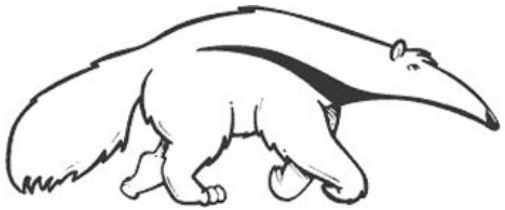
Introduction to Machine Learning: Improve Performance by Observation

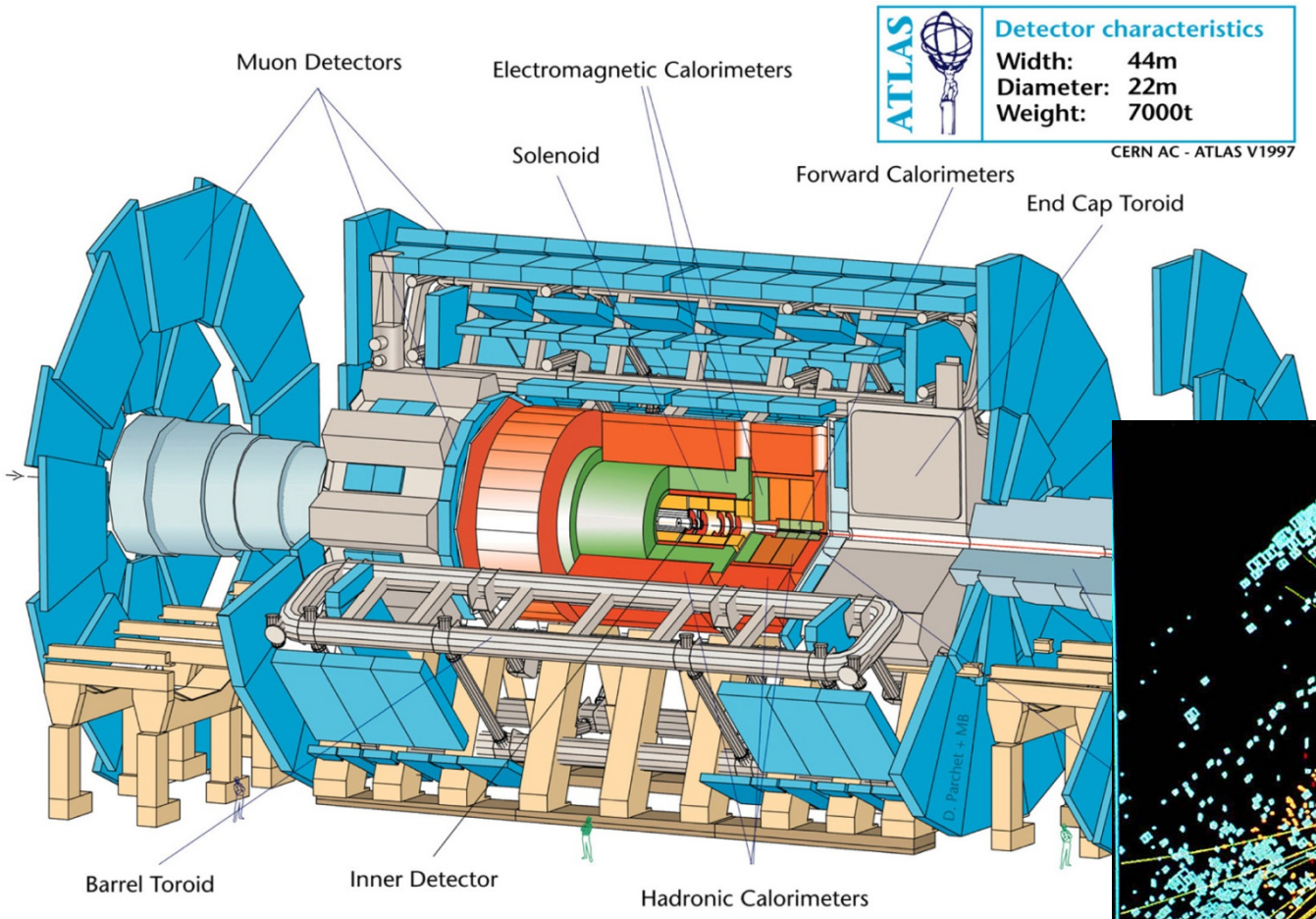
CS171, Fall Quarter, 2018
Introduction to Artificial Intelligence
Prof. Richard Lathrop

