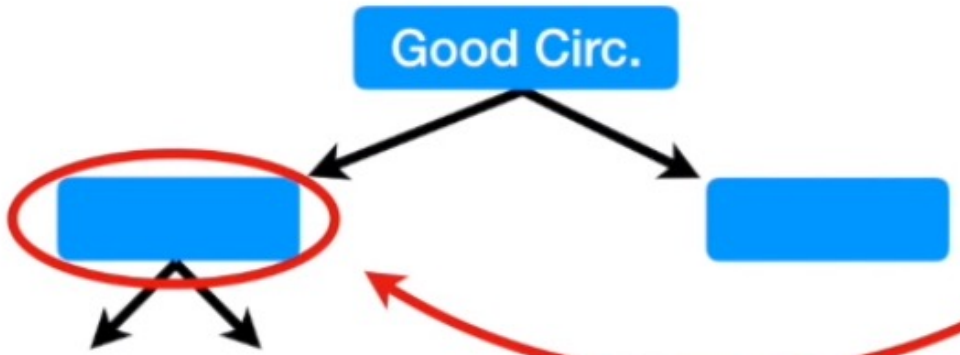
Based on Josh Starmer, Statquest



Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Based on Josh Starmer, Statquest

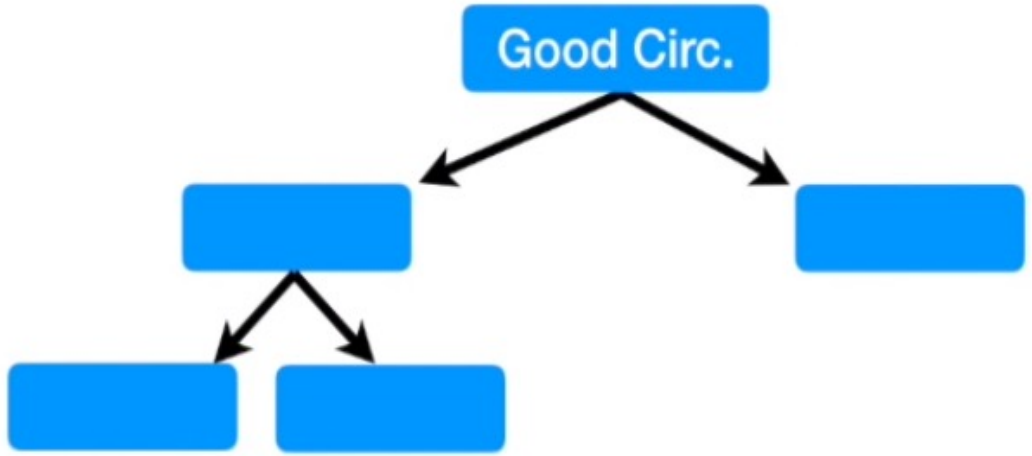


Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Just like for the root, we randomly select 2 variables as candidates, instead of all 3 remaining columns.

