STEVENS
INSTITUTE OF TECHNOLOGY
1870

Artificial Intelligence

Computer Science, CS541 - A

Jonggi Hong

Announcements

- Midterm survey
  - Stevens system
  - Google form (https://forms.gle/8yhd2AcXw6E5Zyaz7)
- HW #3
  - Due 11:59 pm, Tuesday, April 4
- Project midterm report
  - Due 11:59 pm, Tuesday, March 28

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Recap

Bayesian Network - Sampling

Bayes' Nets

- Representation
- Conditional Independences
- Probabilistic Inference
  - Enumeration (exact, exponential complexity)
  - Variable elimination (exact, worst-case exponential complexity, often better)
  - Probabilistic inference is NP-complete
  - Sampling (approximate)
- Learning Bayes' Nets from Data

Sampling

- Sampling is a lot like repeated simulation.
  - Predicting the weather, basketball games, …
- Basic idea
  - Draw N samples from a sampling distribution S.
  - Compute an approximate posterior probability.
  - Show this converges to the true probability P.
- Why sample?
  - Learning: get samples from a distribution you don’t know
  - Inference: getting a sample is faster than computing the right answer (*e.g.*, with variable elimination)

Sampling

- Sampling from given distribution
  - Step 1: Get sample *u* from uniform distribution over [0, 1)
    - E.g., random() in python
  - Step 2: Convert this sample *u* into an outcome for the given distribution by having each outcome associated with a sub-interval of [0,1) with sub-interval size equal to probability of the outcome

- Example

C	P(C)
red	0.6
green	0.1
blue	0.3

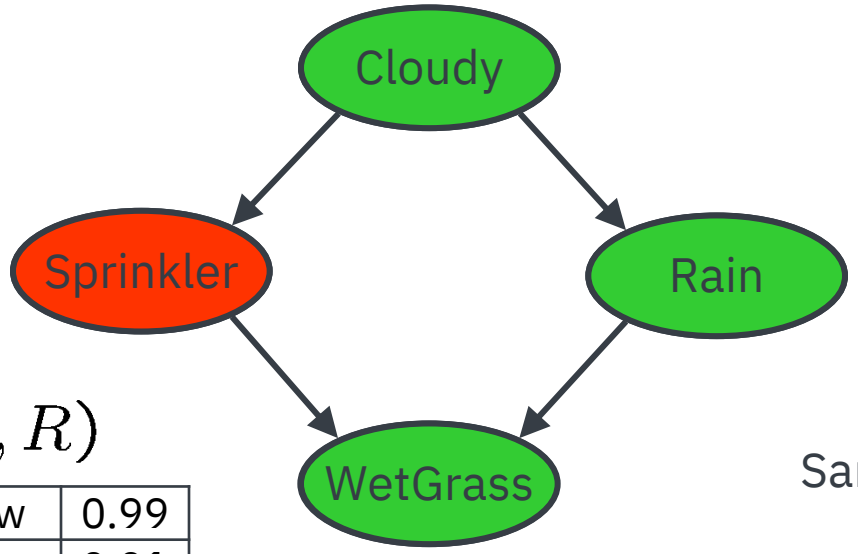
$$0 \leq u < 0.6, \rightarrow C = red$$
$$0.6 \leq u < 0.7, \rightarrow C = green$$
$$0.7 \leq u < 1, \rightarrow C = blue$$

- If random() returns $u = 0.83$, then our sample is C = blue
- E.g., after sampling 8 times:
  - 5 red, 1 green, 2 blue

Prior Sampling

$P(C)$

+c	0.5
-c	0.5



$P(S|C)$

+c	+s	0.1
	-s	0.9
-c	+s	0.5
	-s	0.5

$P(R|C)$

+c	+r	0.8
	-r	0.2
-c	+r	0.2
	-r	0.8

$P(W|S,R)$

+s	+r	+w	0.99
		-w	0.01
	-r	+w	0.90
		-w	0.10
-s	+r	+w	0.90
		-w	0.10
	-r	+w	0.01
		-w	0.99

Samples:

+c, -s, +r, +w

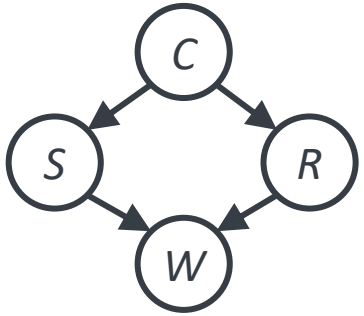
-c, +s, -r, +w

…

Compute $P(Q|e_1 \ldots e_k)$ from samples.

Rejection Sampling

- Let's say we want P(C)
  - No point keeping all samples around
  - Just tally counts of C as we go



- Let' s say we want P(C| +s)
  - Same thing: tally C outcomes, but ignore(reject) samples which don' t have S=+s.
  - This is called rejection sampling.
  - It is also consistent for conditional probabilities (i.e., correct in the limit).

+c, -s, +r, +w
+c, +s, +r, +w
-c, +s, +r, -w
+c, -s, +r, +w
-c, -s, -r, +w

Rejection Sampling

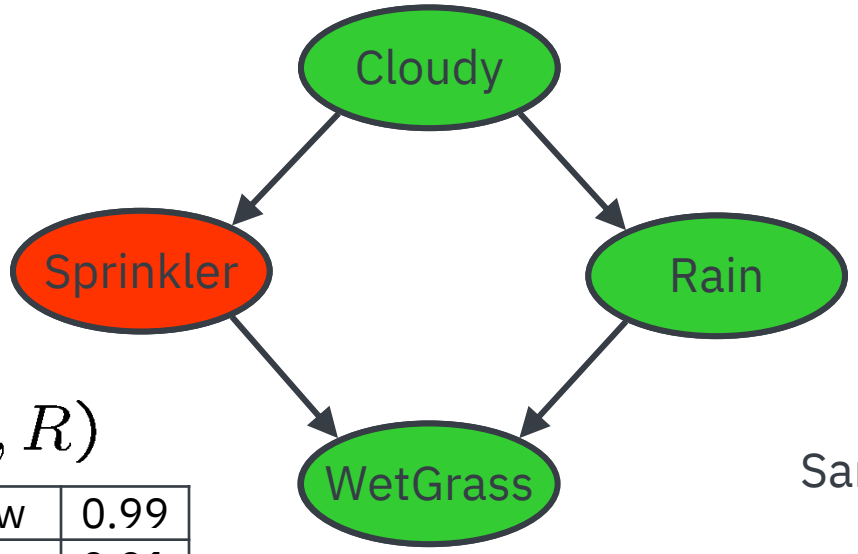
$P(C)$

+c	0.5
-c	0.5

Compute **P(+r|-s)** from samples.

$P(S|C)$

+c	+s	0.1
	-s	0.9
-c	+s	0.5
	-s	0.5



$P(R|C)$

+c	+r	0.8
	-r	0.2
-c	+r	0.2
	-r	0.8

$P(W|S,R)$

+s	+r	+w	0.99
		-w	0.01
	-r	+w	0.90
		-w	0.10
-s	+r	+w	0.90
		-w	0.10
	-r	+w	0.01
		-w	0.99

Samples:

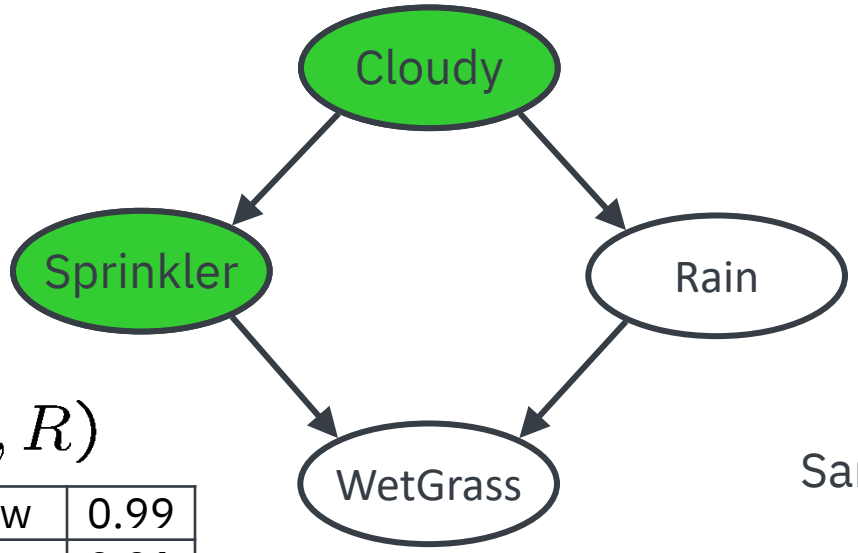
+c, -s, +r, +w

Rejection Sampling

$P(C)$

+c	0.5
-c	0.5

Compute **P(+r|-s)** from samples.



$P(S|C)$

+c	+s	0.1
	-s	0.9
-c	+s	0.5
	-s	0.5

$P(R|C)$

+c	+r	0.8
	-r	0.2
-c	+r	0.2
	-r	0.8

$P(W|S,R)$

+s	+r	+w	0.99
		-w	0.01
	-r	+w	0.90
		-w	0.10
-s	+r	+w	0.90
		-w	0.10
	-r	+w	0.01
		-w	0.99

Samples:

+c, -s, +r, +w

~~-c, +s~~

...

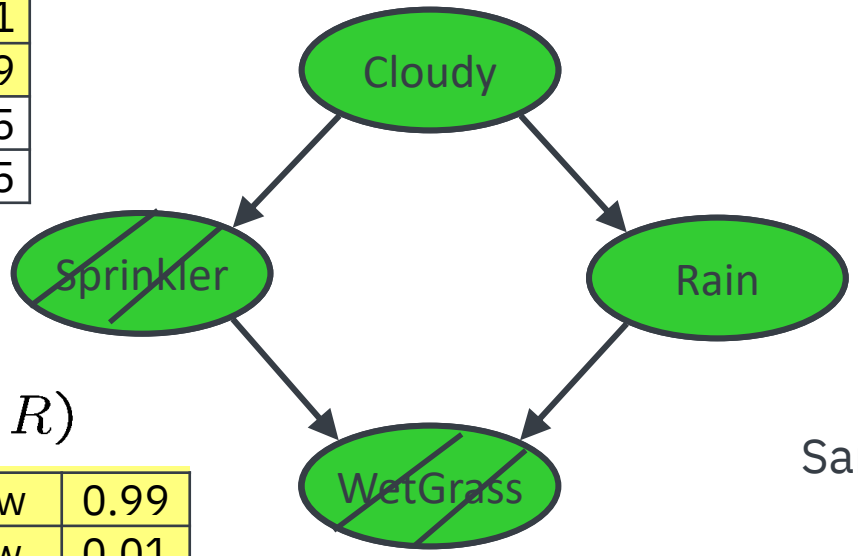
Likelihood Weighting

$P(C)$

+c	0.5
-c	0.5

$P(S|C)$

+c	+s	0.1
	-s	0.9
-c	+s	0.5
	-s	0.5



$P(R|C)$

+c	+r	0.8
	-r	0.2
-c	+r	0.2
	-r	0.8

$P(W|S,R)$

+s	+r	+w	0.99
		-w	0.01
	-r	+w	0.90
		-w	0.10
-s	+r	+w	0.90
		-w	0.10
	-r	+w	0.01
		-w	0.99

Samples:

+c, +s, +r, +w

…

$$w = 1.0 \times 0.1 \times 0.99$$

Likelihood Weighting: Example

weight

+c, +s, +r, +w : **0.099**
+c, +s, +r, +w : **0.099**
-c, +s, +r, +w : **0.495**
+c, +s, +r, +w : **0.099**
-c, +s, -r, +w : **0.450**

Sample	Count/N	Weight	Joint (=Count/N* Weight)
+c, +s, +r, +w	3/5	0.099	0.0594
-c, +s, +r, +w	1/5	0.495	0.099
+c, +s, -r, +w	0/5	-	0
-c, +s, -r, +w	1/5	0.450	0.09

What is P(+r, +s, +w)? P(+r, +s, +w)=0.0594+0.099=0.1584

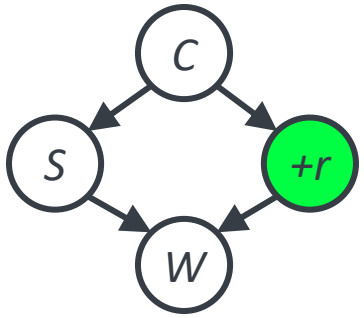
What is P(+r | +s, +w)? P(+s, +w) = 0.0594 + 0.099 + 0 + 0.09 = 0.2484

P(+r | +s, +w)= P(+r, +s, +w)/P(+s, +w)

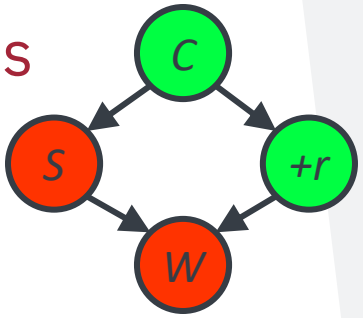
= 0.1584 / 0.2484 = 0.638

Gibbs Sampling Example: P(S | +r)

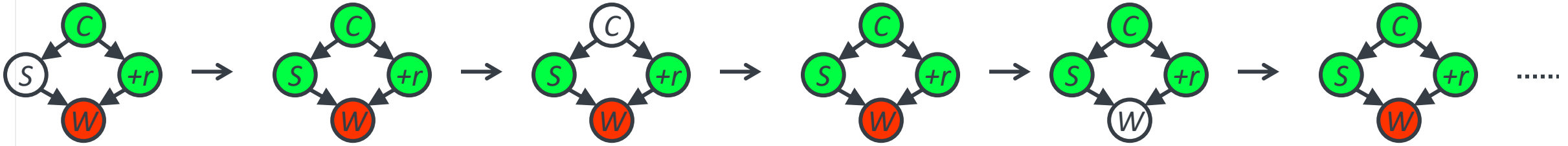
- Step 1: Fix evidence
  - R = +r



- Step 2: Initialize other variables
  - Randomly



- Steps 3: Repeat
  - Choose a non-evidence variable X
  - Resample X from P(X | all other variables)



Sample from $P(S|+c,-w,+r)$ Sample from $P(C|+s,-w,+r)$ Sample from $P(W|+s,+c,+r)$

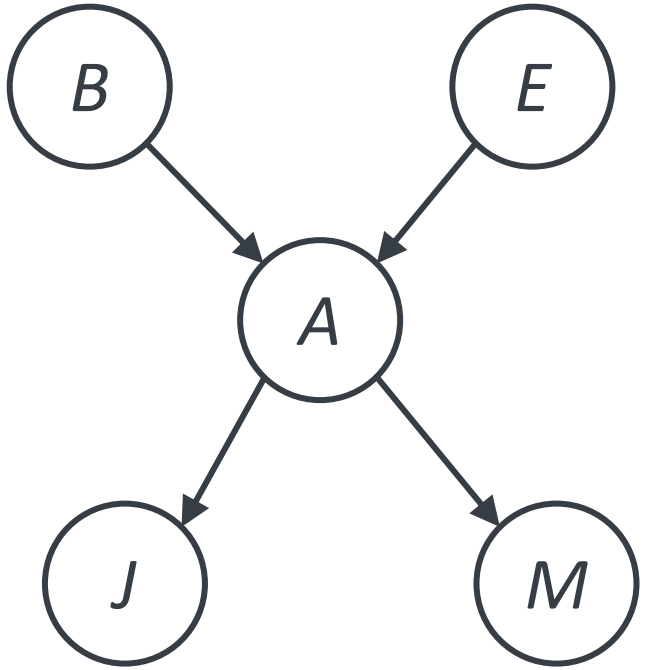
Gibbs Sampling

- How do we get P(+s | +r) with Gibbs sampling?
  - Just tally counts of +s as we go (as you did with prior sampling)

Machine Learning

Burglars and Earthquakes

Earthquake	Burglars	Alarm	#
0	0	0	1000
0	0	1	10
0	1	0	20
0	1	1	100
1	0	0	200
1	0	1	50
1	1	0	0
1	1	1	5



	P(+a|E, B)
+e, +b	1
+e, -b	0.2
-e, +b	0.83
-e, -b	0.01

Model-Based Classification

- Model-based approach
  - Build a model (e.g., Bayes' net) where both the label and features are random variables
  - Instantiate any observed features
  - Query for the distribution of the label conditioned on the features

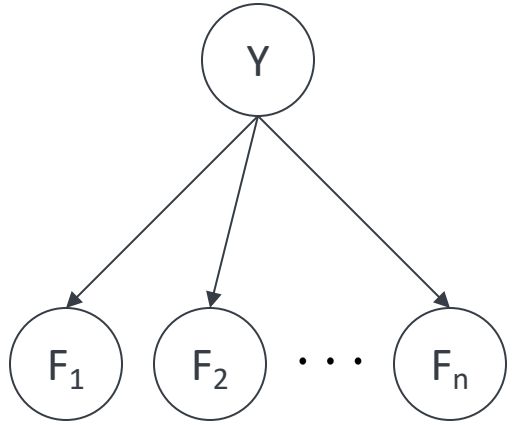
- Challenges
  - What structure should the BN have?
  - How should we learn its parameters?