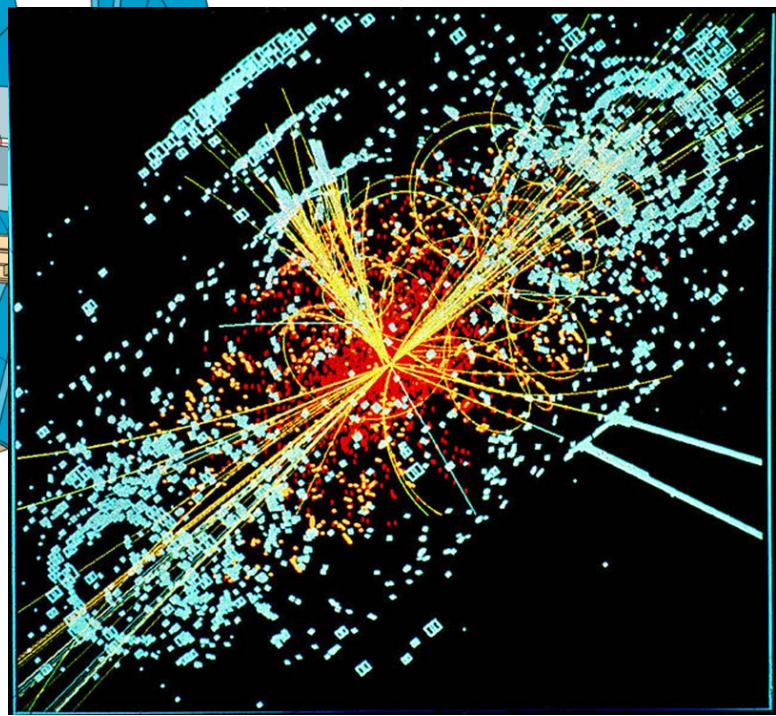


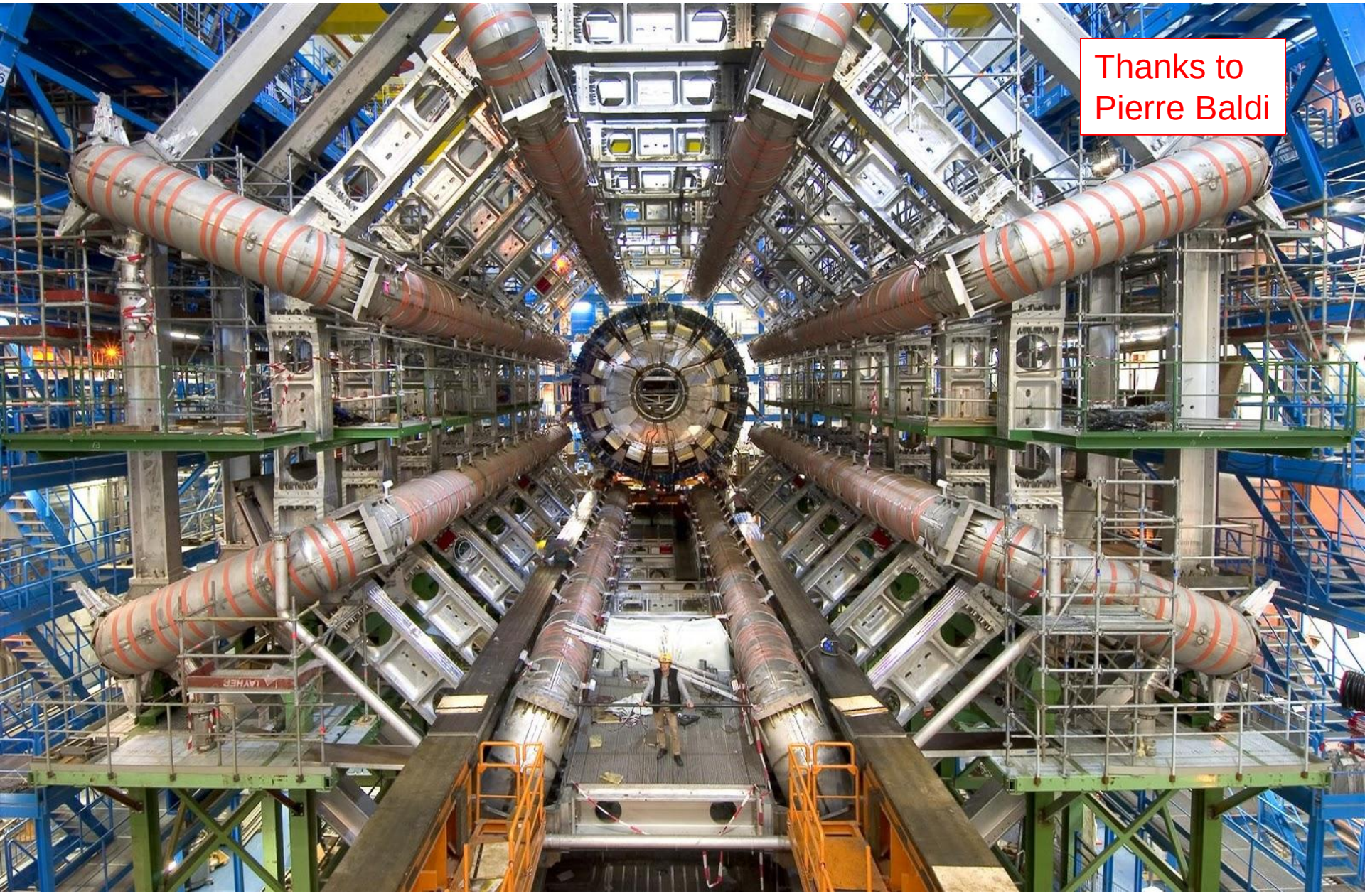
Read Beforehand: R&N Ch. 18.1-18.4

Deep Learning in Physics: Searching for Exotic Particles

Thanks to Pierre Baldi






Thanks to
Pierre Baldi

nature COMMUNICATIONS

ARTICLE

Received 19 Feb 2014 | Accepted 4 Jun 2014 | Published 2 Jul 2014

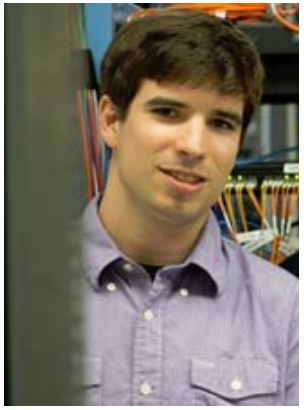
DOI: 10.1038/ncomms5308

Thanks to Pierre Baldi

Daniel Whiteson

Searching for exotic particles in high-energy physics with deep learning

P. Baldi[1], P. Sadowski[1] & D. Whiteson[2]

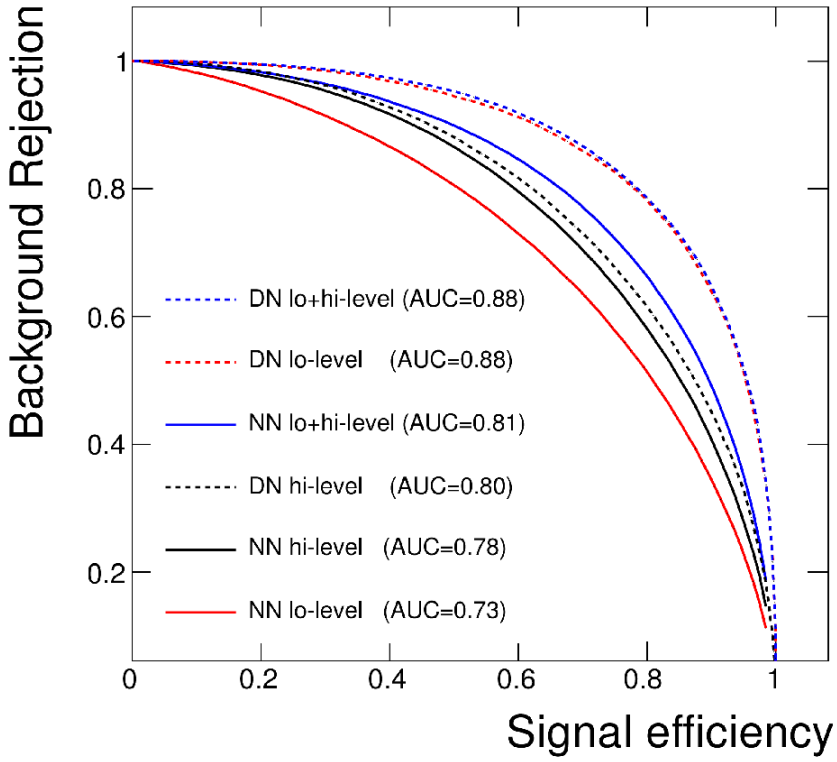


Peter Sadowski

Collisions at high-energy particle colliders are a traditionally fruitful source of exotic particle discoveries. Finding these rare particles requires solving difficult signal-versus-background classification problems, hence machine-learning approaches are often used. Standard approaches have relied on 'shallow' machine-learning models that have a limited capacity to learn complex nonlinear