Based on Josh Starmer, Statquest

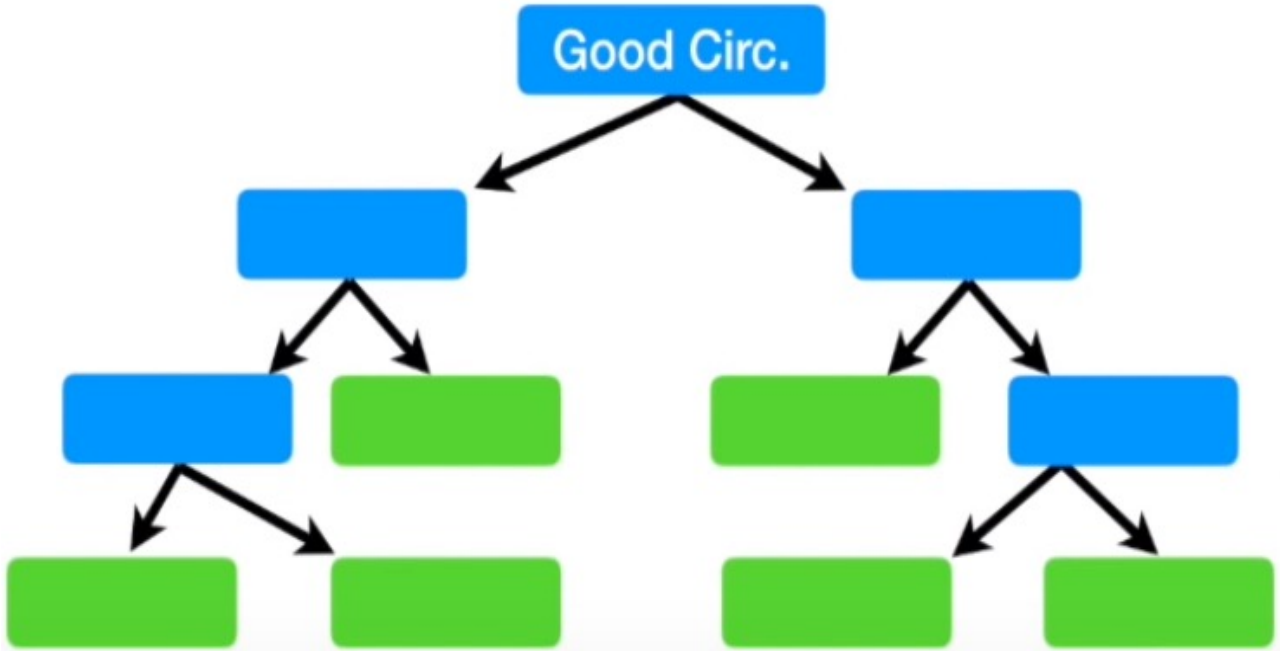


Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
Yes	No	Yes	167	Yes

And we just build the tree as usual, but only considering a random subset of variables at each step.

Based on Josh Starmer, Statquest



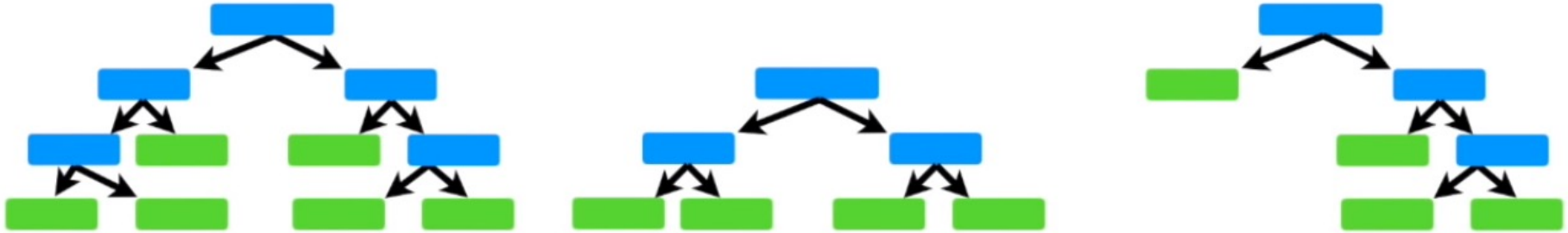
Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

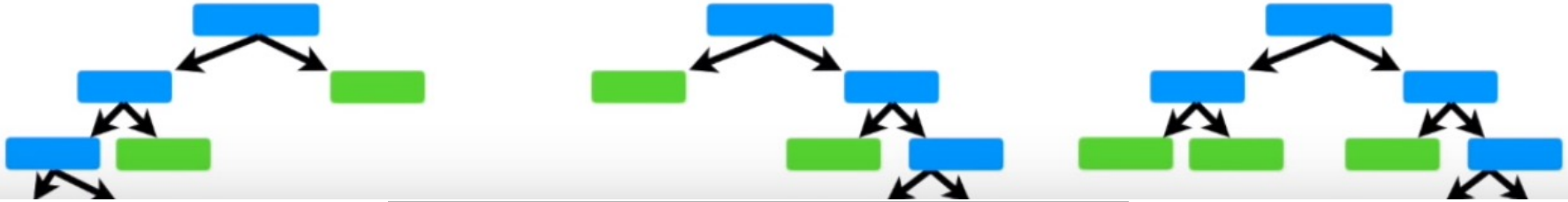
Based on Josh Starmer, Statquest

Now go back to Step 1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.

Using a bootstrapped sample and considering only a subset of the variables at each step results in a wide variety of trees.



The variety is what makes random forests more effective than individual decision trees.



Ideally, you'd do this 100's of times

Now that we've created a Random Forest – How do we use it?