Naïve Bayes for Digits

- Naïve Bayes: Assume all features are independent effects of the label
- Simple digit recognition version:
  - One feature (variable) F_{ij} for each grid position <i,j>
  - Feature values are on / off, based on whether intensity is more or less than 0.5 in underlying image
  - Each input maps to a feature vector, e.g.

$$\rightarrow \langle F_{0,0} = 0 \;\; F_{0,1} = 0 \;\; F_{0,2} = 1 \;\; F_{0,3} = 1 \;\; F_{0,4} = 0 \;\; \ldots F_{15,15} = 0 \rangle$$

  - Here: lots of features, each is binary valued

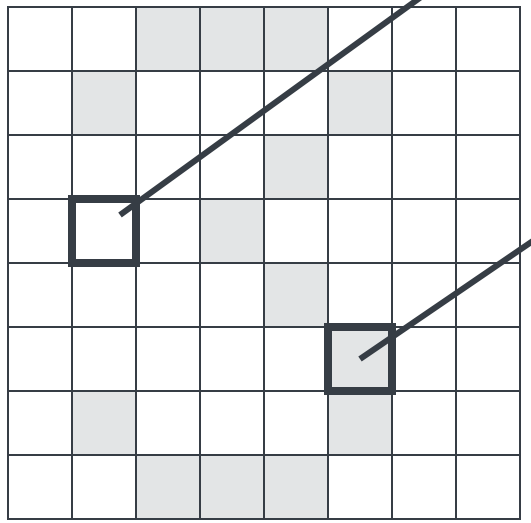


- Naïve Bayes model: $P(Y|F_{0,0} \ldots F_{15,15}) \propto P(Y) \prod_{i,j} P(F_{i,j}|Y)$
- What do we need to learn?

Example: Conditional Probabilities

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5} = on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

Inference for Naïve Bayes

- Goal: compute posterior distribution over label variable Y
  - Step 1: get joint probability of label and evidence for each label

$$P(Y, f_1 \dots f_n) = \begin{bmatrix} P(y_1, f_1 \dots f_n) \\ P(y_2, f_1 \dots f_n) \\ \vdots \\ P(y_k, f_1 \dots f_n) \end{bmatrix} \Rightarrow \begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}$$

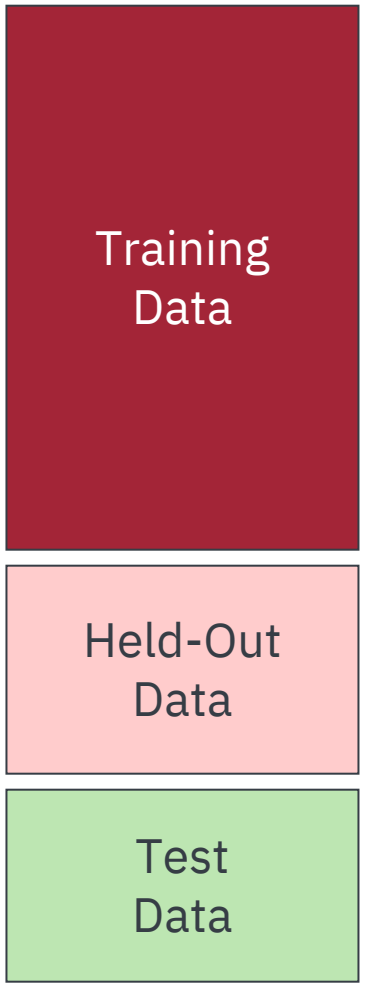
$$\Rightarrow \frac{\begin{bmatrix} P(y_1) \prod_i P(f_i|y_1) \\ P(y_2) \prod_i P(f_i|y_2) \\ \vdots \\ P(y_k) \prod_i P(f_i|y_k) \end{bmatrix}}{P(f_1 \dots f_n)} \Rightarrow P(Y|f_1 \dots f_n)$$

  - Step 2: sum to get probability of evidence
  - Step 3: normalize by dividing Step 1 by Step 2

Training and Testing

Important Concepts

- Data: labeled instances, *e.g.,* emails marked spam/ham
  - Training set
  - Held out set
  - Test set
- Features: attribute-value pairs which characterize each x
- Experimentation cycle
  - Learn parameters (*e.g.,* model probabilities) on training set
  - (Tune hyperparameters on held-out set)
  - Compute accuracy of test setEvaluation
  - Accuracy: fraction of instances predicted correctly
- Overfitting and generalization
  - Want a classifier which does well on *test* data
  - Overfitting: fitting the training data very closely, but not generalizing well
  - We'll investigate overfitting and generalization formally.



Example: Overfitting

$P(\text{features}, C = 2)$

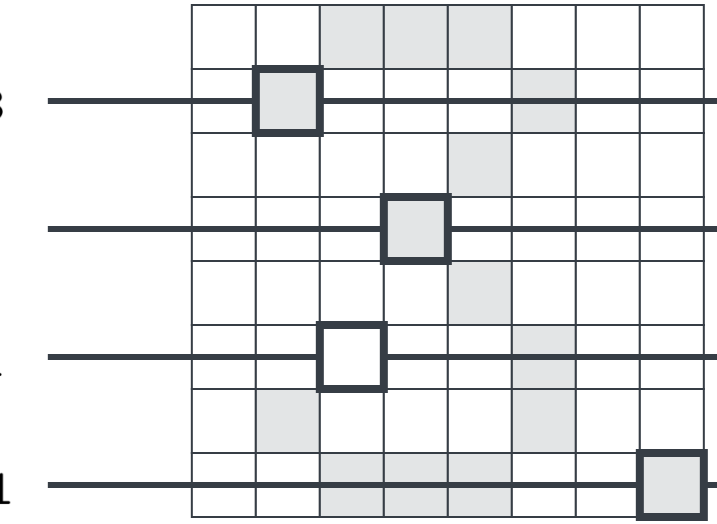
$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$



$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

2 wins!!

Laplace Smoothing

- Laplace's estimate:
  - Pretend you saw every outcome once more than you actually did.

$$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]}$$

$$= \frac{c(x)+1}{N+|X|}$$

$$P_{ML}(X) =$$

$$P_{LAP}(X) =$$

Laplace Smoothing

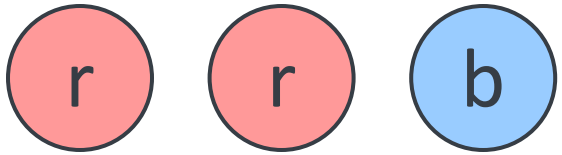
- Laplace's estimate (extended):
  - Pretend you saw every outcome k extra times

$$P_{LAP,k}(x) = \frac{c(x) + k}{N + k|X|}$$

  - What's Laplace with k = 0?
  - k is the **strength** of the prior

- Laplace for conditionals:
  - Smooth each condition independently:

$$P_{LAP,k}(x|y) = \frac{c(x,y) + k}{c(y) + k|X|}$$



$$P_{LAP,0}(X) =$$

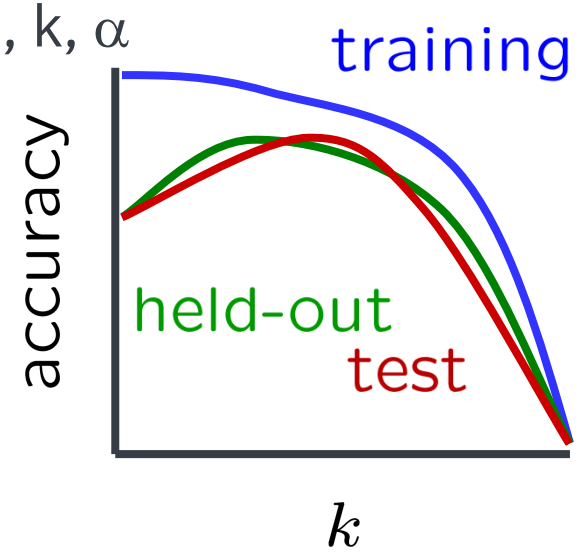
$$P_{LAP,1}(X) =$$

$$P_{LAP,100}(X) =$$

Tuning

Tuning on Held-Out Data

- Now we've got two kinds of unknowns
  - Parameters: the probabilities P(X|Y), P(Y)
  - Hyperparameters: e.g., the amount / type of smoothing to do, k, α



- What should we learn where?
  - Learn parameters from training data
  - Tune hyperparameters on different data
    - Why?
  - For each value of the hyperparameters, train and test on the held-out data
  - Choose the best value and do a final test on the test data

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Machine Learning

Chapter 18

What is Machine Learning

- Definition

  A computer program is said to **learn** from **experience** E with respect to some class of **tasks** T and **performance measures** P, if its performance at tasks in T, as measured by P, improves with experience E.

  (T Mitchell, 2007)

Examples

- Handwriting recognition
  - Task: recognize and classify handwritten letters and digits
  - Experience: database of pre-classified letters and digits
  - Performance measure: percent of letters/digits correctly classified
- Game playing problem
  - Task: playing the game
  - Experience: playing practice games against itself (self-play)
  - Performance measure: percentage of games won against an opponent
- Autonomous driving
  - Task: driving on a public four-lane highway using vision sensors
  - Experience: sequence of images and steering commands recorded while observing a human driver
  - Performance: Average distance traveled before an error was made (as judged by human overseer)

Common Learning Tasks

- Supervised Classification
  - Given a set of pre-classified training examples, classify a new instance
- Unsupervised Learning
  - Find natural classes for examples
- Reinforcement Learning
  - Determine what to do based on rewards and punishments
- Transfer Learning
  - Learning from an expert
- Active Learning
  - Actively seek to learn

Representations

- The representation of what we are learning is crucial.
  - It determines how the learning algorithm will work.
- Common representations:
  - Linear weighted polynomials
  - Propositional logic
  - First order logic
  - Bayes nets
  - …

Supervised Learning

Supervised Learning

- Given a training set of examples of the form (x, f(x))
  - x is the input, f(x) is the output.
- Return a function h that approximates f
  - h is the **hypothesis**.

Attribute

Sky	Air Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

x

f(x)

Hypothesis Representation

- We need a **hypothesis representation** for the problem.
  - A reasonable candidate for our example is a conjunction of constraints.

Sky	Air Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

x f(x)

Hypothesis Representation

- We need a **hypothesis representation** for the problem.
  - Vector of 6 constraints specifying the values of the 6 attributes
    - ? to denote that any value is acceptable
    - Specify a single required value (*e.g.*, Warm)
    - 0 to specify that no value is acceptable

Sky	Air Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

x f(x)

Hypothesis Representation

- We need a **hypothesis representation** for the problem.
  - If some instance satisfies all constraints of hypothesis h, then h classifies x as a positive example (h(x)=1)
  - h=<?,Cold,High,?,?,?> represents a hypothesis that someone enjoys her favorite sport only on cold days with high humidity.

Hypothesis Space

- Most general hypothesis
  - <?, ?, ?, ?, ?, ?> (every day is a positive example)
- Most specific hypothesis
  - <0, 0, 0, 0, 0, 0> (no day is a positive example)
- Hypothesis space, H
  - Set of all possible hypothesis that the learner may consider regarding the target concept
  - Can think of learning as a search through a hypothesis space

Learning Problem

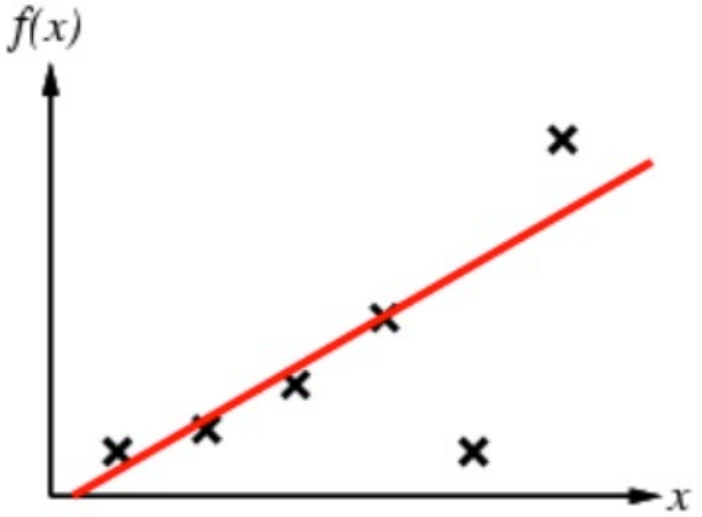
- Our goal is to find a good hypothesis.
- Why is this hard?

Inductive Learning Hypothesis

- Any hypothesis found to approximate the target function well **over a sufficiently large set of training examples** will also approximate the target function well over any unobserved examples.
  - I.e., the training set needs to ”represent” the whole domain (which may be infinite).