functions of the inputs, and rely on a painstaking search through manually constructed nonlinear features. Progress on this problem has slowed, as a variety of techniques have shown equivalent performance. Recent advances in the field of deep learning make it possible to learn more complex functions and better discriminate between signal and background classes. Here, using benchmark data sets, we show that deep-learning methods need no manually constructed inputs and yet improve the classification metric by as much as 8% over the best current approaches. This demonstrates that deep-learning approaches can improve the power of collider searches for exotic particles.

Higgs Boson Detection

Thanks to Pierre Baldi



	AUC		
Technique	Low-level	High-level	Complete
BDT	0.73	0.78	0.81
NN	0.733 (0.007)	0.777 (0.001)	0.816 (0.004)
DN	0.880 (0.001)	0.800 (< 0.001)	0.885 (0.002)

Deep network improves AUC by 8%

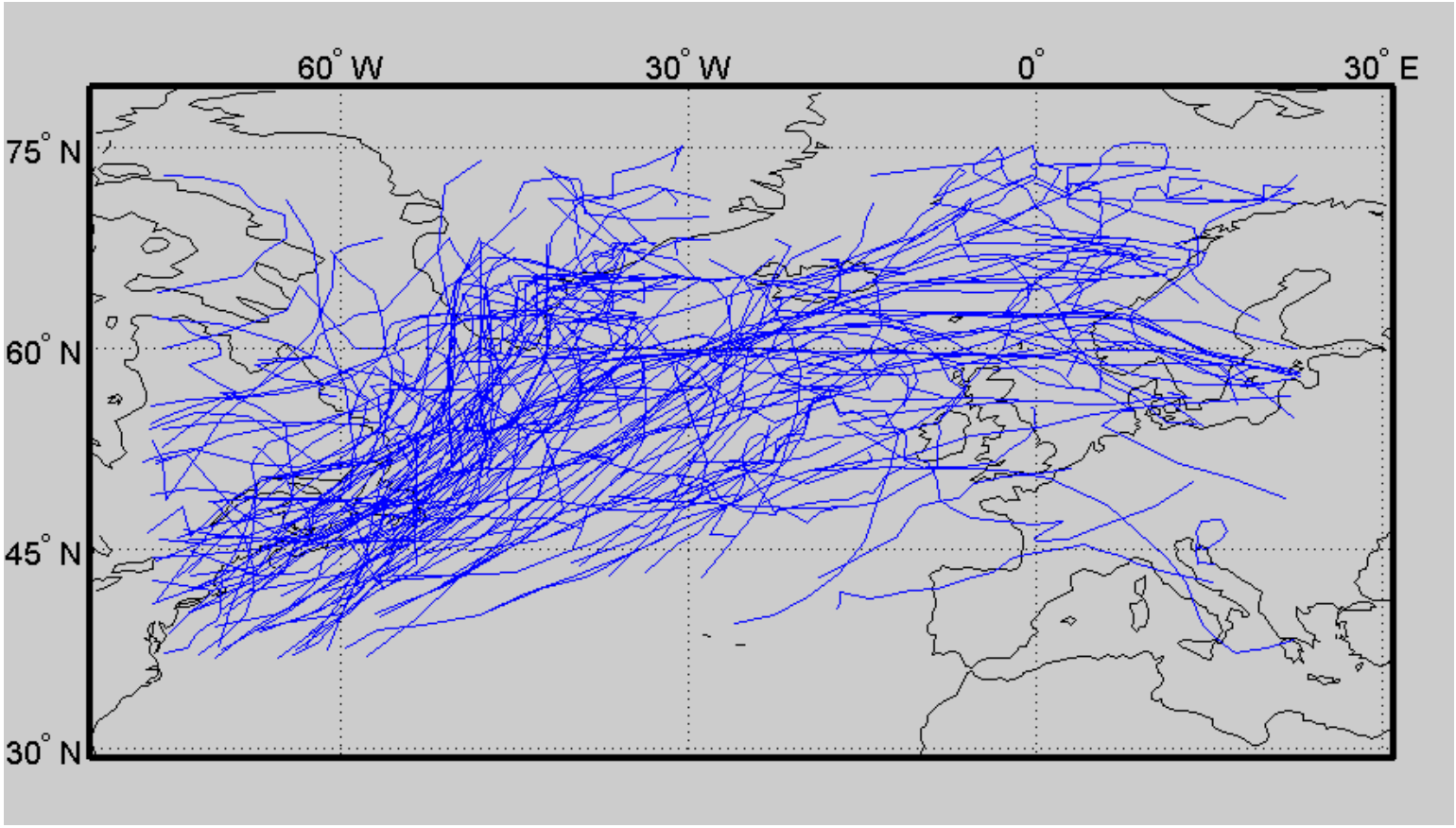
BDT= Boosted Decision Trees in TMVA package