Based on Josh Starmer, Statquest

Typically, about 1/3 of the original data does not end up in the bootstrapped dataset.

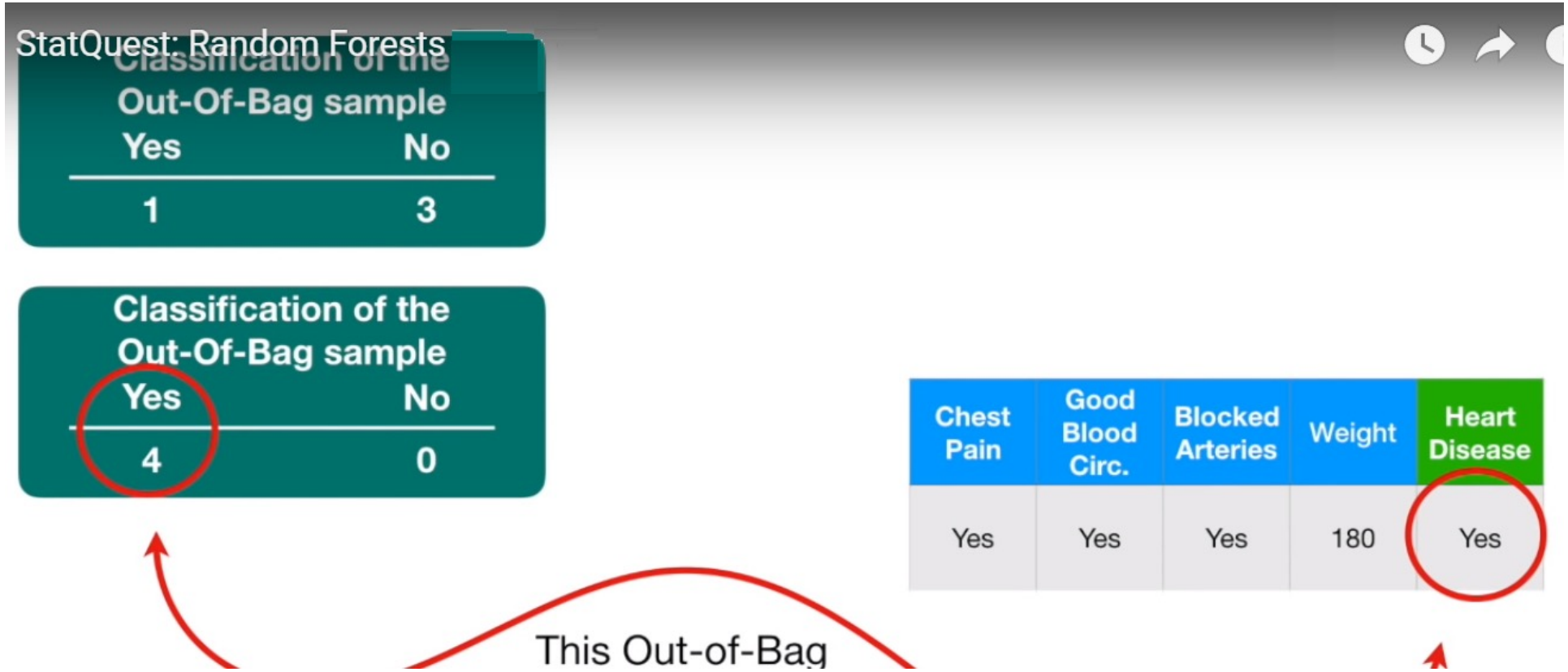
Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Based on Josh Starmer, Statquest



Based on Josh Starmer, Statquest

Classification of the Out-Of-Bag sample	
Yes	No
1	3

Classification of the Out-Of-Bag sample	
Yes	No
4	0

Classification of the Out-Of-Bag sample	
Yes	No
3	1

Ultimately, we can measure how accurate our random forest is by the proportion of Out-Of-Bag samples that were correctly classified by the Random Forest.

The proportion of Out-Of-Bag samples that were *incorrectly* classified is the "**Out-Of-Bag Error**"

Based on Josh Starmer, Statquest

Feature Importance

- Scikit-Learn measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity by, on average (across all trees in the forest).

- More precisely, it is a weighted average, where each node's weight is equal to the number of training samples that are associated with it.