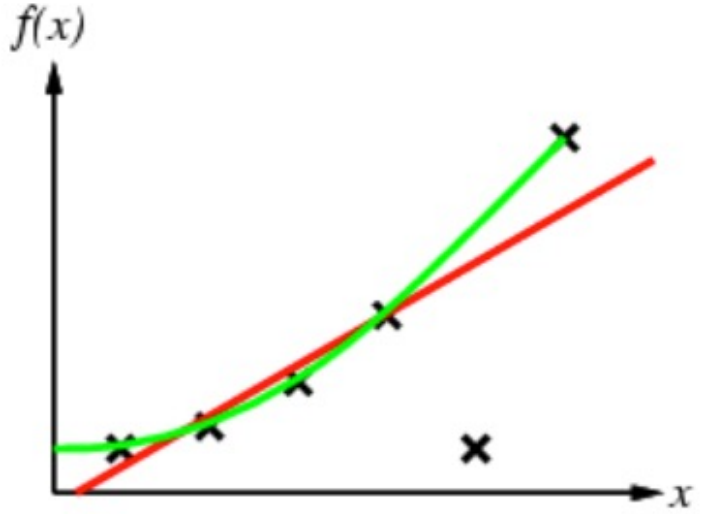
Supervised Learning

- Construct/adjust h to agree with f on training set
  - h is consistent if it agrees with f on all examples.
  - e.g., curve fitting



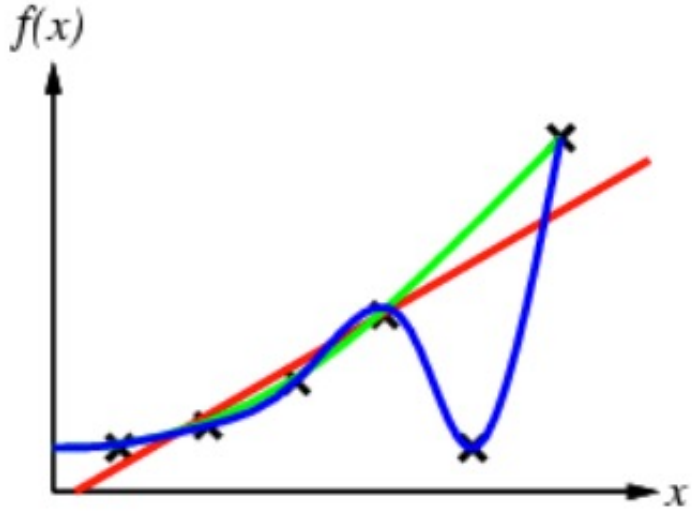
Supervised Learning

- Construct/adjust h to agree with f on training set
  - h is consistent if it agrees with f on all examples.
  - e.g., curve fitting



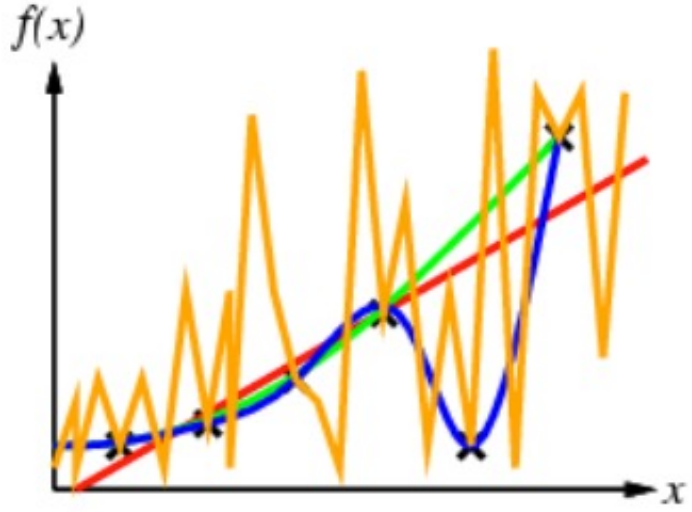
Supervised Learning

- Construct/adjust h to agree with f on training set
  - h is consistent if it agrees with f on all examples.
  - e.g., curve fitting



Supervised Learning

- Construct/adjust h to agree with f on training set
  - h is consistent if it agrees with f on all examples.
  - e.g., curve fitting



Ockham's Razor: prefer the simplest hypothesis consistent with the data

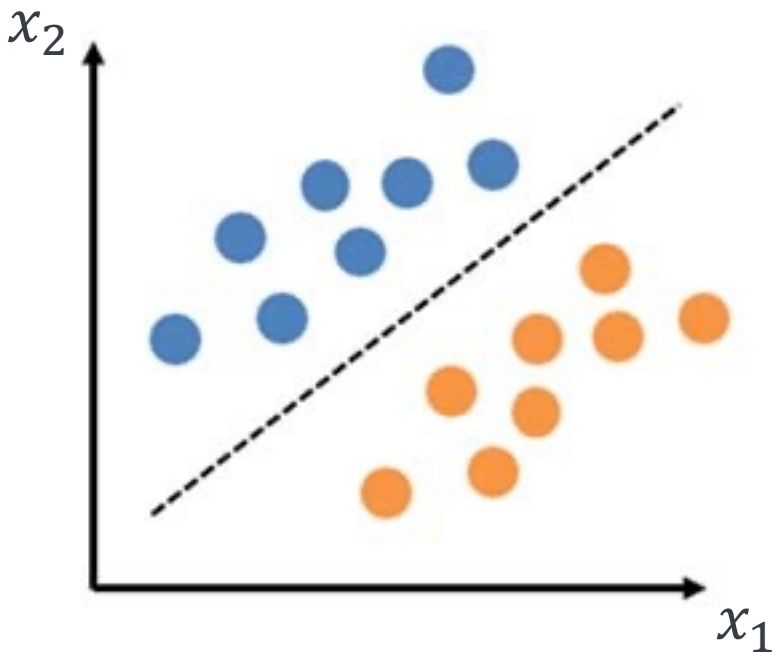
Supervised Learning

- Possibility of finding a single consistent hypothesis depends on the hypothesis space
  - Realizable: hypothesis space contains the true function
- Can use large hypothesis space (*e.g.*, space of all Turing machines)
  - Tradeoff between expressiveness and complexity of finding a simple consistent hypothesis

Perceptron

Linear Threshold Classifiers

Imagine you have data of the form (**x**, f(**x**)) where **x** in $\mathbf{R}^n$ and f(x)=0 or 1

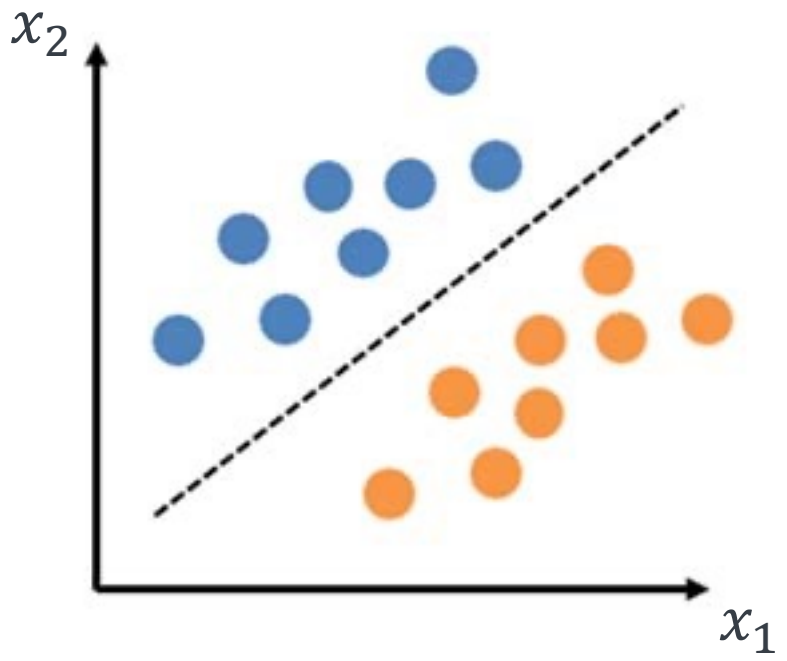


Linear Threshold Classifiers

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\boldsymbol{w} \cdot \boldsymbol{x} = w_0 + w_1 x_1 + w_2 x_2$$

(bias term)



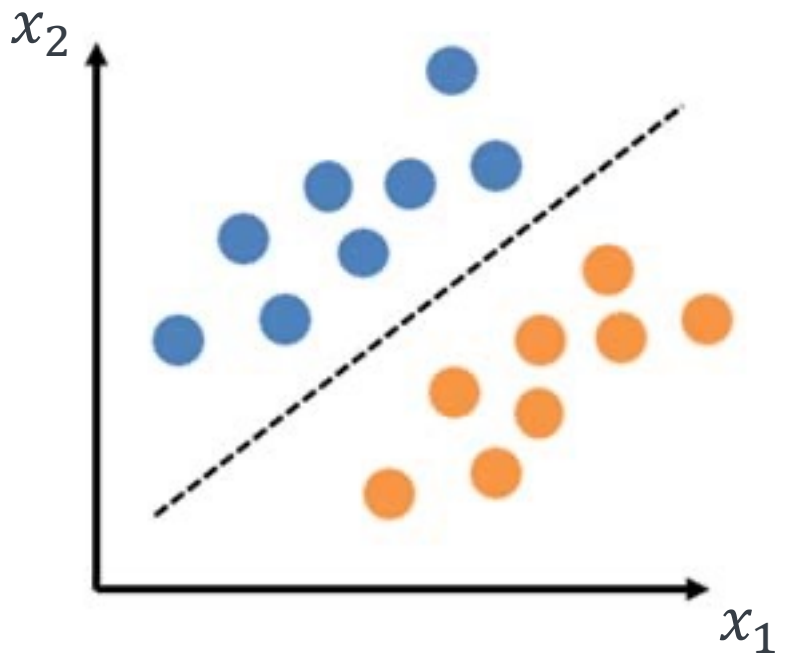
Linear Threshold Classifiers

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

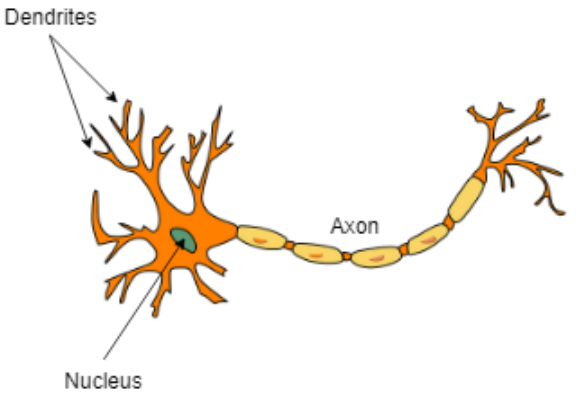
Learning problem:
Find weights, w, to minimize loss

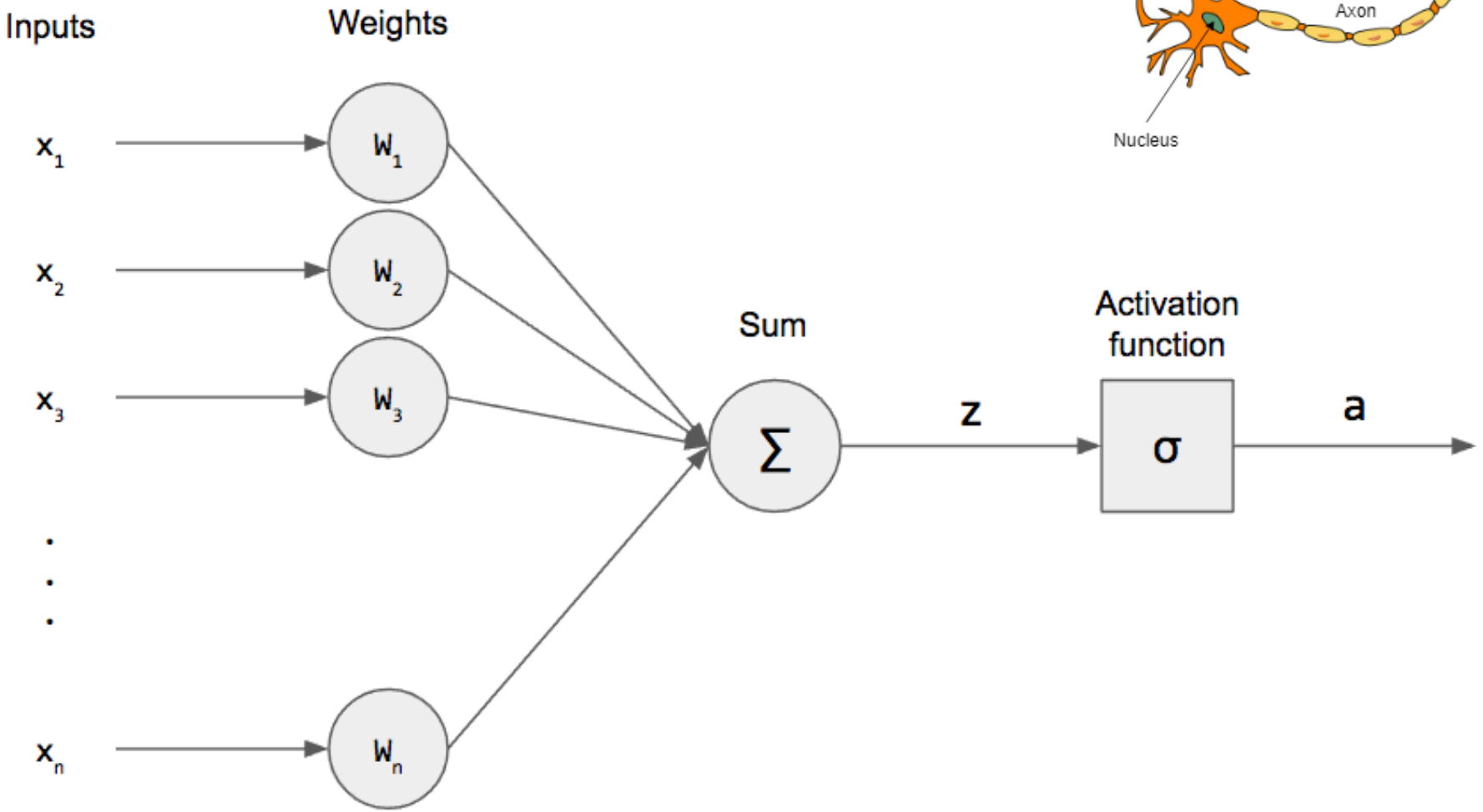
$$Loss(h_{\boldsymbol{w}}) = L_2\big(y, h_{\boldsymbol{w}}(\boldsymbol{x})\big) = \sum_{j=1}^{N} \Big(y_j - h_{\boldsymbol{w}}\big(x_j\big)\Big)^2$$

$$w_i \leftarrow w_i + \alpha(y - h_{\boldsymbol{w}}(\boldsymbol{x}))x_i$$



Binary Perceptron

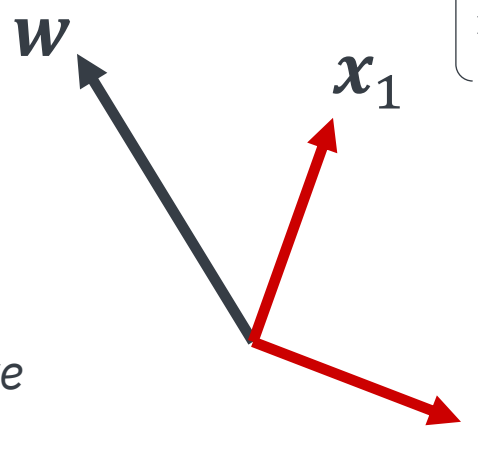




Perceptron

- Binary case: compare features to a weight vector
- Learning: figure out the weight vector from examples

```
BIAS         : 2
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```