Nature Communications, July 2014

Thanks to Padhraic Smyth

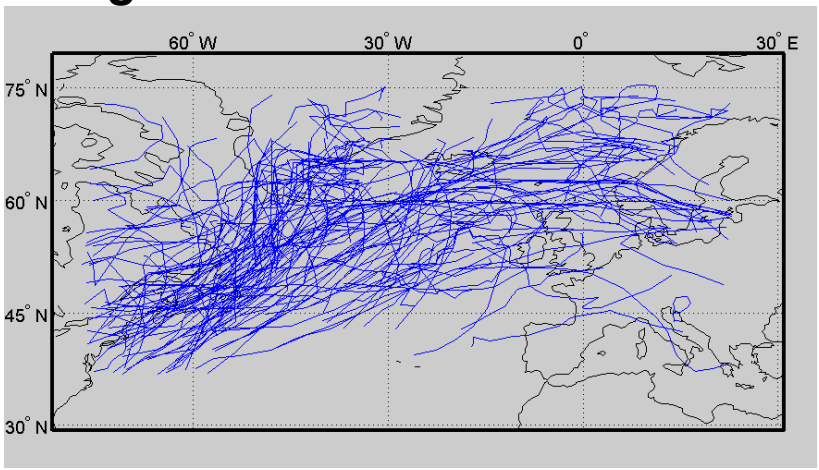
Application to Extra-Tropical Cyclones

Gaffney et al, *Climate Dynamics*, 2007

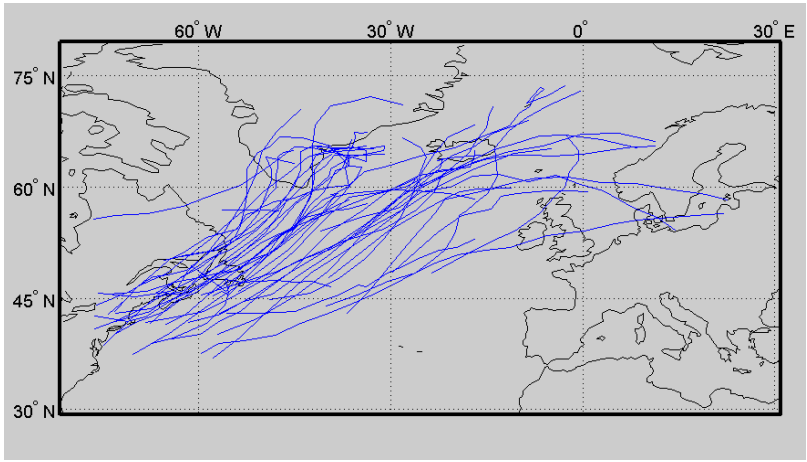


Thanks to
Padhraic Smyth

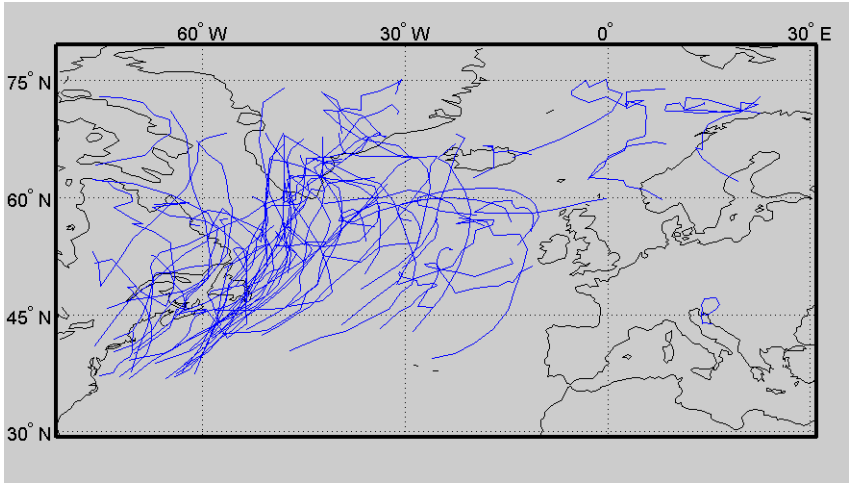
Original Data



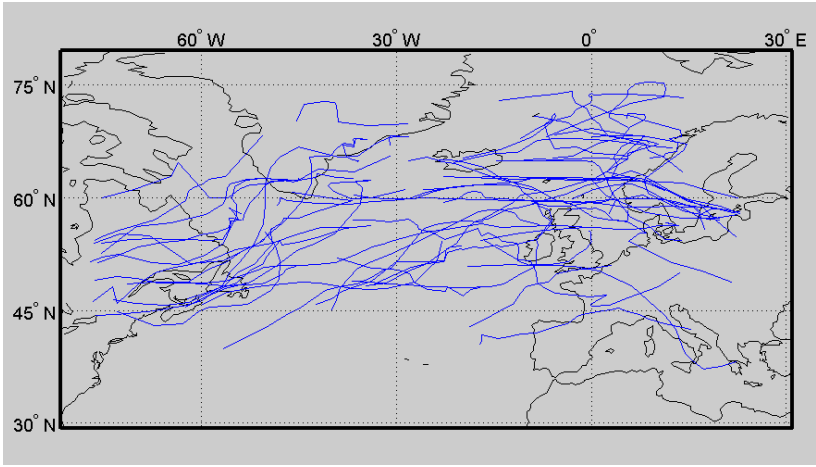
Iceland Cluster



Greenland Cluster

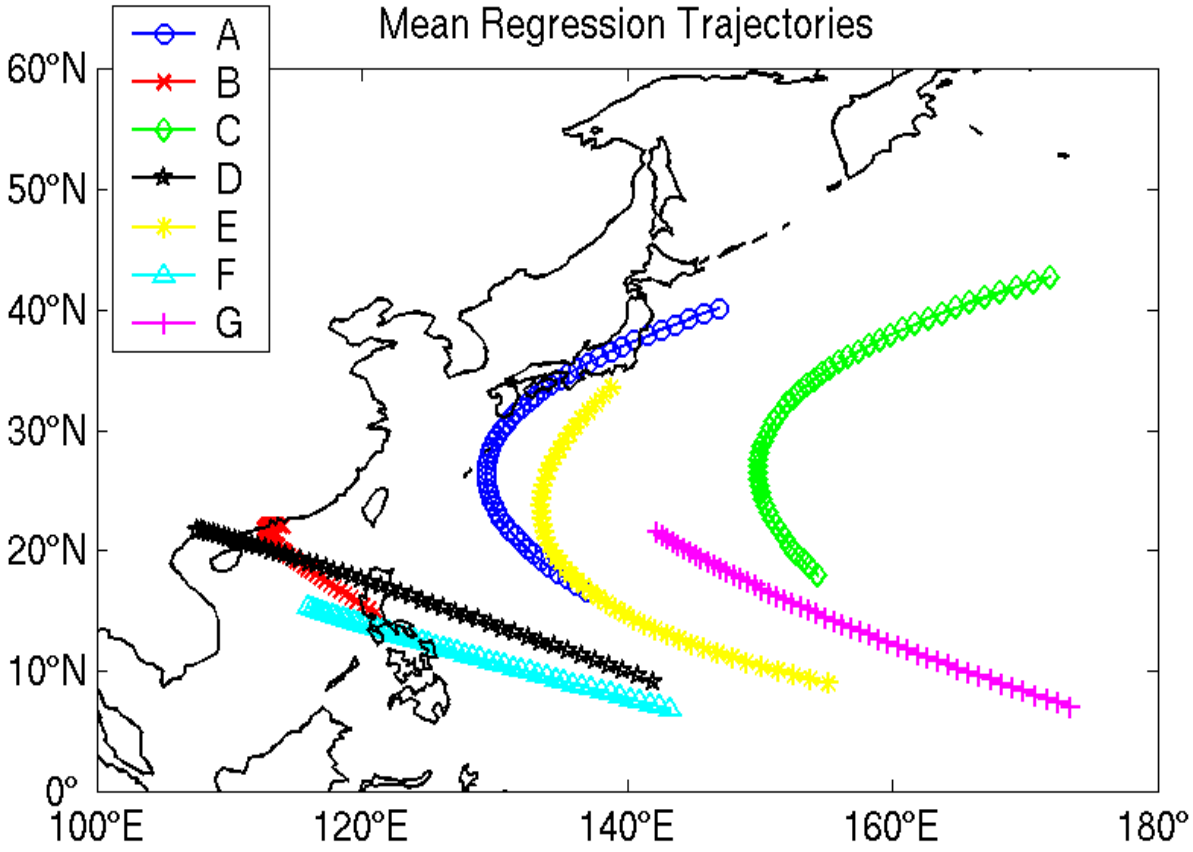


Horizontal Cluster

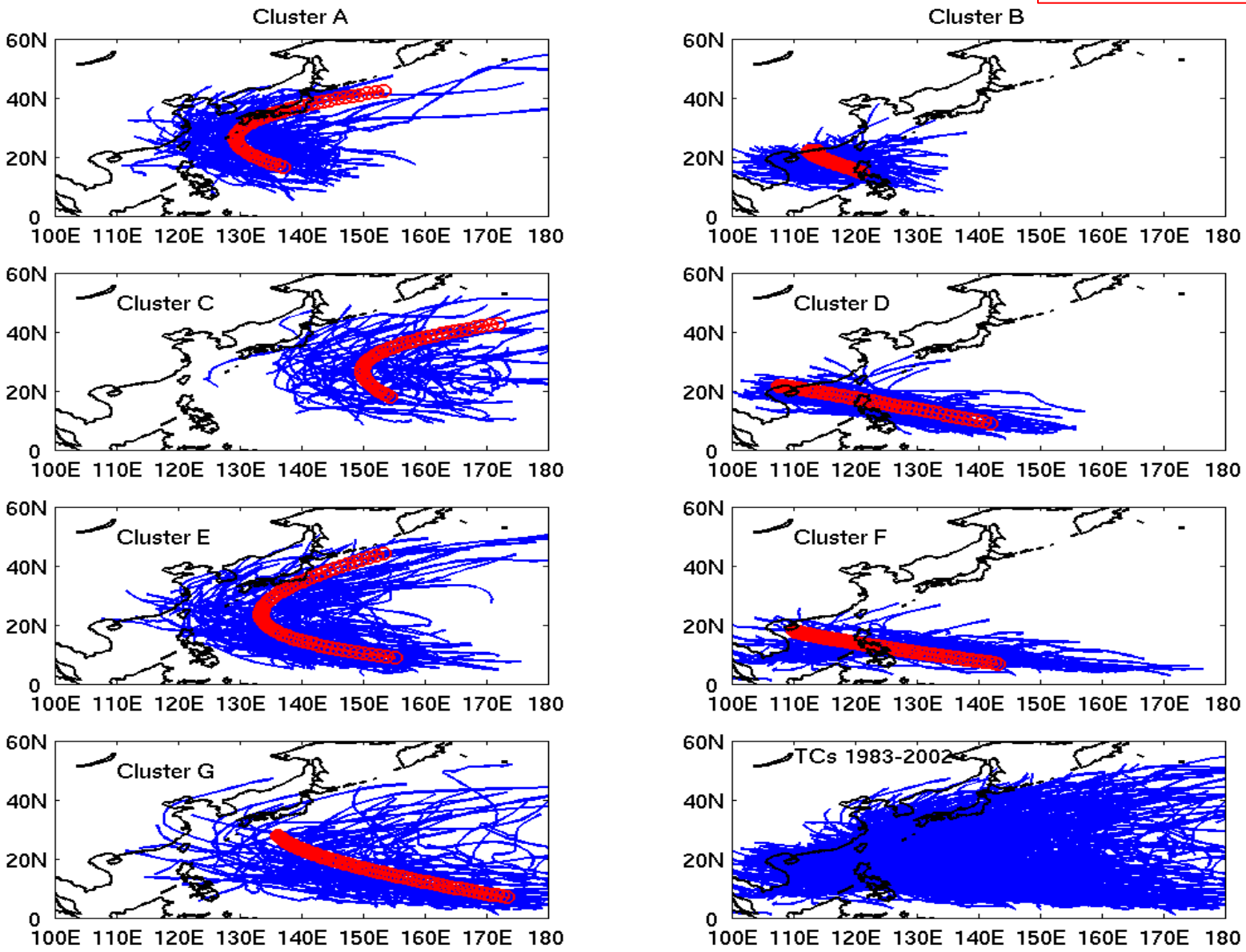


Thanks to
Padhraic Smyth

Cluster Shapes for Pacific Typhoon Tracks



Thanks to Padhraic Smyth



Thanks to Padhraic Smyth

An ICS Undergraduate Success Story

"The key student involved in this work started out as an ICS undergrad. Scott Gaffney took ICS 171 and 175, got interested in AI, started to work in my group, decided to stay in ICS for his PhD, did a terrific job in writing a thesis on curve-clustering and working with collaborators in climate science to apply it to important scientific problems, and is now one of the leaders of Yahoo! Labs reporting directly to the CEO there, http://labs.yahoo.com/author/gaffney/. Scott grew up locally in Orange County and is someone I like to point as a great success story for ICS."