- Scikit-Learn computes this score automatically for each feature after training, then it scales the results so that the sum of all importances is equal to 1.

Feature Importance

- Random forests make it easy to measure the relative importance of each feature.

- Random Forests are very handy to get a quick understanding of what features actually matter, in particular if you need to perform feature selection.

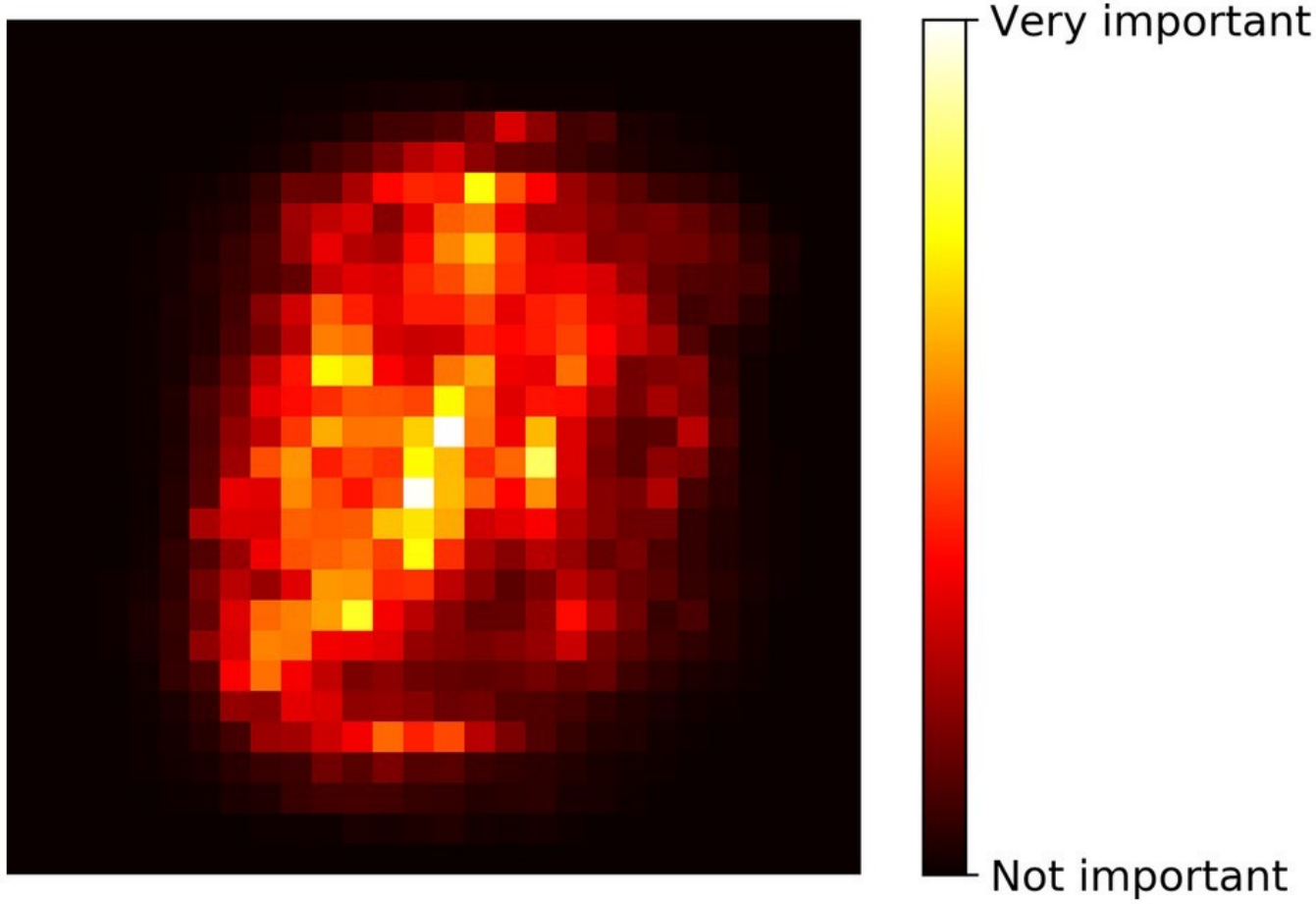


Figure 7-6. MNIST pixel importance (according to a Random Forest classifier)