```
BIAS         : 1
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

Dot product $\boldsymbol{w} \cdot \boldsymbol{x}$ positive means the positive class

```
BIAS         : 1
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

Binary Perceptron

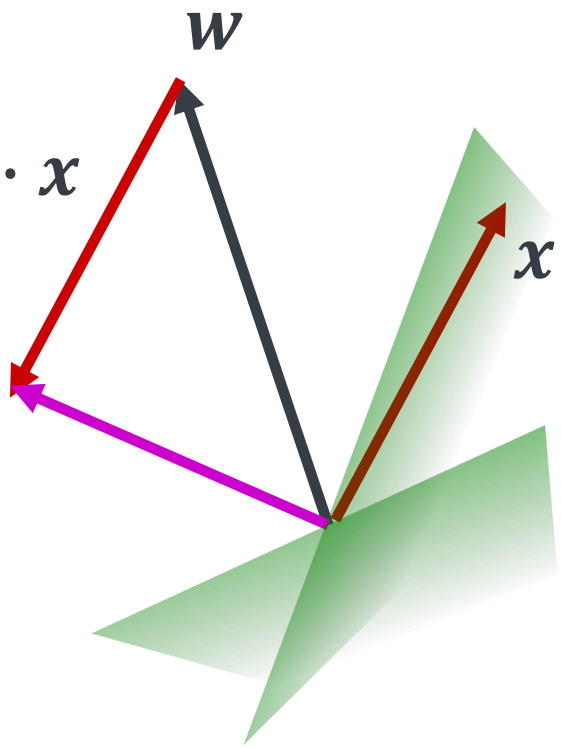
- Start with random weights
- For each training instance:
  - Classify with current weights

$$h_w(x) = \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

  - If correct (i.e., $y = h_{\boldsymbol{w}}(\boldsymbol{x})$), no change!
  - If wrong: adjust the weight vector by adding or subtracting the feature vector.

$$w_i \leftarrow w_i + (y - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot x_i$$

$(\boldsymbol{y} - \boldsymbol{h_w}(\boldsymbol{x})) \cdot \boldsymbol{x}$



Multiclass Decision Rule

- If you have multiple classes
  - A weight vector for each class:

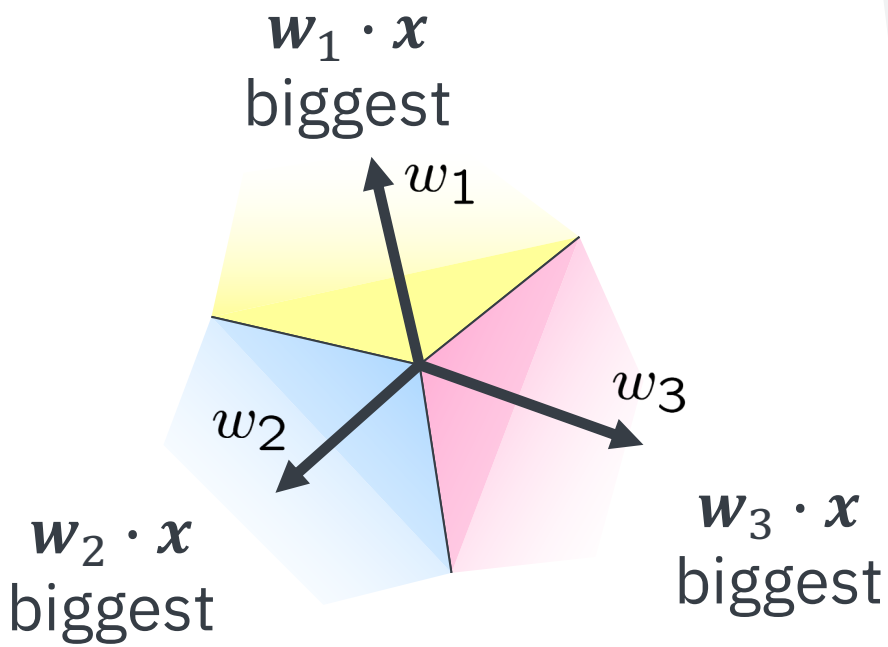
    $\boldsymbol{w}_y$

  - Score (activation) of a class y:

    $\boldsymbol{w} \cdot \boldsymbol{x}$

  - Prediction highest score wins

    $$y = \underset{y}{\operatorname{argmax}}\, \boldsymbol{w}_y \cdot \boldsymbol{x}$$



Binary = multiclass where the negative class has weight zero

Multiclass Perceptron

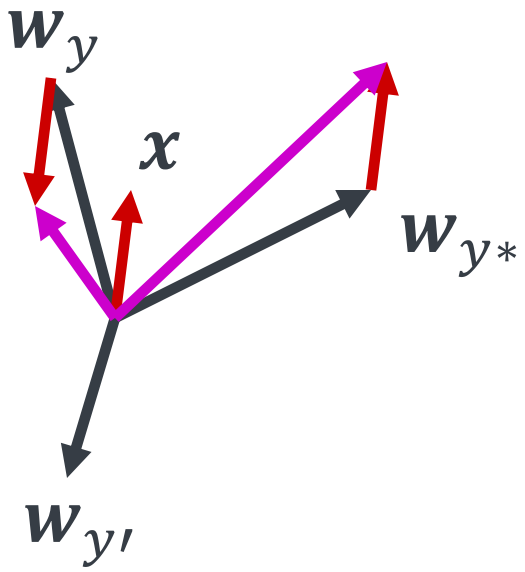
- Start with random weights
- Pick up training examples one by one
- Predict with current weights

$$y = \underset{y}{\operatorname{argmax}}\, \boldsymbol{w}_y \cdot \boldsymbol{x}$$

- If correct, no change!
- If wrong, lower score of wrong answer (y), raise score of right answer (y*)

$$\boldsymbol{w}_y = \boldsymbol{w}_y - \boldsymbol{x}$$

$$\boldsymbol{w}_{y^*} = \boldsymbol{w}_{y^*} + \boldsymbol{x}$$



Example: Multiclass Perceptron

"win the vote"

"win the election"

"win the game"

w_{SPORTS}

```
BIAS   : 1
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

$w_{POLITICS}$

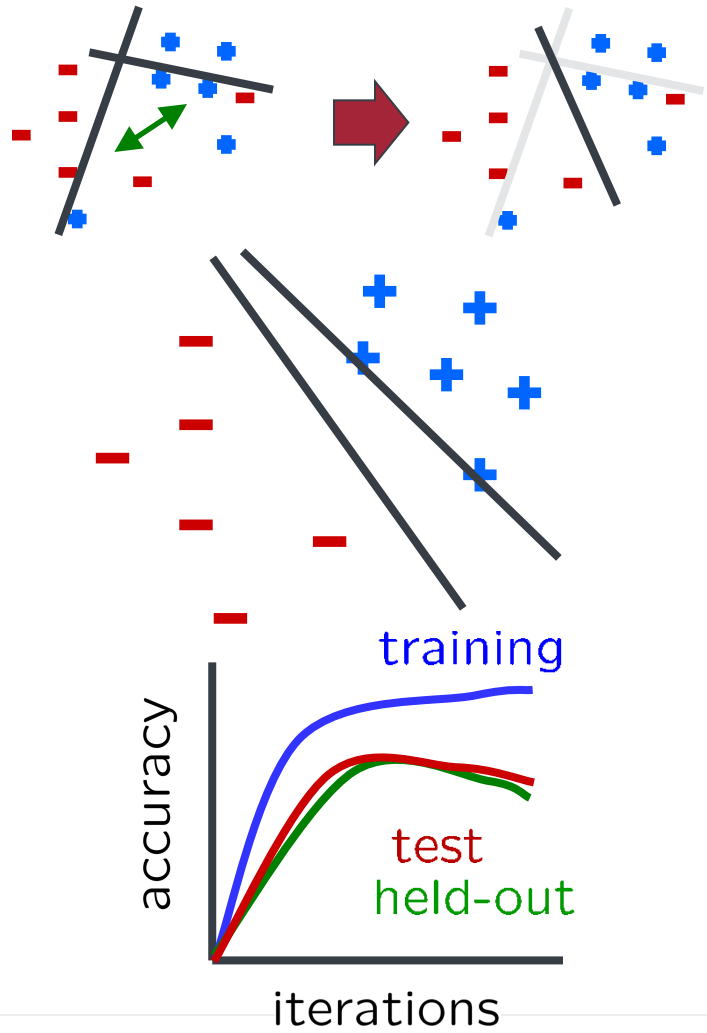
```
BIAS   : 0
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

w_{TECH}