--- From Padhraic Smyth

Handwritten Hangul recognition using deep convolutional neural networks

Thanks to Xiaohui Xie

In-Jung Kim & Xiaohui Xie

Fig. 1 Examples of Hangul characters

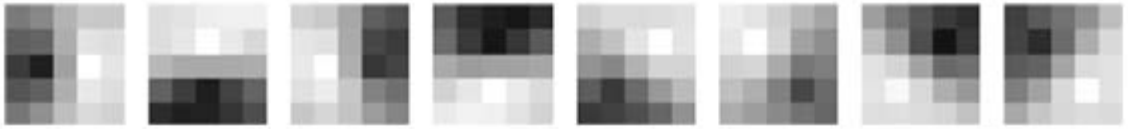


Fig. 4 Edge operators used to initialize convolution masks of the bottom layer

Thanks to Xiaohui Xie

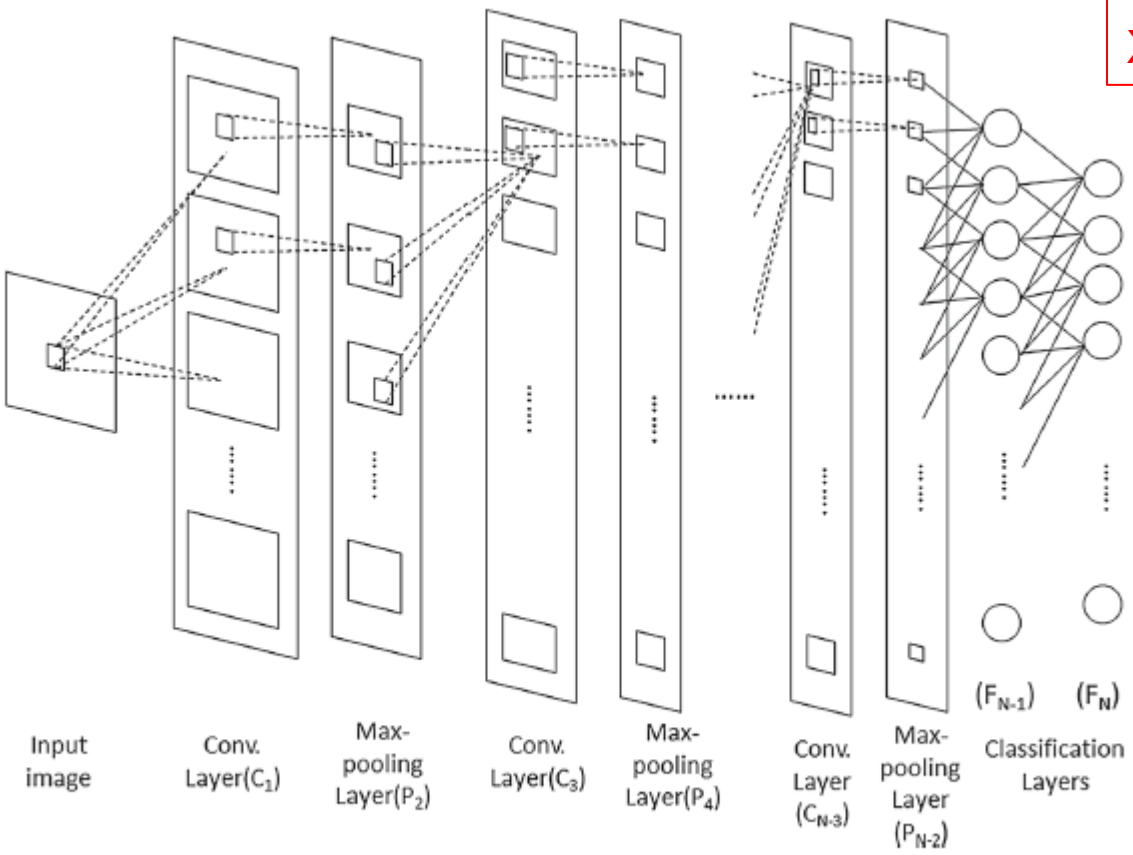
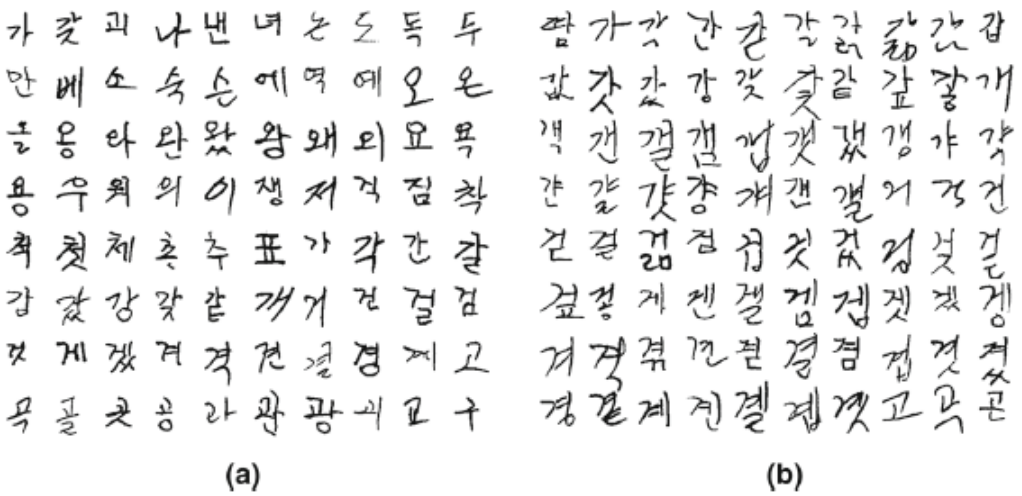


Fig. 2 The overall architecture of the DCNN used by us, which includes an input layer, multiple alternating convolution and max-pooling layers, and two fully connected classification layers. *N* denotes the total number of layers in the network



Thanks to Xiaohui Xie

Fig. 5 Example images in SERI95a and PE92 databases

Table 5 Structure of digit recognizer (MNIST)

Layer	Type	# of feature maps	Feature map size	Window size	Stride	# of parameters
C_1	Convolution	32	28 × 28	5 × 5	1	832
P_2	Max-pooling	32	14 × 14	2 × 2	2	0
C_3	Convolution	32	10 × 10	5 × 5	1	25,632
P_4	Max-pooling	32	5 × 5	2 × 2	2	0
C_5	Convolution	256	1 × 1	5 × 5	1	205,056
F_6	Fully connected	256	1 × 1	N/A	N/A	65,792
F_7	Fully connected	10	1 × 1	N/A	N/A	2,570

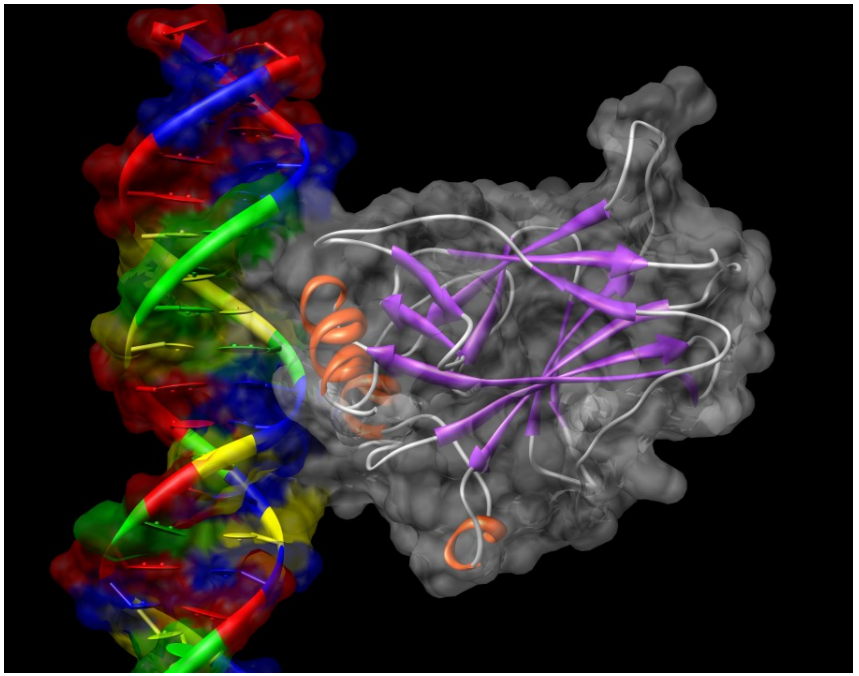
Table 6 Digit recognition results

	MSE		CE	
	Recog. rate (%)	Error rate (%)	Recog. rate (%)	Error rate (%)
Baseline	99.29	0.71	99.22	0.78
Edge operators	99.31	0.69	99.28	0.72
Elastic distortion	99.51	0.49	99.63	0.37
Edge operators + elastic distortion	99.65	0.35	99.67	0.33

p53 and Human Cancers

Thanks to Richard Lathrop

- p53 is a central tumor suppressor protein
  "The guardian of the genome"
- Cancer Mutants:
  About 50% of all human cancers have p53 mutations.
- Rescue Mutants:
  Several second-site mutations restore functionality to some p53 cancer mutants *in vivo.*



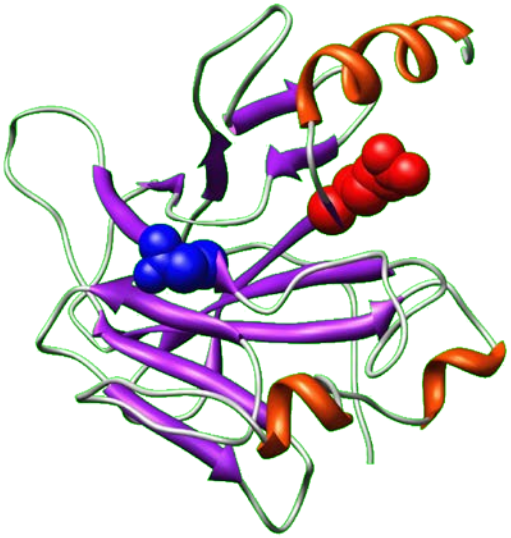
p53 core domain bound to DNA

Image Generated with UCSF Chimera

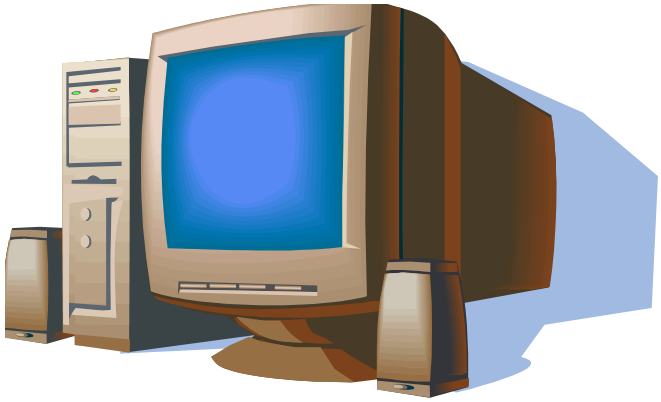
Cho, Y., **Gorina, S.**, **Jeffrey, P.D.**, **Pavletich, N.P.** Crystal structure of a p53 tumor suppressor-DNA complex: understanding tumorigenic mutations. *Science v265 pp.346-355 , 1994*