Randomization and Random Forests

- Can randomize learning algorithm instead of input
- 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 in a nearest-neighbor classifier
  - Bagging does not work with stable classifiers such as nearest neighbor classifiers
- Can be combined with bagging
  - When using decision trees, this yields the famous random forest method for building ensemble classifiers

Based on Witten

Advantages of Random Forests

- Handles both Classification and Regression
- Handles missing values and maintains accuracy for missing data
- Somewhat immune to overfitting
- Handles large datasets with higher dimensionality

Disadvantages of Random Forests

- Although good for classification, not as accurate on regression
- “Black box”
- User has little control over what model does (random seeds, parameters)
- Relatively Slow during Testing so not good for real-time operation

Extremely Randomized Trees

- A forest of such extremely random trees is called an Extremely Randomized Trees ensemble (or Extra-Trees for short).
- This technique trades more bias for a lower variance.
- It also makes Extra-Trees much faster to train than regular Random Forests, because finding the best possible threshold for each feature at every node is one of the most time-consuming tasks of growing a tree.
- Application Tip: It is hard to tell in advance whether a RandomForestClassifier will perform better or worse than an ExtraTreesClassifier (for training extremely randomized trees).
  - Generally, the only way to know is to try both and compare them using cross-validation (tuning the hyperparameters using grid search).

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

What to Expect for the Midterm

Professor Hamza F. Alsarhan
SEAS 8505

What to Expect for the Midterm (subject to change)

- Roughly 40 questions
  - Approximately 10 questions per lecture (lectures 1-4)
- Multiple Choice and True/False
- Some mathematical calculations, such as performance metrics (no math derivations)
- No coding during the exam
- Heavy focus on the lecture slides
  - Recommend starting with lecture slides and then going to course readings for further details and deeper understanding of those topics
- You are permitted to bring a single, 8.5”x11”, reference sheet (front and back), any format (***must show to camera at beginning of exam***)
  - Recommend spending the time to review the course material and to create this reference sheet

Reminders for Test Requirements

The online test environment should mimic the in-class test environment, and conform to the following:

- Test Area
  - Sit at a clean desk or table (not on a bed or couch).
  - Ensure that lighting in the room is bright enough to be considered "daylight" quality. Overhead lighting is preferred; however, if overhead is not possible, the source of light should not be behind you.
  - Clear the desk or table of all materials: Students can have a single sheet of 8.5 x 11 inch paper with handwritten or typed notes on the front and back only
  - Use one computer monitor only; dual monitors are not permitted.
  - Have no writing on desk or walls or any notes or writing saved as your computer desktop background.

Test Area requirements (continued)

- No software other than HonorLock and Blackboard should be open unless permitted by the instructor.
- Close all other programs and/or windows on the testing computer before logging in to the proctored test environment.
- Do not have a radio or television playing in the background.
- Do not talk to anyone else – you may not communicate with others by any means.
- No other persons except the test-taker is permitted in the room during testing.
- If a calculator is required, you may use the calculator that comes with the Mac or the Windows operating system only. No calculators will be allowed in the testing area.

Reminder for Behavior During Midterm

- Dress as if in a public setting
- You will be allowed to take a brief bathroom break during the exam. You should not leave the room for any other reason during the exam. Do not take the computer into another room to finish testing (exam must be completed in the same room as the “Exam Environment View”).
- No headsets, ear plugs, or similar audio devices are permitted
- Cell phones are not permitted in the exam room. The only exception is if a student needs to contact HonorLock at the beginning of an exam. Once reconnected students will be required to remove the cell phone from the environment.
- Your entire face must be visible throughout the exam. Being out of camera view is considered an exam violation. You should check the thumbnail at the top of the screen to confirm.
- Your photo ID should be readable. If it is not, you may be contacted via email by HonorLock to resend your photo ID photo. Failure to comply is considered a violation.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Homework Overview

Professor Hamza F. Alsarhan
SEAS 8505

This Week

- Reading:
  - Chapter 7 in the textbook
- HW #4 (Run/Write Python Script in Google Colab first, and then answer the homework questions)
- Discussion #4
- Take the practice exam by 9:00 am ET on Saturday
- Study for the midterm

- Reminder: No extensions provided. Start assignments early!

Next Steps

- Come to office hours with any questions you may have.
- Work on your HW and Discussion and submit them by 9:00 am ET on Saturday.
- See you next class!

Thank you!