```
BIAS   : 0
win    : 0
game   : 0
vote   : 0
the    : 0
...
```

Problems with the Perceptron

- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)
- Mediocre generalization: finds a "barely" separating solution
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



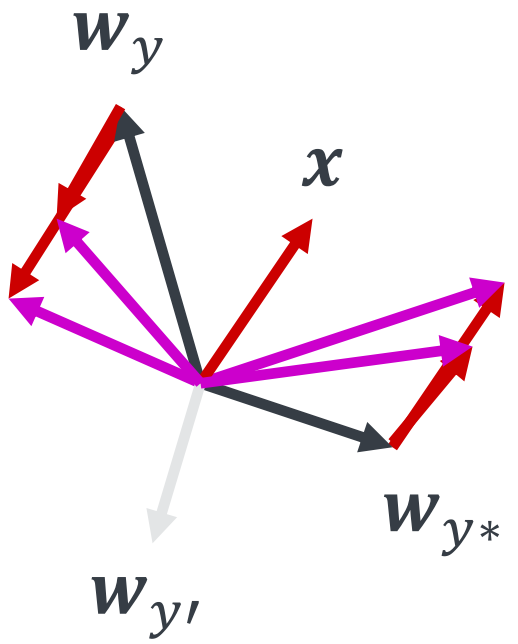
Fixing the Perceptron

- Idea: adjust the weight update to mitigate these effects
- Margin Infused Relaxed Algorithm (MIRA): choose an update size that fixes the current mistake…
- … but, minimizes the change to w

$$\min_{\boldsymbol{w}} \frac{1}{2} \sum_{y} \left\| \boldsymbol{w}_y - \boldsymbol{w}'_y \right\|^2$$

(**w**′ is the previous weights)

$$\boldsymbol{w}_{y^*} \cdot \boldsymbol{x} \geq \boldsymbol{w}_y \cdot \boldsymbol{x} + C$$



Guessed y instead of y^*

$$\boldsymbol{w}_y = \boldsymbol{w}'_y - \alpha \cdot \boldsymbol{x}$$

$$\boldsymbol{w}_{y^*} = \boldsymbol{w}'_{y^*} + \alpha \cdot \boldsymbol{x}$$

Minimum Correcting Update

$$\min_{\boldsymbol{w}} \frac{1}{2} \sum_{y} \left\| \boldsymbol{w}_y - \boldsymbol{w}'_y \right\|^2$$

(**w**′ is the previous weights)

$$\boldsymbol{w}_{y^*} \cdot \boldsymbol{x} \geq \boldsymbol{w}_y \cdot \boldsymbol{x} + C$$

$$\boldsymbol{w}_y = \boldsymbol{w}'_y - \alpha \cdot \boldsymbol{x}$$
$$\boldsymbol{w}_{y^*} = \boldsymbol{w}'_{y^*} + \alpha \cdot \boldsymbol{x}$$

$$\min_{\alpha} \|\alpha \boldsymbol{x}\|^2$$
$$\boldsymbol{w}_{y^*} \cdot \boldsymbol{x} \geq \boldsymbol{w}_y \cdot \boldsymbol{x} + C$$

$$\left(\boldsymbol{w}'_{y^*} + \alpha \cdot \boldsymbol{x}\right) \cdot \boldsymbol{x} = \left(\boldsymbol{w}'_y - \alpha \cdot \boldsymbol{x}\right) \cdot \boldsymbol{x} + C$$

$$\alpha = \frac{\left(\boldsymbol{w}'_y - w'_{y*}\right) \cdot \boldsymbol{x} + C}{2\boldsymbol{x} \cdot \boldsymbol{x}}$$

min of α will be where equality holds

STEVENS
INSTITUTE OF TECHNOLOGY
1870

15 min. break

Decision Tree

Decision Tree

- Decision trees classify instances by sorting them down the tree from root to leaf.
  - Nodes correspond with a test of some attribute.
  - Each branch corresponds to some value an attribute can take.
- Classification algorithm
  - Start at root, test attribute specified by root
  - Move down the branch corresponding to value of the attribute
  - Continue until you reach leaf (classification)

Decision Tree

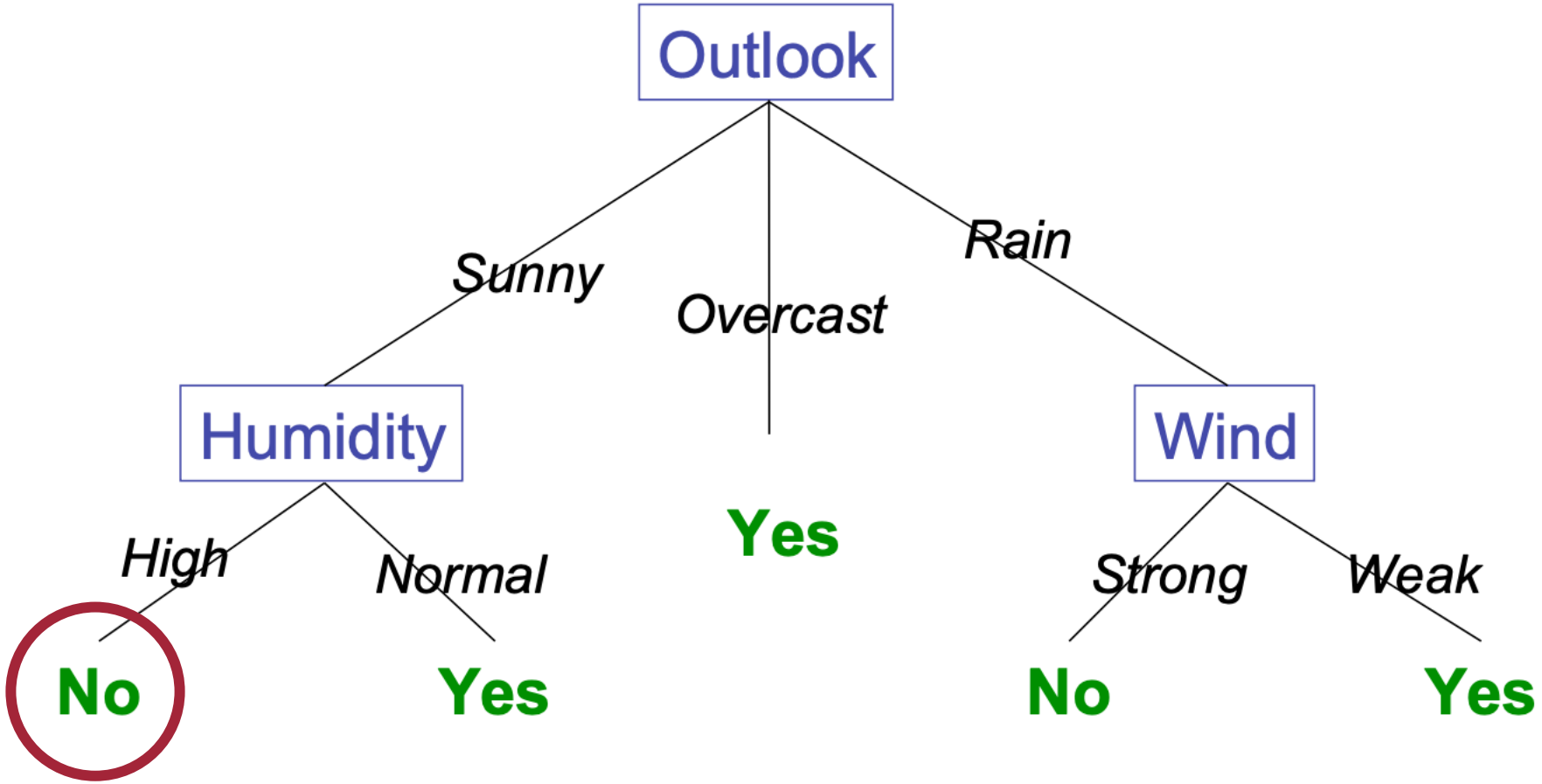
Attribute

Sky	Air Temp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

x

f(x)

Decision Tree



<Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong>

Classification (EnjoySport): No

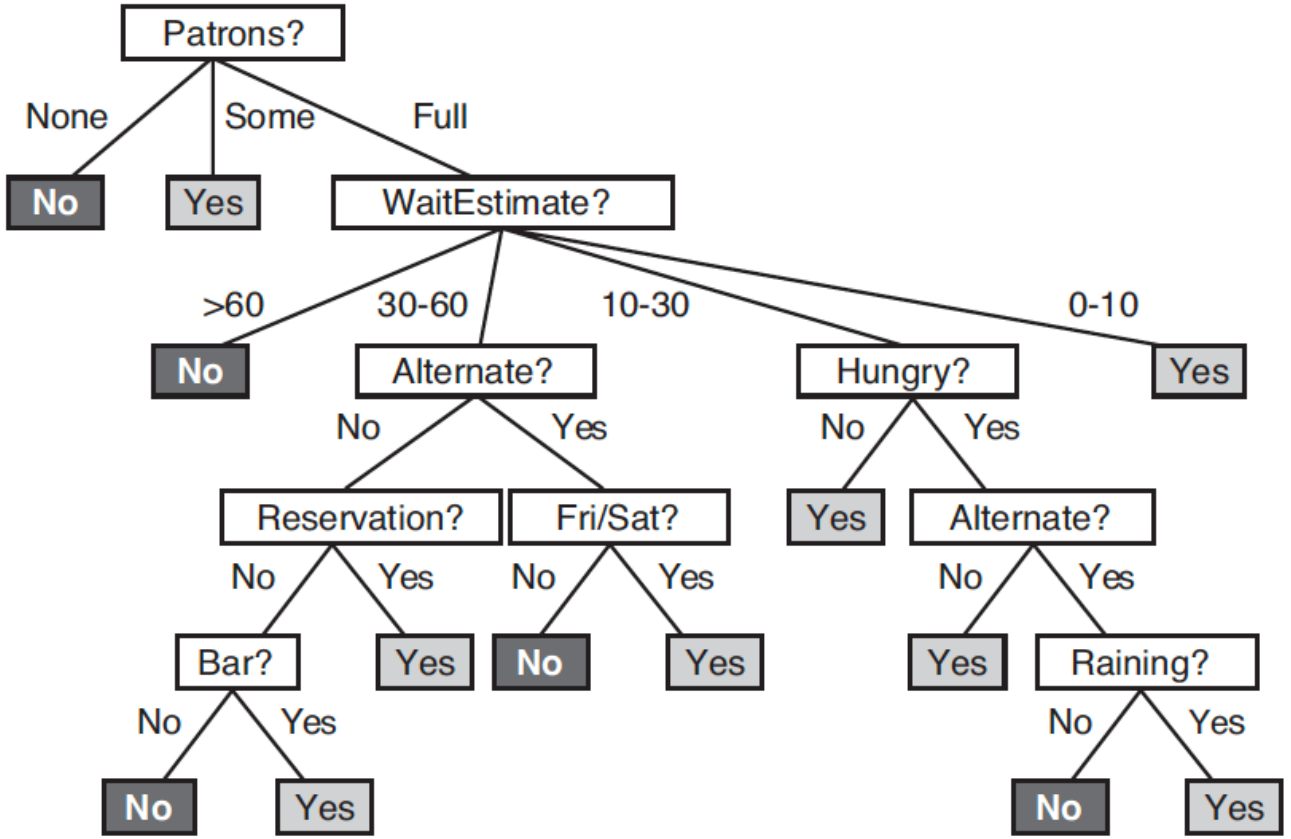
Decision-Tree Representation

- Decision trees are fully expressive within the class of propositional languages.
- Any Boolean function can be written as a decision tree.

Example: Restaurants

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Example: Restaurants



Inducing a Decision Tree

- Aim: find a small tree consistent with the training examples
  - pick the attribute that goes as far as possible toward providing an **exact classification of the examples**
  - (recursively) choose the **most significant** attribute as root of (sub)tree

Inducing a Decision Tree

```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

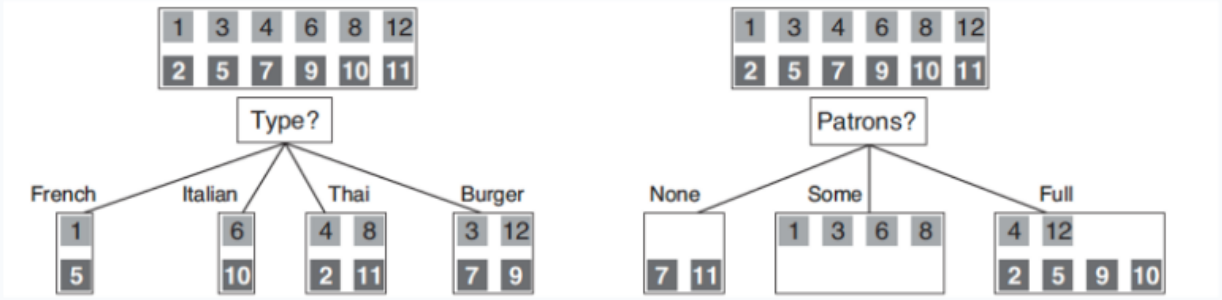
Return the most common output value.

What attribute do we choose?

Build the tree recursively.

Inducing a Decision Tree

Which attribute is more significant?

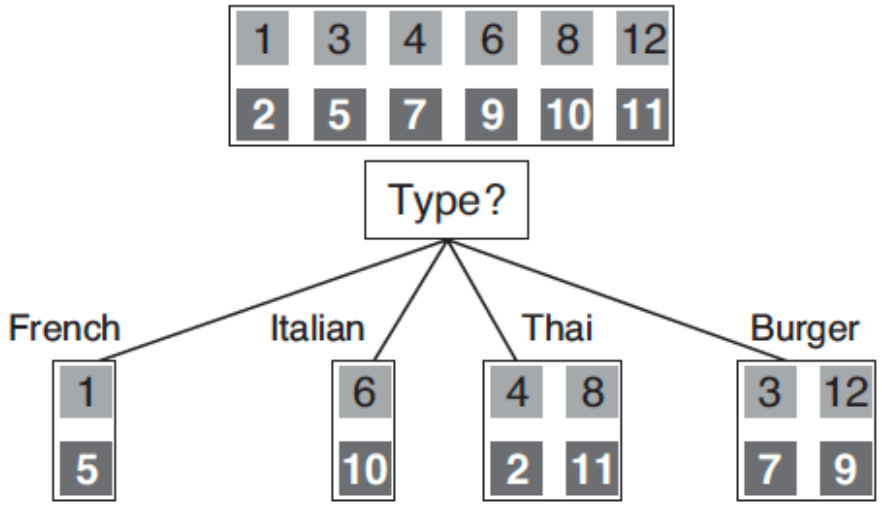


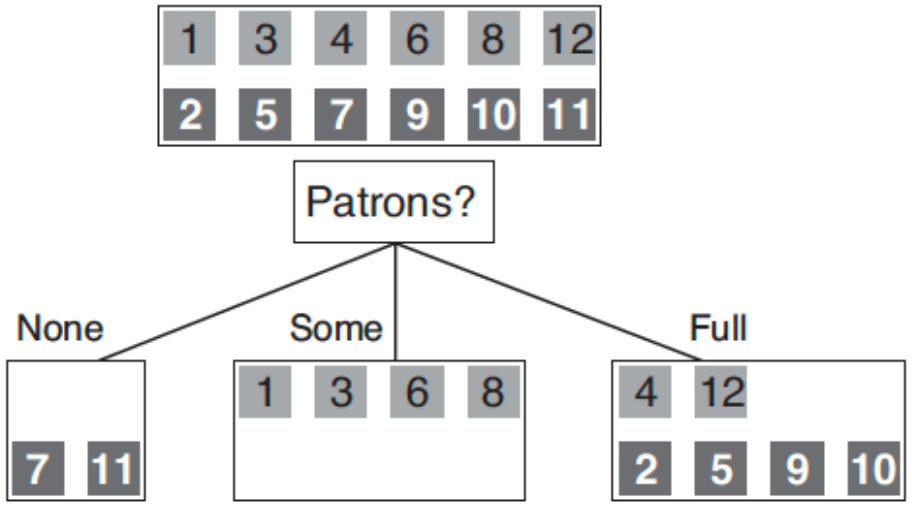
Type

Patrons

CHOOSE-ATTRIBUTE

- A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.





Using Information Theory

- Information content (entropy, a measure of the uncertainty of a random variable):

$$I\big(P(v_1), \ldots, P(v_2)\big) = \sum_i -P(v_i)\log_2 P(v_i)$$

- For training set containing p positive examples and n negative examples,

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Information Gain

- Chosen attribute A divides the training set E into subsets E_1, …, E_v according to their values for A, where A has v distinct values.

$$remainder(A) = the\ expected\ entropy\ remaining\ after\ testing\ attribute\ A$$

$$= \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

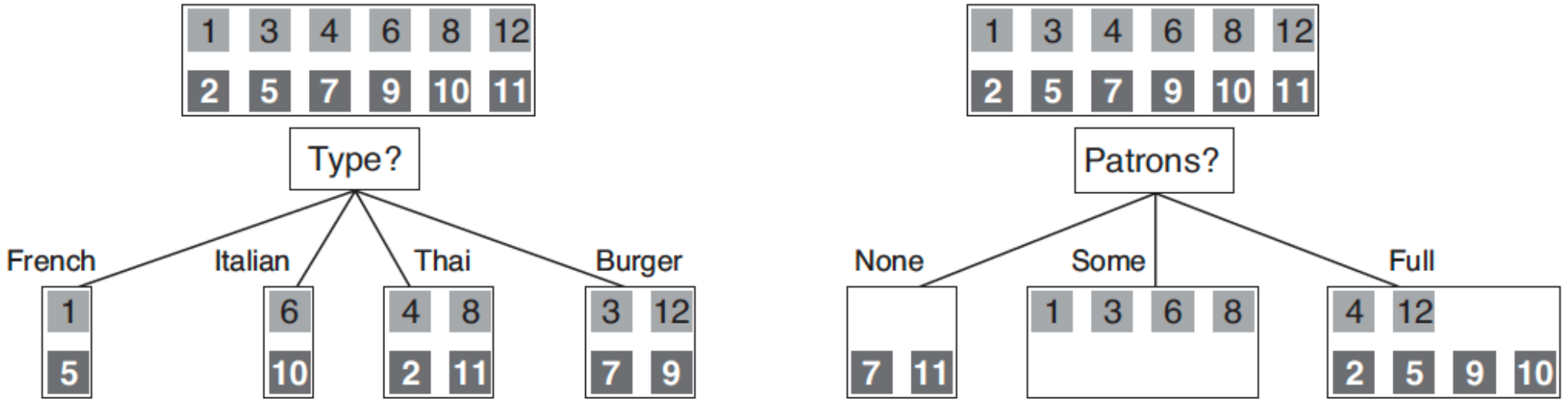
- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = Current\ entropy\ - remainder(A)$$

$$= I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

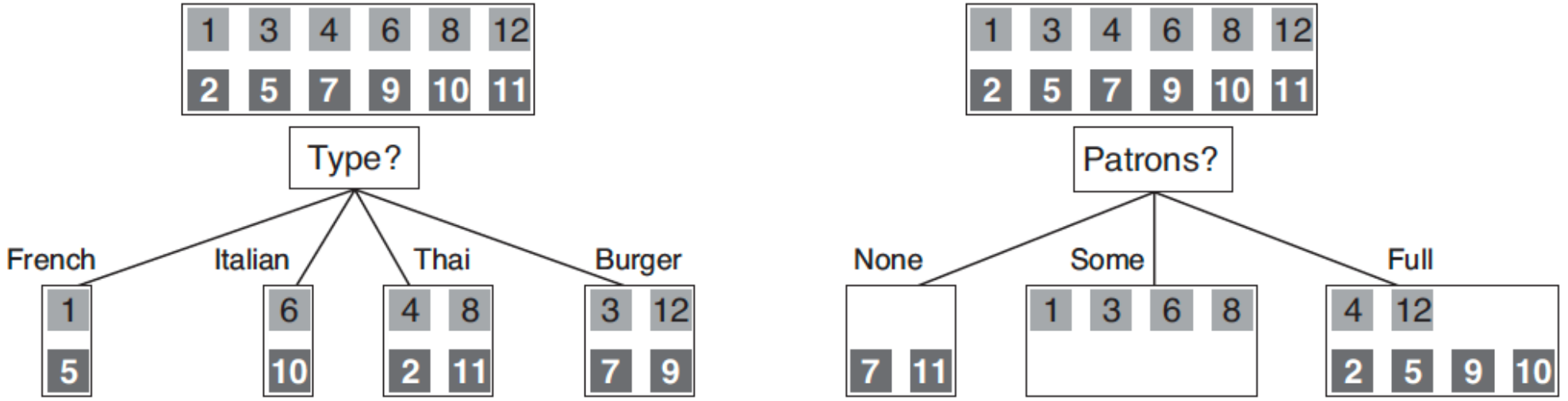
CHOOSE-ATTRIBUTE

- A good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”.