Active Learning for Biological Discovery

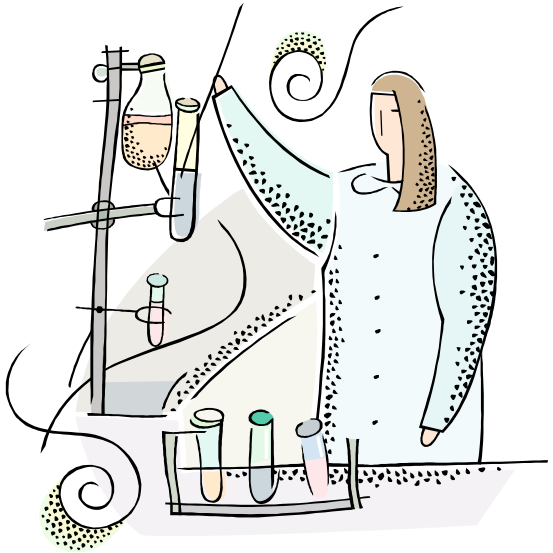
Thanks to
Richard Lathrop



Find Cancer Rescue Mutants



Knowledge

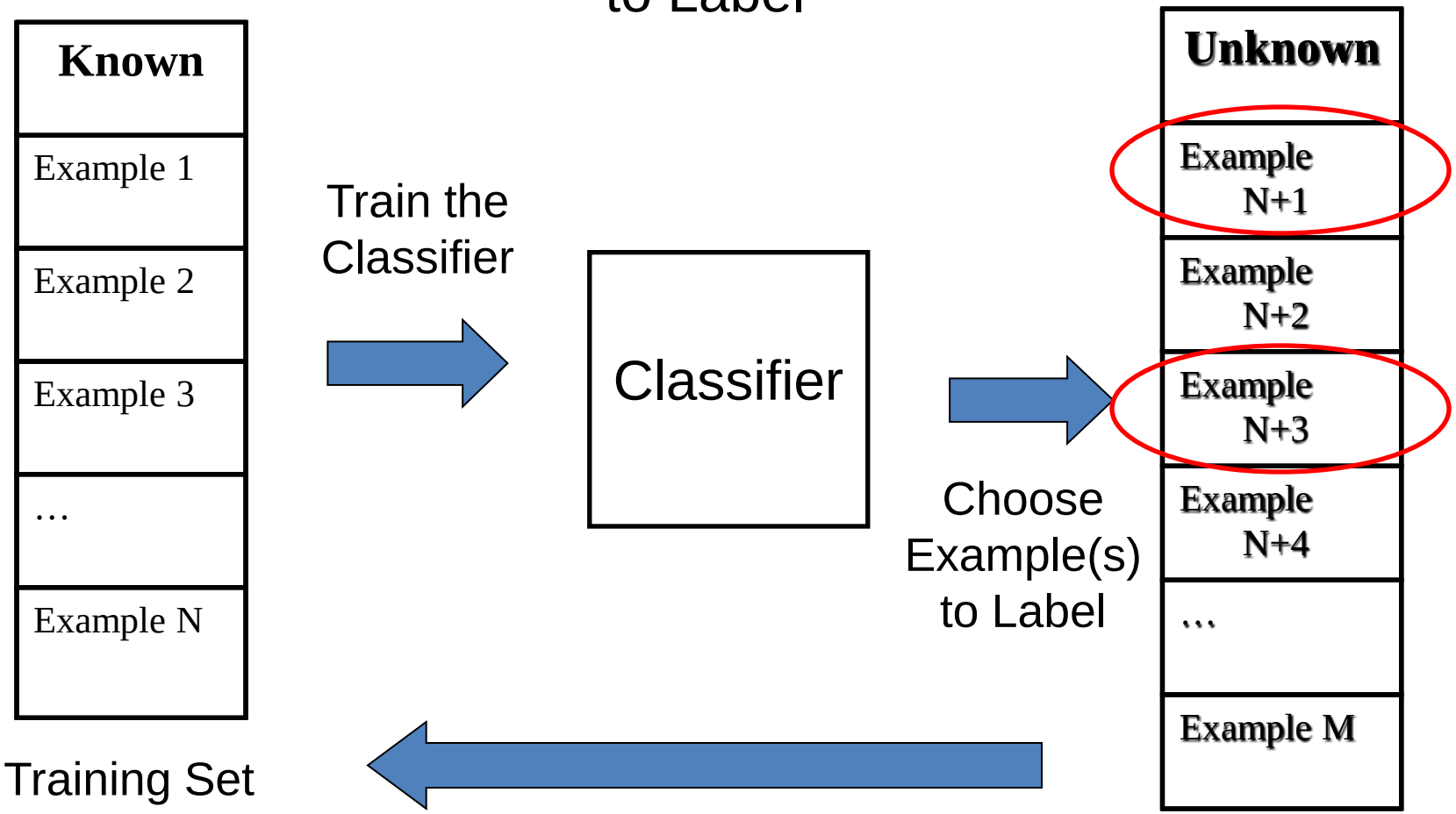


Theory

Experiment

Computational Active Learning

Pick the Best (= Most Informative) Unknown Examples to Label



Visualization of Selected Regions

Thanks to Richard Lathrop

- **Positive Region:**

  Predicted Active
  96-105 (Green)

- **Negative Region:**

  Predicted Inactive
  223-232 (Red)

- **Expert Region:**

  Predicted Active
  114-123 (Blue)

Novel Single-a.a. Cancer Rescue Mutants

Thanks to Richard Lathrop

	MIP Positive (96-105)	MIP Negative (223-232)	Expert (114-123)
# Strong Rescue	8	0 ($p < 0.008$)	6 (not significant)
# Weak Rescue	3	2 (not significant)	7 (not significant)
Total # Rescue	11	2 ($p < 0.022$)	13 (not significant)

No significant differences between the MIP Positive and Expert regions.

Both were statistically significantly better than the MIP Negative region.

The Positive region rescued for the first time the cancer mutant P152L.

No previous single-a.a. rescue mutants in any region.

Complete architectures for intelligence?

- Search?
  - Solve the problem of what to do.
- Learning?
  - Learn what to do.
- Logic and inference?
  - Reason about what to do.
  - Encoded knowledge/”expert” systems?
    - Know what to do.
- Modern view: It’s complex & multi-faceted.

Automated Learning

- Why learn?
  - Key hallmark of intelligence
  - Take real data → get feedback → improve performance → repeat
  - Check out USC Autonomous Flying Vehicle Project!

- Types of learning
  - **Supervised learning:** learn mapping, attributes → target
    - Classification: target variable is discrete (e.g., spam email)
    - Regression: target variable is real-valued (e.g., stock market)
  - **Unsupervised learning:** understand hidden data structure
    - Clustering: group data into "similar" groups
    - Latent space embedding: learn a simple data representation
  - **Other types of learning**
    - Reinforcement learning: e.g., game-playing agent
    - Learning to rank, e.g., document ranking in Web search
    - And many others….

Importance of representation

- Definition of “state” can be very important
- A good representation
  - Reveals important features
  - Hides irrelevant detail
  - Exposes useful constraints

  } **Most important**

  - Makes frequent operations easy to do
  - Supports local inferences from local features
    - Called “soda straw” principle, or “locality” principle
    - Inference from features “through a soda straw”
  - Rapidly or efficiently computable
    - It’s nice to be fast

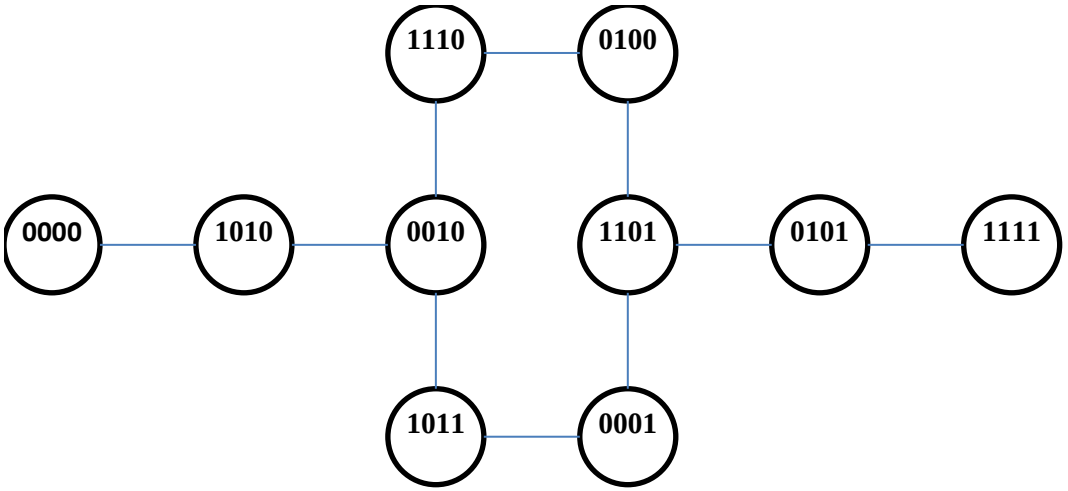
Reveals important features / Hides irrelevant detail

"You can't learn what you can't represent." --- G. Sussman

- **In search:** *A man is traveling to market with a fox, a goose, and a bag of oats. He comes to a river. The only way across the river is a boat that can hold the man and exactly one of the fox, goose or bag of oats. The fox will eat the goose if left alone with it, and the goose will eat the oats if left alone with it.*

How can the man get all his possessions safely across the river?