Choose the attribute with the highest information gain!

Which attribute is more significant?



$$IG(Type) = 1 - [\frac{2}{12} I\left(\frac{1}{2},\frac{1}{2}\right) + \frac{2}{12} I(\frac{1}{2},\frac{1}{2}) + \frac{4}{12} I(\frac{2}{4},\frac{2}{4}) + \frac{4}{12} I(\frac{2}{4},\frac{2}{4})] = 0$$

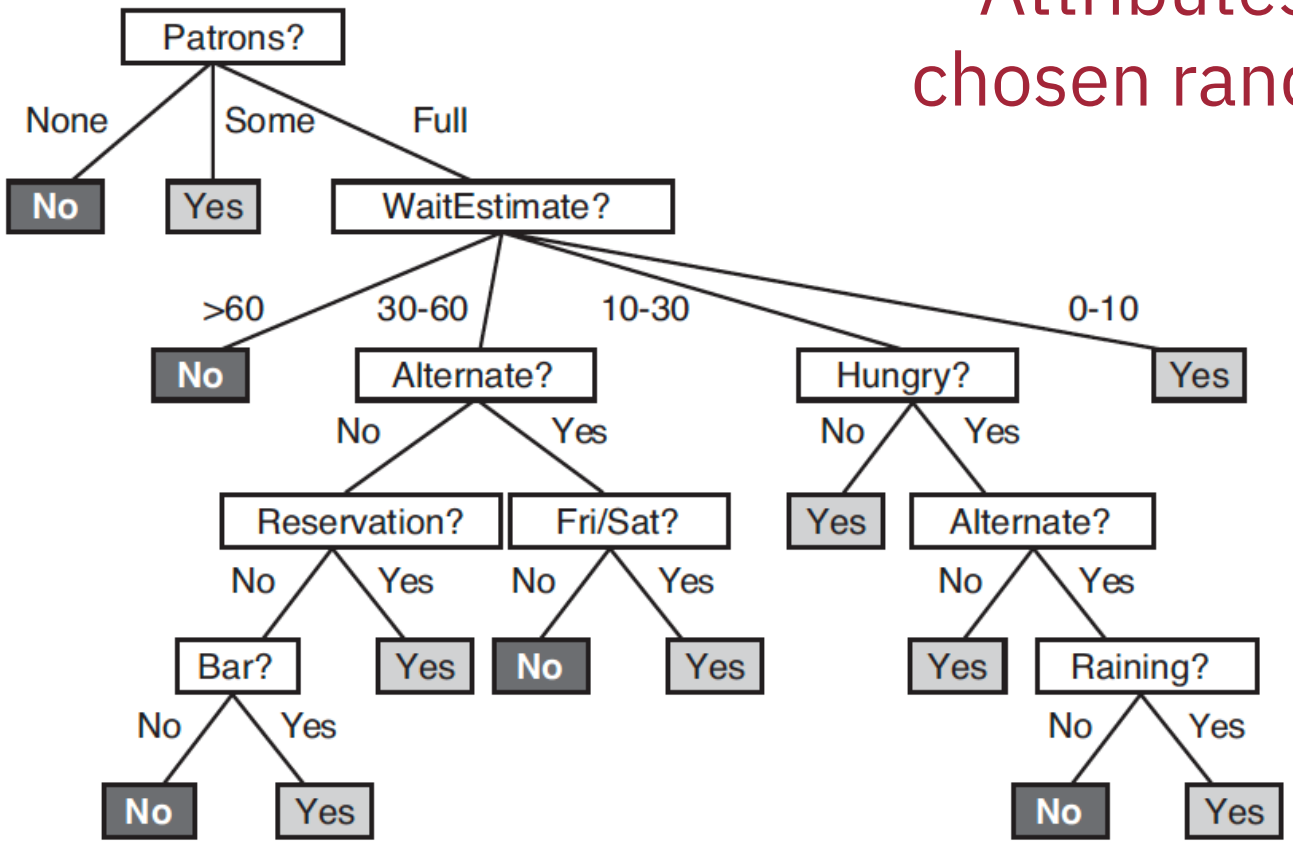
$$\leq$$

$$IG(Patrons) = 1 - [\frac{2}{12} I\left(\frac{0}{2},\frac{2}{2}\right) + \frac{4}{12} I(\frac{4}{4},\frac{0}{4}) + \frac{6}{12} I(\frac{2}{6},\frac{4}{6})] \approx 0.541$$

Example: Restaurants

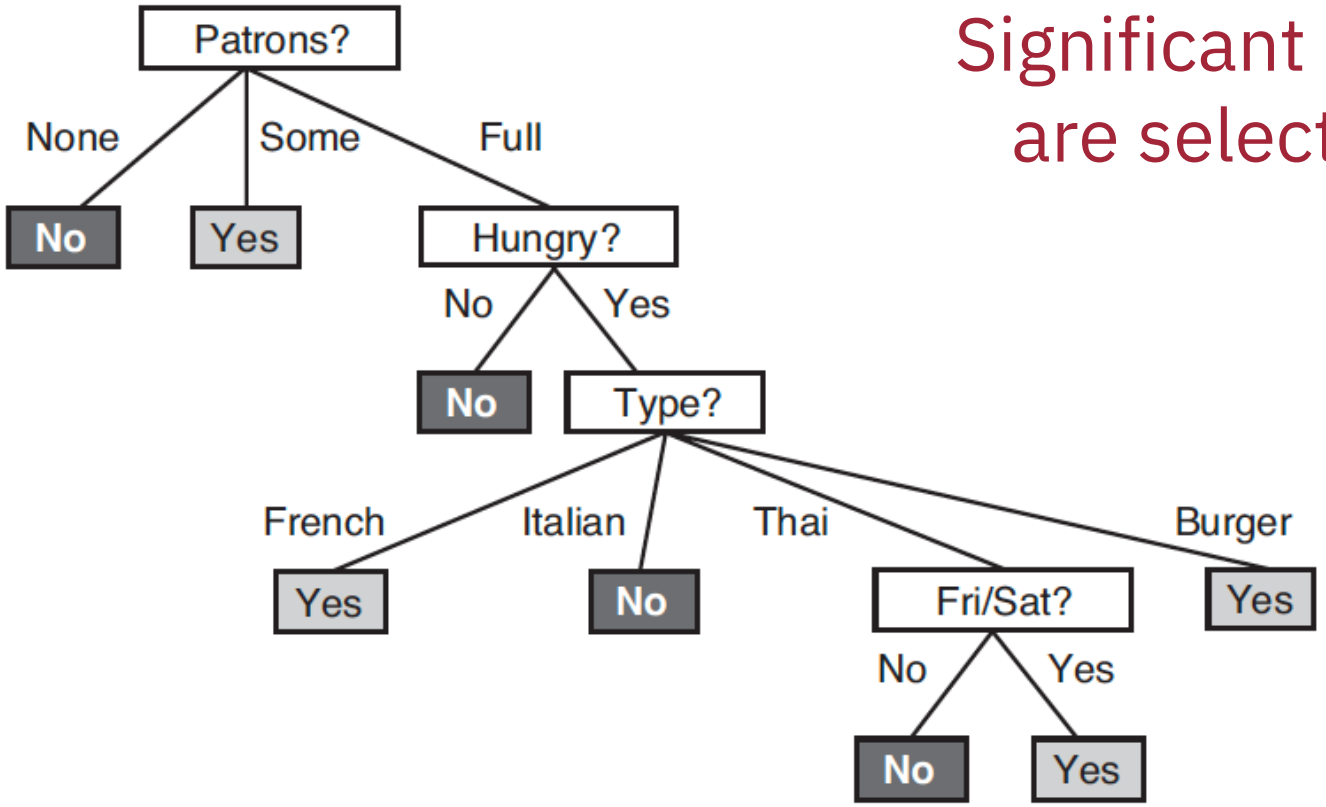
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Example: Restaurants



Attributes are chosen randomly.

Example: Restaurants



Significant attributes are selected first.

We can find a good approximate solution: a small (but not smallest) consistent tree.

Evaluating a Learning Algorithm

Performance of a Learning Algorithm

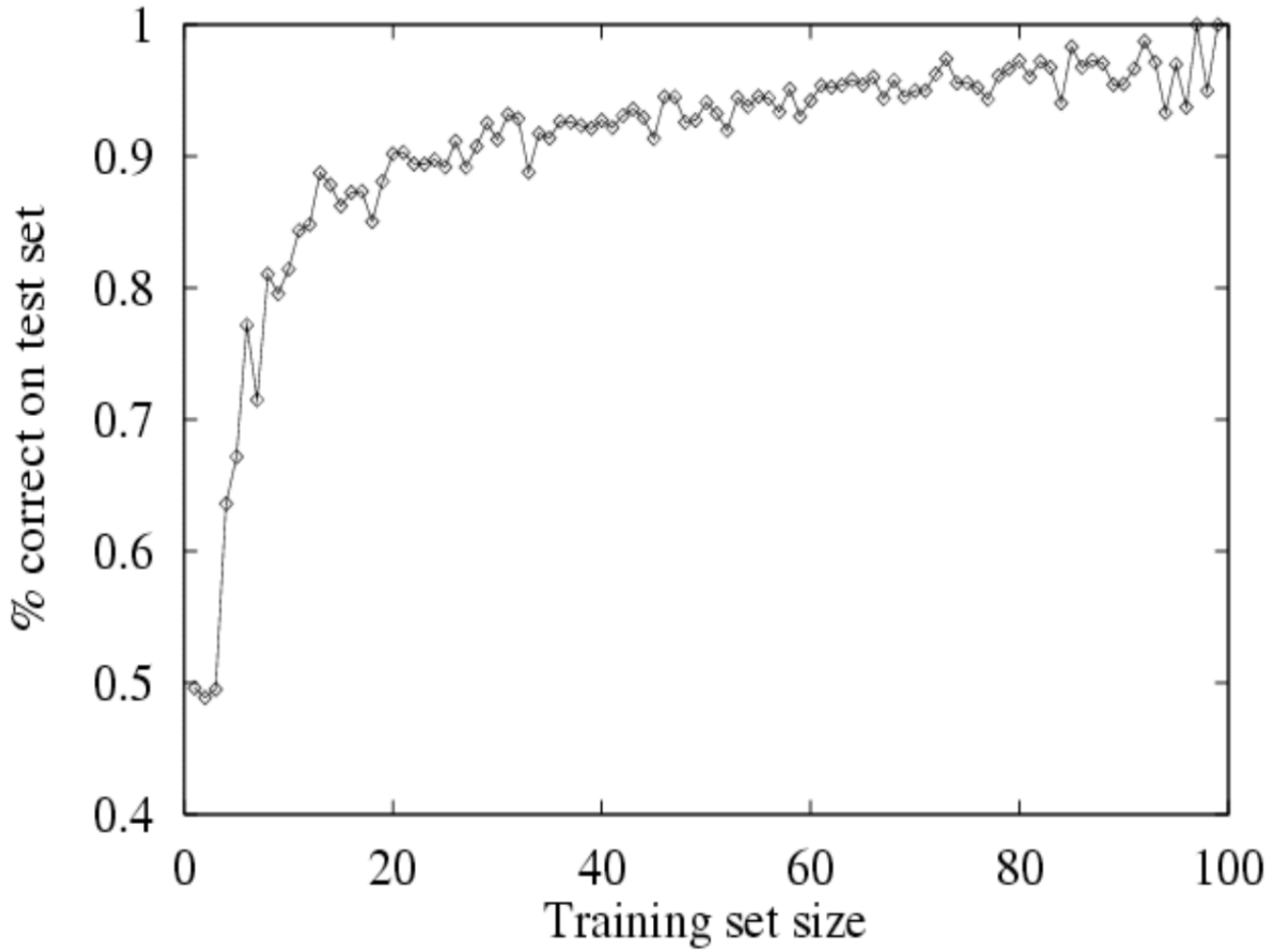
- A learning algorithm is good if it produces a hypothesis that does a good job of predicting classifications of unseen examples
- There are theoretical guarantees (learning theory)
- Can also test this

Performance of a Learning Algorithm

- Test set
  - Collect a large set of examples.
  - Divide them into 2 disjoint sets: training set and test set.
  - Apply learning algorithm to the training set to get h.
  - Measure percentage of examples in the test set that are correctly classified by h.

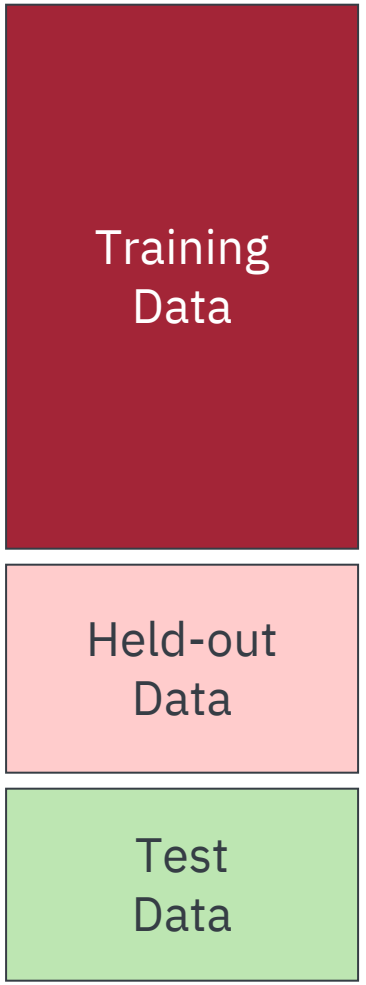
Learning Curves

As the training set grows, accuracy increases.



No Peeking at the Test Set!