- **A good representation makes this problem easy:**



MFGO

M = man
F = fox
G = goose
O = oats
0 = starting side
1 = ending side

Reveals important features / Hides irrelevant detail

"You can't learn what you can't represent." --- G. Sussman

- **In logic:**

  If the unicorn is mythical, then it is immortal, but if it is not mythical, then it is a mortal mammal. If the unicorn is either immortal or a mammal, then it is horned. The unicorn is magical if it is horned.

  ⇒ ***Prove that the unicorn is both magical and horned.***

- **A good representation makes this problem easy:**

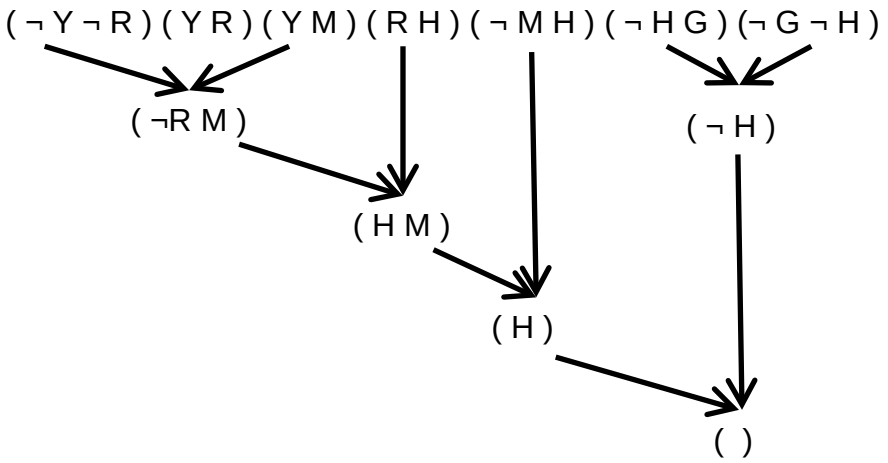
  Y = unicorn is m<u>Y</u>thical

  R = unicorn is mo<u>R</u>tal

  M = unicorn is a ma<u>M</u>mal

  H = unicorn is <u>H</u>orned

  G = unicorn is ma<u>G</u>ical



Simple illustrative learning problem

Problem:

Decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Training Data for Supervised Learning

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Terminology

- Attributes
  - Also known as features, variables, independent variables, covariates
- Target Variable
  - Also known as goal predicate, dependent variable, …
- Classification
  - Also known as discrimination, supervised classification, …
- Error function
  - Also known as objective function, loss function, …

Inductive or Supervised learning

- Let x = input vector of attributes (feature vectors)

- Let f(x) = target label
  - The implicit mapping from x to f(x) is unknown to us
  - We only have training data pairs, D = {**x**, **f(x)**} available

- We want to learn a mapping from x to f(x)
  - Our hypothesis function is h(x, θ)
  - h(x, θ) ≈ f(x) for all training data points x
  - θ are the parameters of our predictor function h

- Examples:
  - h(x, θ) = sign($\theta_1 x_1 + \theta_2 x_2 + \theta_3$) (perceptron)
  - h(x, θ) = $\theta_0 + \theta_1 x_1 + \theta_2 x_2$ (regression)
  - $h_k(x) = (x_1 \wedge x_2) \vee (x_3 \wedge \neg x_4)$

Empirical Error Functions

- $E(h) = \Sigma_x$ distance[$h(x, \theta)$, $f(x)$]

  Sum is over all training pairs in the training data D

  Examples:

  distance = squared error if h and f are real-valued (regression)

  distance = delta-function if h and f are categorical (classification)

  In learning, we get to choose

  1. what class of functions h(..) we want to learn
     – potentially a huge space! ("hypothesis space")

  2. what error function/distance we want to use
     - should be chosen to reflect real "loss" in problem
     - but often chosen for mathematical/algorithmic convenience

Inductive Learning as Optimization or Search

- Empirical error function:

  $E(h) = \Sigma_x$ distance[$h(x, \theta)$, $f(x)$]

- Empirical learning = finding $h(x)$, or $h(x; \theta)$ that minimizes $E(h)$
  - In simple problems there may be a closed form solution
    - E.g., "normal equations" when h is a linear function of x, E = squared error
  - If $E(h)$ is **differentiable** → continuous optimization problem using gradient descent, etc
    - E.g., multi-layer neural networks
  - If $E(h)$ is **non-differentiable** (e.g., classification) → systematic search problem through the space of functions h
    - E.g., decision tree classifiers
- Once we decide on what the functional form of h is, and what the error function E is, then machine learning typically reduces to a large search or optimization problem
- Additional aspect: we really want to learn a function h that will generalize well to new data, not just memorize training data – will return to this later

Our training data example (again)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- If all attributes were binary, h(..) could be any arbitrary Boolean function
- Natural error function E(h) to use is classification error, i.e., how many incorrect predictions does a hypothesis h make
- Note an implicit assumption:
  - For any set of attribute values there is a unique target value
  - This in effect assumes a “no-noise” mapping from inputs to targets
    - This is often not true in practice (e.g., in medicine). Will return to this later

Learning Boolean Functions

William of Ockham
c. 1288-1347

- Given examples of the function, can we learn the function?

  $f : \mathbf{B}^d \rightarrow \mathbf{B}$ where $\mathbf{B} = \{0, 1\}$

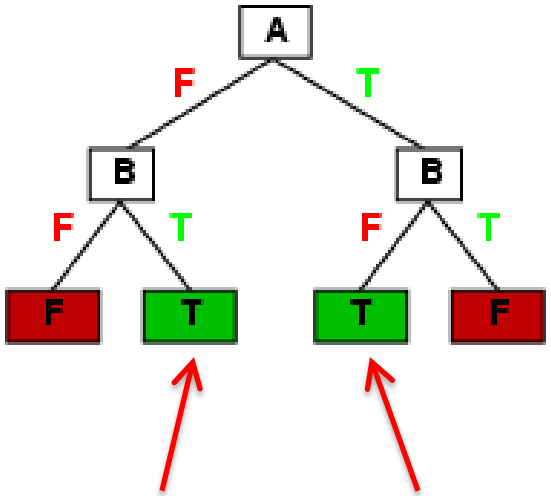
- How many Boolean functions can be defined on **d attributes**?
  - Boolean function = Truth table + binary target column
  - Truth table $\mathbf{2}^{\boldsymbol{d}}$ rows + binary $\mathbf{2}^{\boldsymbol{d}}$ targets = $\mathbf{2}^{\mathbf{2}^{\boldsymbol{d}}}$
  - $\mathbf{2}^{\mathbf{2}^{\boldsymbol{d}}}$ hypothesis search space
  - i.e. for d = 6, there are 1.84 x 10^{19} possible Boolean functions

- Observations:
  - Huge hypothesis spaces –> directly searching over all functions is impossible
  - Given a small data (n pairs) our learning problem may be under constrained
    - Ockham's razor: if multiple candidate functions all explain the data equally well, pick the simplest explanation (least complex function)
    - Constrain our search to classes of Boolean functions, i.e.,
      - decision trees
      - Weighted linear sums of inputs (e.g., perceptron's)

Decision Tree Representations

- Decision trees are fully expressive
  - Can represent any Boolean function (in DNF)
  - Every path in the tree could represent 1 row in the truth table
  - Might yield an exponentially large tree
    - Truth table is of size 2^d, where d is the number of attributes