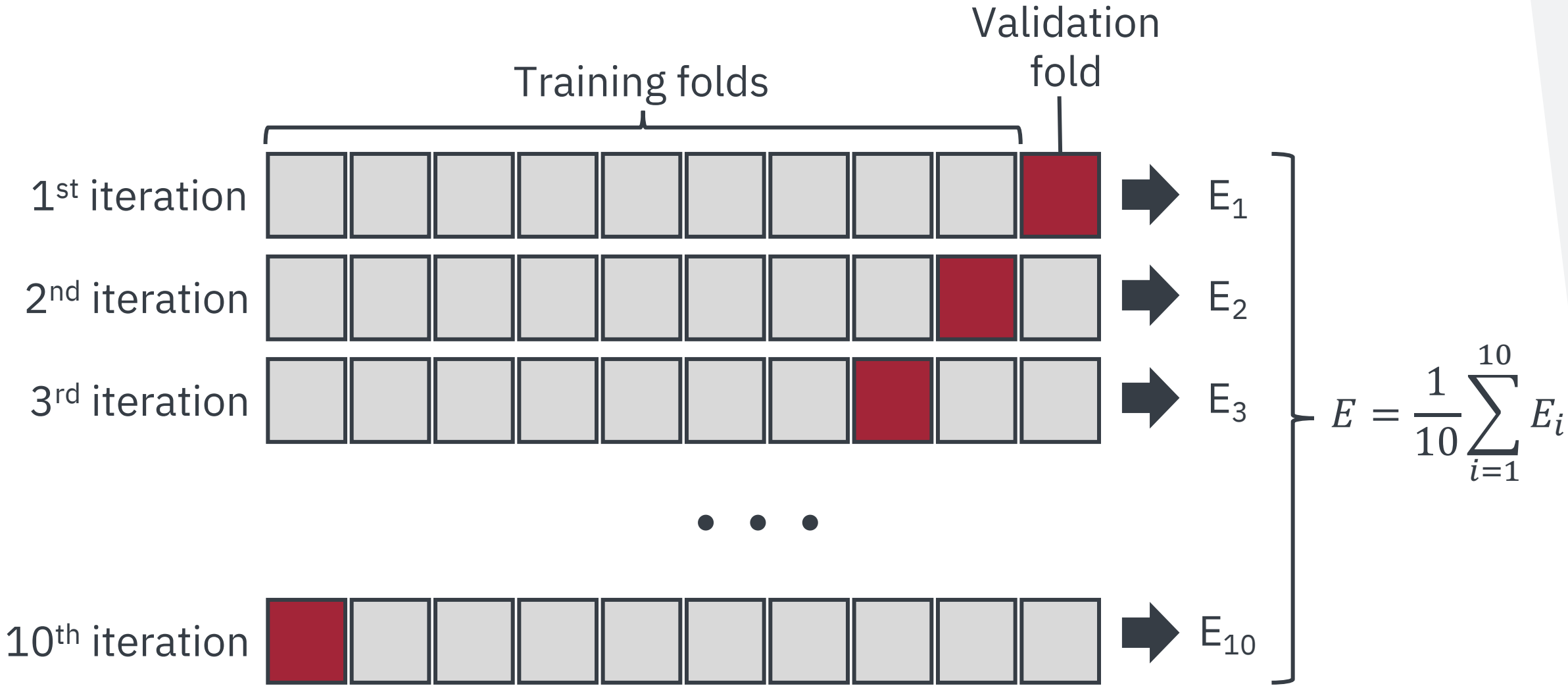
- A learning algorithm should **not** be allowed to see the test set data before the hypothesis is tested on it.
- Every time you want to compare performance of a hypothesis on a test set **you should use a new test set!**
- If you want to measure the performance of your hypothesis, divide the available data (without the test set) into a training set and **a validation set (Held-out data)**.



Cross Validation

- Split the training set into two parts, one for training (training data) and one for choosing the hypothesis with highest accuracy (held-out data).
  - K-fold cross validation means you run k experiments, each time putting aside 1/k of the data to test on
  - Leave-one-out cross validation.

10-Fold Validation



Ensemble Learning

Ensemble Learning

- So far our learning methods have had the following general approach.
  - Choose **a single hypothesis** from the hypothesis space.
  - Use this hypothesis to make predictions.
- Maybe we can do better by using **a lot of hypothesis** from the hypothesis space and combine their predictions.

Ensemble Learning

- Analogies
  - Elections
  - Committees
- Intuitions
  - Individuals may make mistakes.
    - The majority may be less likely to make a mistake
  - Individuals have partial information
    - Committees pool expertise

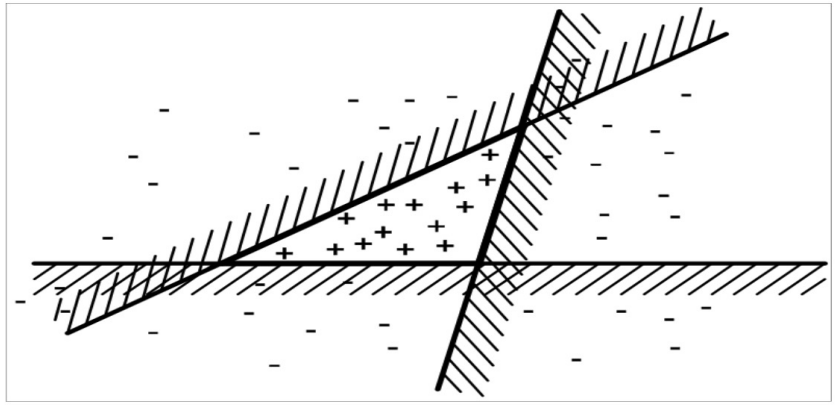
Ensemble Expressiveness

- Using ensembles can also enlarge the hypothesis space.
  - Ensemble as hypothesis
  - Set of all ensembles as hypothesis space

Linear threshold hypothesis

Simple, efficient learning algorithms but not particularly expressive

Ensemble allows us to learn more expressive hypothesis



Ensemble Learning Techniques

- Bagging
- Boosting

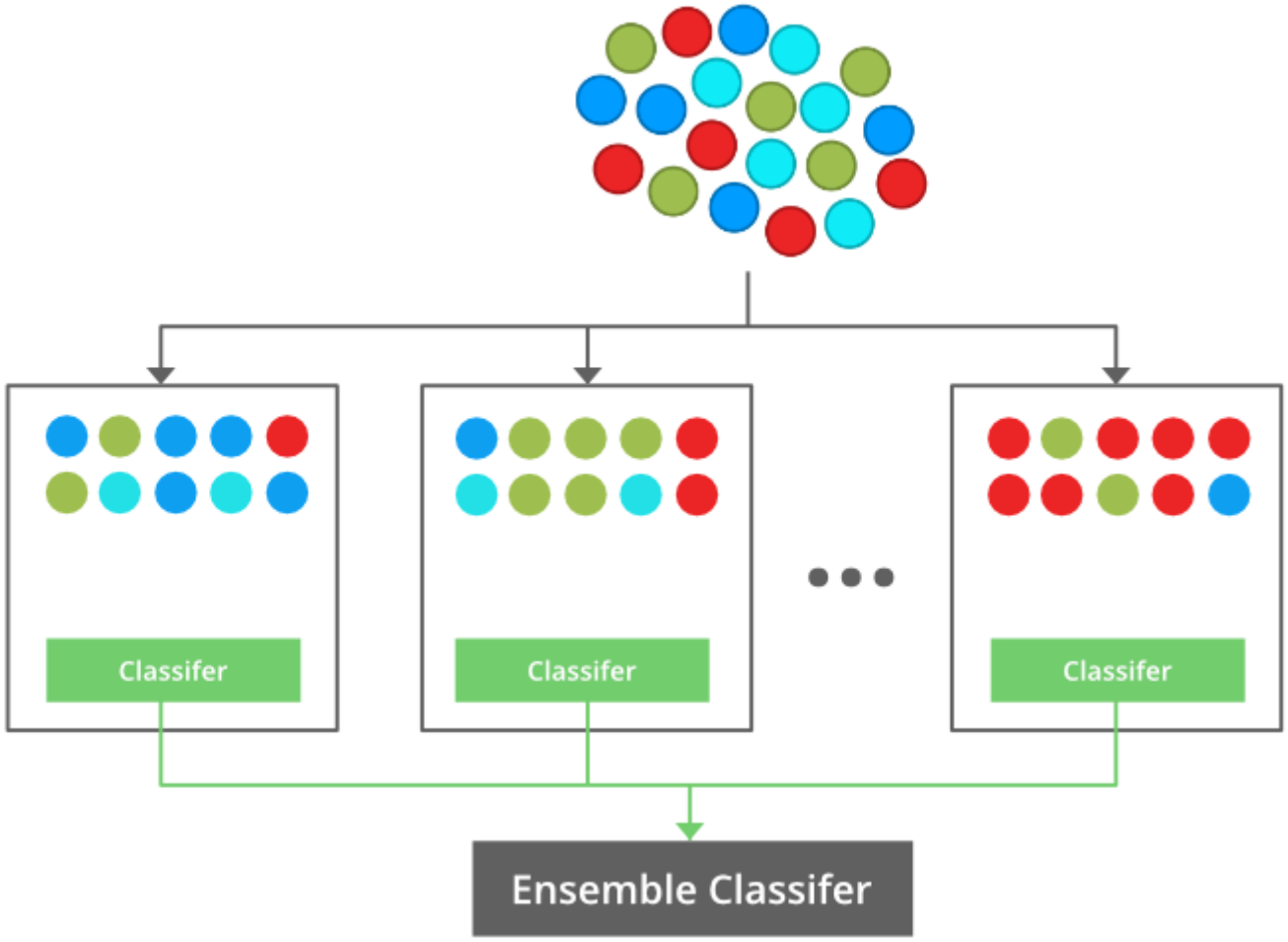
Bagging

- Assumptions
  - Each hypothesis makes an error with probability p.
  - Hypotheses are independent.
- Majority voting of n hypotheses
  - Probability k hypotheses make an error?
  - Probability majority make an error?

Bagging

1. **Multiple subsets** are created from the original data set with equal tuples, selecting observations with replacement.
2. **A base model is created** on each of these subsets.
3. Each model is **learned in parallel with each training** set and independent of each other.
4. The final predictions are determined by **combining the predictions** from all the models.
   - Majority vote
   - Averaging
   - ...

Bagging



Weighted Majority

- In practice
  - Hypotheses are rarely independent
  - Some hypotheses have less errors than others
- Weighted majority
  - Decrease weights of poor hypotheses
  - Increase weights of good hypotheses

Boosting

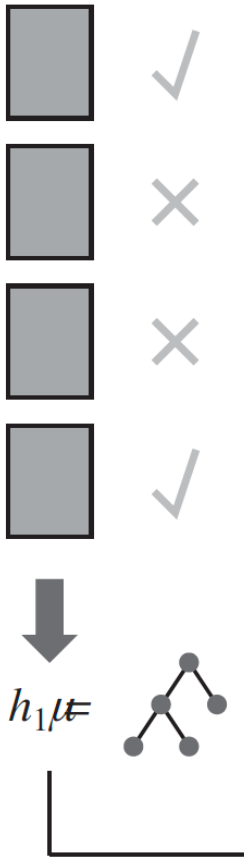
- Boosting is the most commonly used form of ensemble learning.
  - Computes a weighted majority
  - Operates on a weighted training set

Boosting

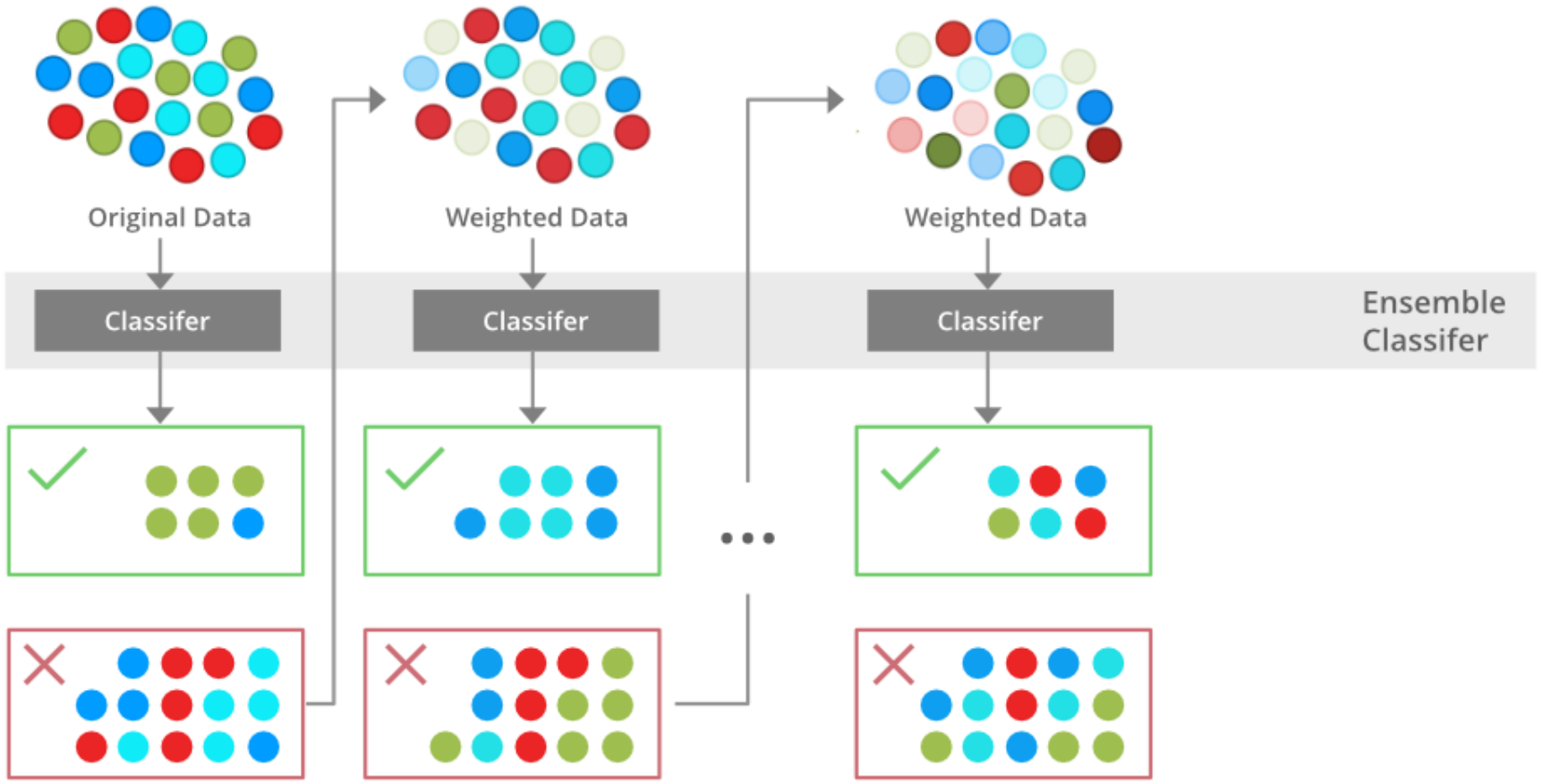
1. Initialize the dataset and **assign equal weight to each of the training examples**.
2. Provide this as input to the model and **identify the wrongly classified training examples**.
3. **Increase the weight** of the wrongly classified examples and **decrease the weights** of correctly classified examples.
4. **Normalize** the weights of all training examples.
5. **if** got required results
   Finish
   else
   Go to step 2

Visualizing the Process of Boosting

- **Shaded rectangle**: training example
  - **Height**: weight
  - **Checks and crosses**: whether the example was classified correctly by the current hypothesis.
- The size of the **decision tree**
  - weight of that hypothesis

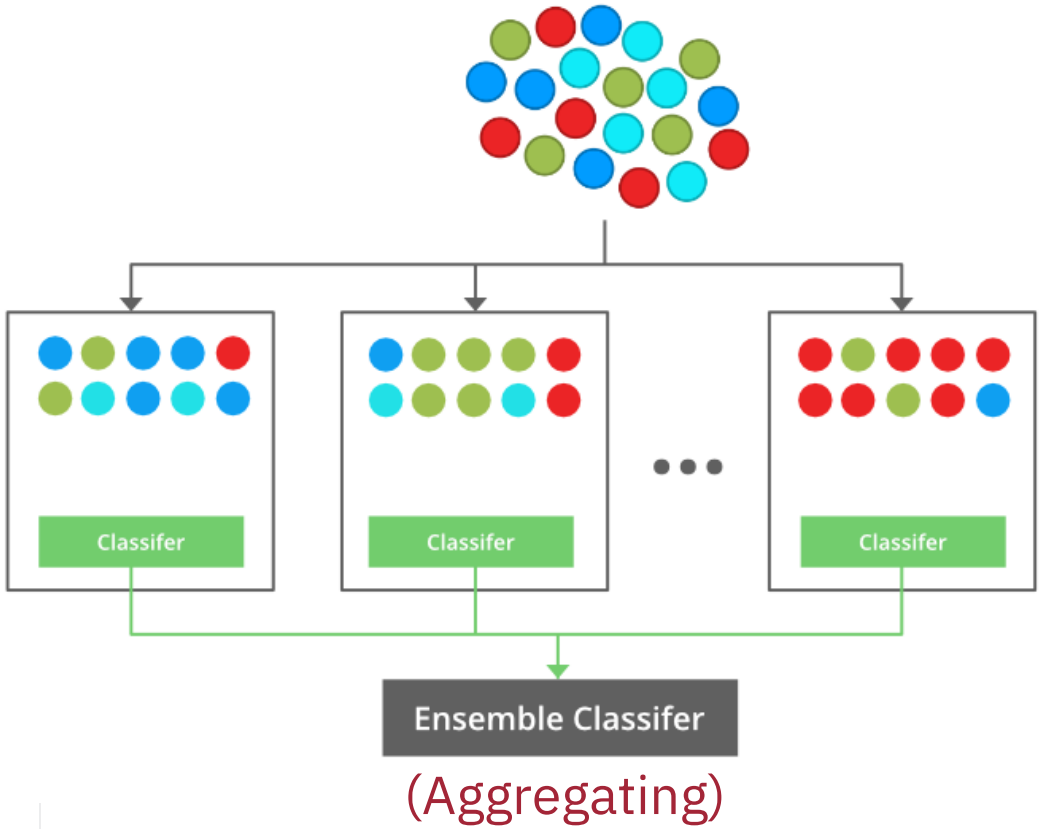


Boosting


Original Data
Weighted Data
Weighted Data
Classifer
Classifer
Classifer
Ensemble
Classifer

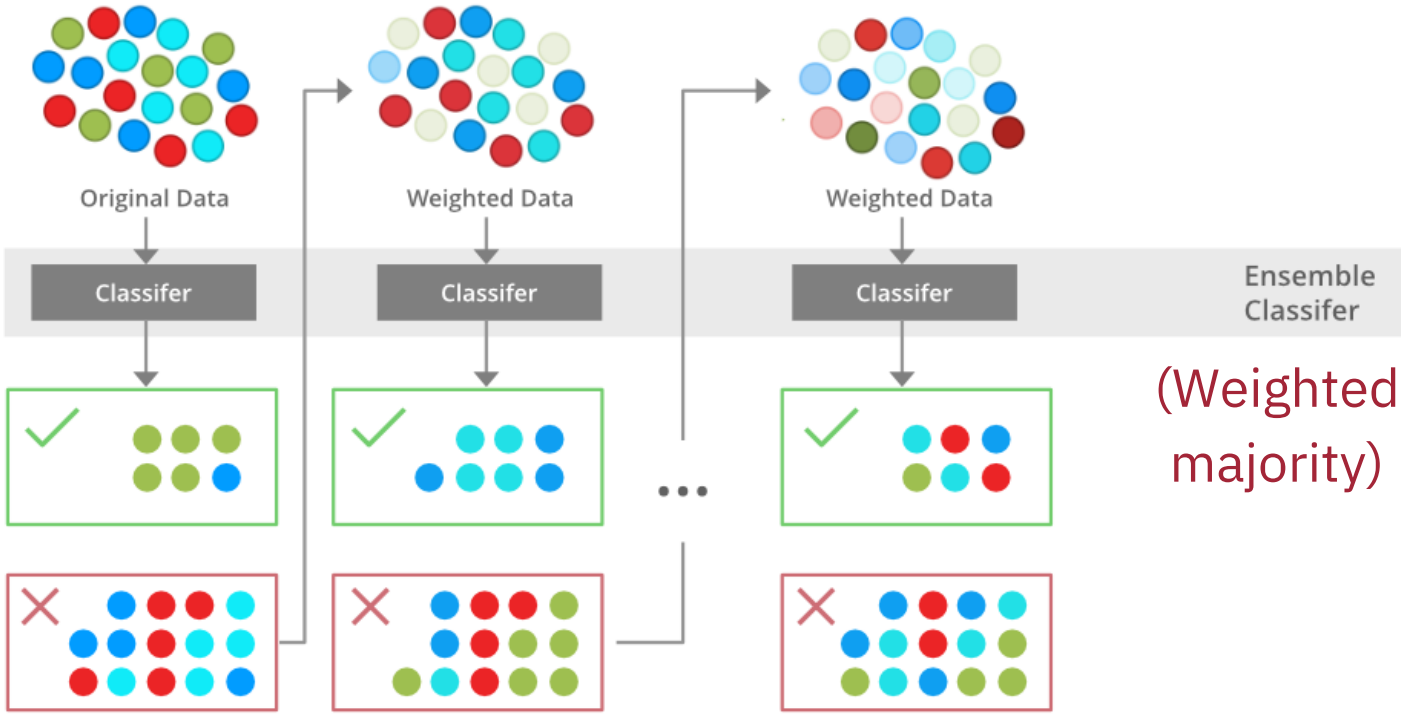
Bagging vs. Boosting

Bagging



(Aggregating)

Boosting



(Weighted majority)

AdaBoost

- There are many variants of the basic boosting idea
  - Different ways of adjusting the weights and combining the hypotheses
- AdaBoost (Adaptive Boosting)
  - a very popular boosting technique that combines multiple weak classifiers to build one strong classifier
  - It was initially created to increase the efficiency of binary classifiers.

AdaBoost

function ADABOOST($examples, L, K$) **returns** a weighted-majority hypothesis
  inputs: $examples$, set of N labeled examples $(x_1, y_1), \ldots, (x_N, y_N)$
    L, a learning algorithm
    K, the number of hypotheses in the ensemble
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of K hypotheses
    z, a vector of K hypothesis weights

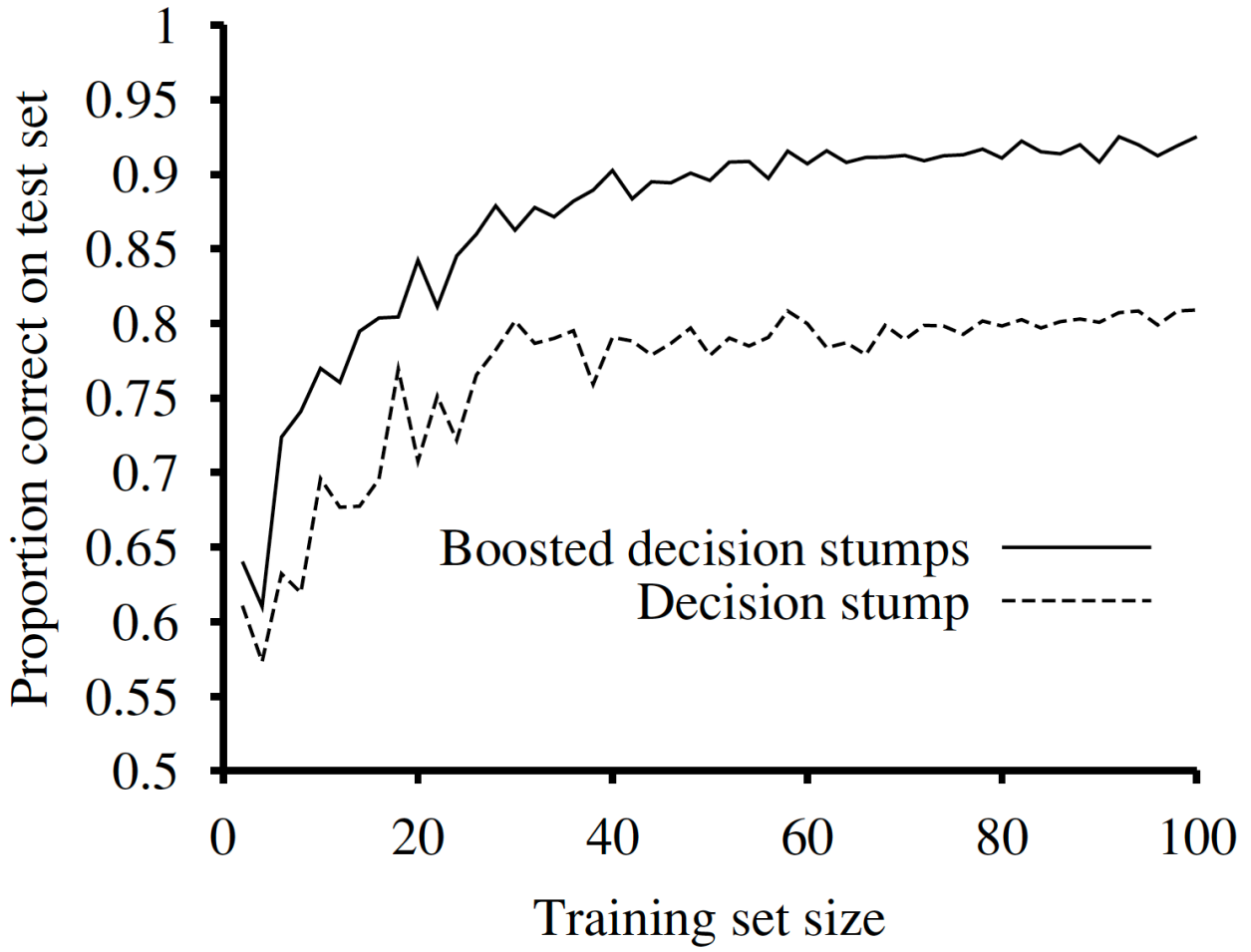
  for $k = 1$ **to** K **do**
    $\mathbf{h}[k] \leftarrow L(examples, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ **to** N **do**
      if $\mathbf{h}[k](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE(**w**)
    $\mathbf{z}[k] \leftarrow \log(1 - error)/error$
  return WEIGHTED-MAJORITY(**h**, **z**)

AdaBoost

- AdaBoost with **K weak learning algorithms** returns a hypothesis that classifies the training data **perfectly** for large enough K.
  - **A week learning algorithm** always returns a hypothesis with accuracy on the training set that is slightly better than random guessing (i.e., 50%+ for Boolean classification).
- AdaBoost boosts the accuracy of the original learning algorithm on the training data.

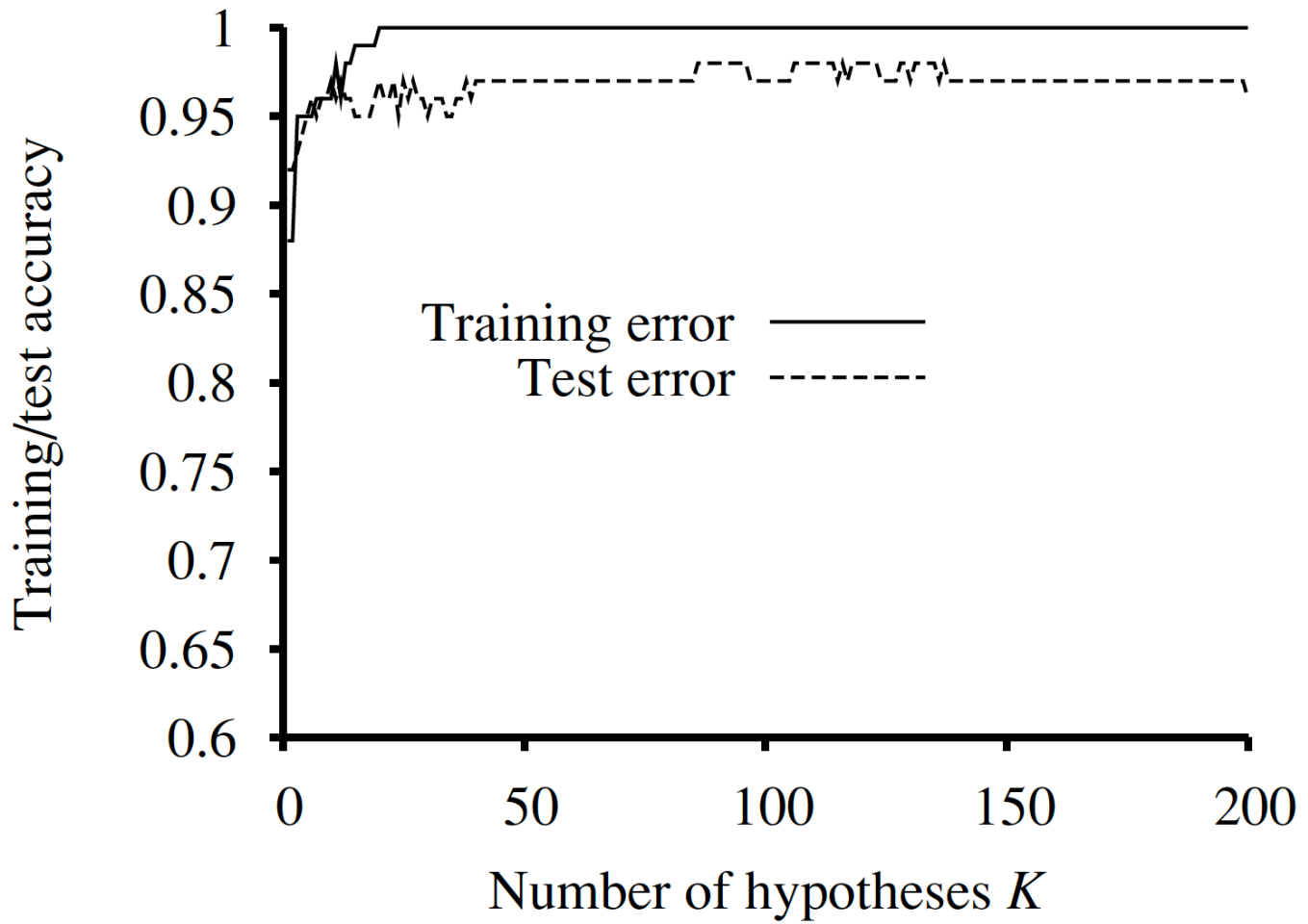
AdaBoost with Restaurant Example

- Decision stump
  - a decision tree with just one test, at the root
- Boosted decision stumps
  - AdaBoost with 5 stumps



AdaBoost with Restaurant Example

- What happens if K increases?



Advantages of Boosting Paradigm

- No need to learn a perfect hypothesis
- Can boost any weak learning algorithm
- Easy to program
- Good generalization

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Questions?

Acknowledgement

Fahiem Bacchus, University of Toronto
Dan Klein, UC Berkeley
Kate Larson, University of Waterloo