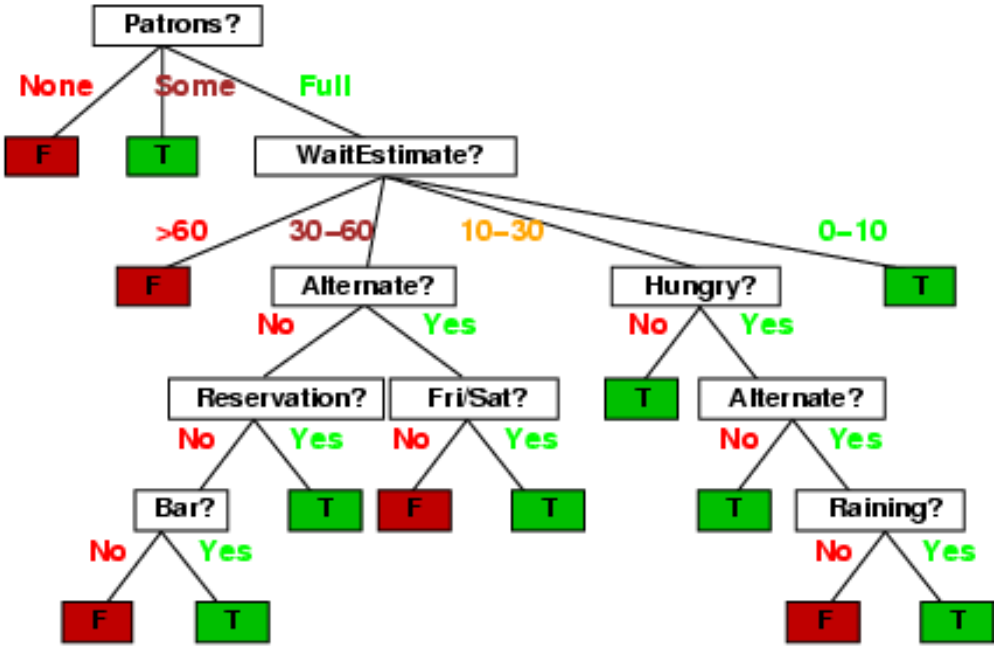
A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



A xor B = (¬ A ∧ B) ∨ (A ∧ ¬ B) in DNF

Decision Tree Learning

- Constrain h(..) to be a decision tree
  - This is the R&N tree for the Restaurant Wait problem:



Decision Tree Representations

- Decision trees are DNF representations
  - often used in practice → often result in compact approximate representations for complex functions
  - E.g., consider a truth table where most of the variables are irrelevant to the function

- Simple DNF formulae can be easily represented
  - E.g., $f = (A \wedge B) \vee (\neg A \wedge D)$
  - DNF = disjunction of conjunctions

- Trees can be very inefficient for certain types of functions
  - Parity function: 1 only if an even number of 1's in the input vector
    - Trees are very inefficient at representing such functions
  - Majority function: 1 if more than ½ the inputs are 1's
    - Also inefficient

Decision Tree Learning

- Find the smallest decision tree consistent with the n examples
  - Unfortunately this is provably intractable to do optimally

- Greedy heuristic search used in practice:
  - Select root node that is “best” in some sense
  - Partition data into 2 subsets, depending on root attribute value
  - Recursively grow subtrees
  - Different termination criteria
    - For noiseless data, if all examples at a node have the same label then declare it a leaf and backup
    - For noisy data it might not be possible to find a “pure” leaf using the given attributes
      - we’ll return to this later – but a simple approach is to have a depth-bound on the tree (or go to max depth) and use majority vote

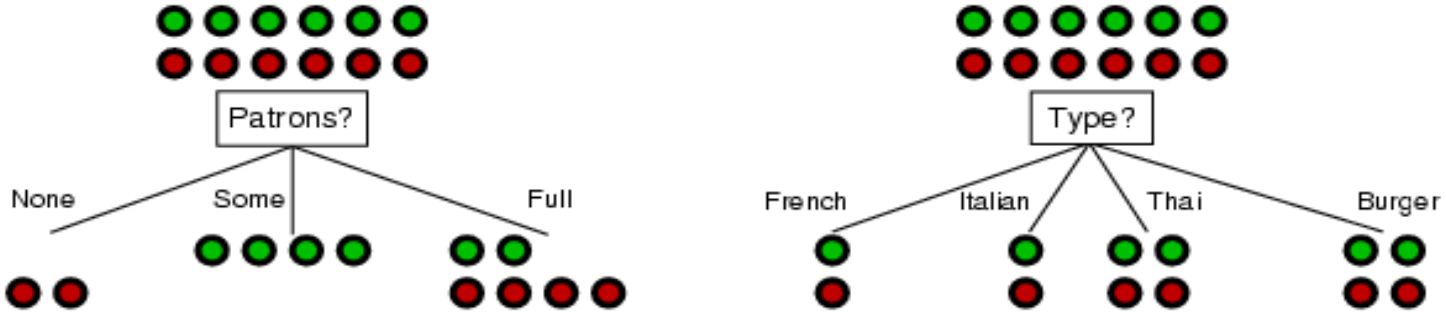
- We have talked about binary variables up until now, but we can trivially extend to multi-valued variables

Pseudocode for Decision tree learning

```
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
```

Choosing an attribute

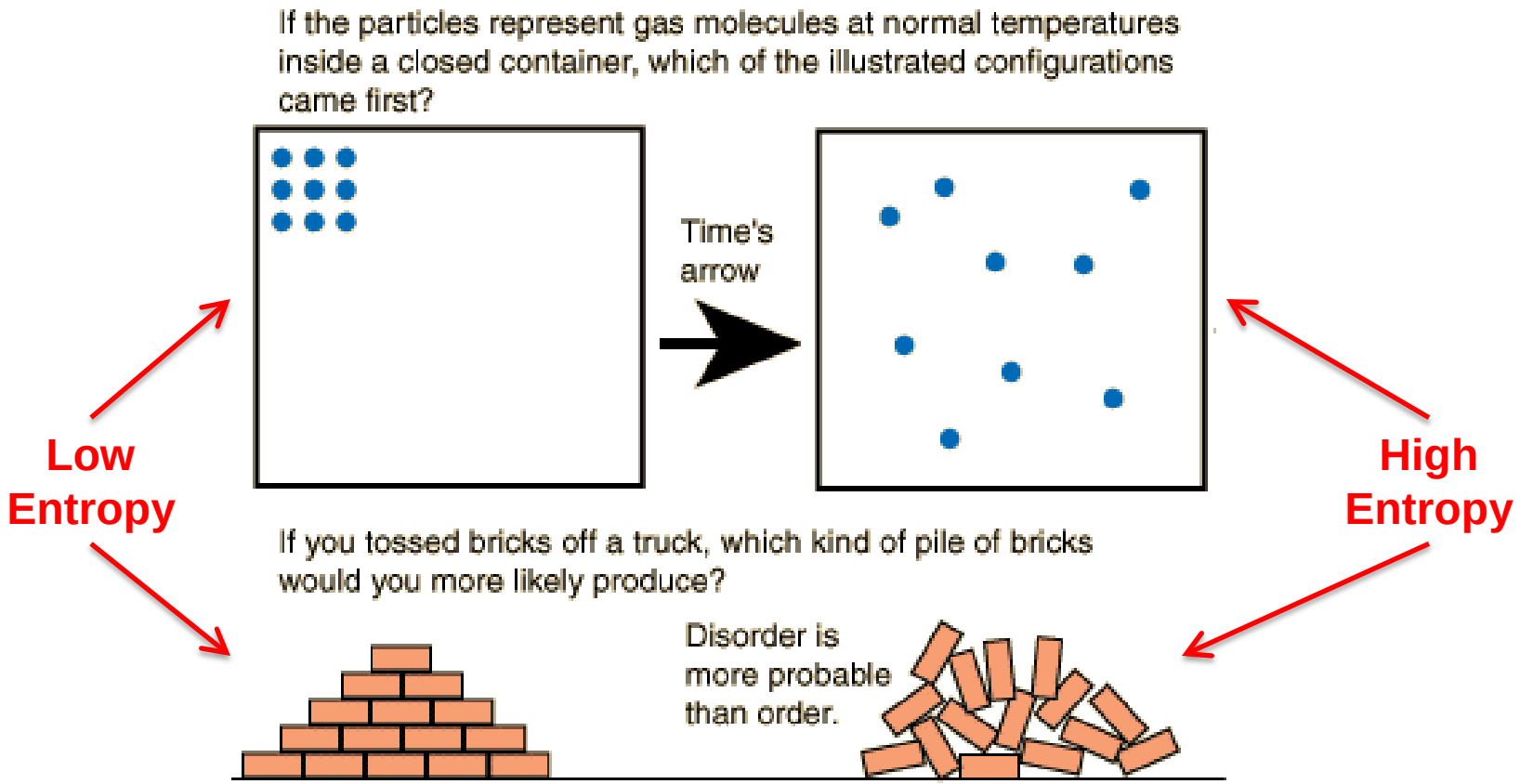
- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative"



- *Patrons?* is a better choice
  - How can we quantify this?
  - One approach would be to use the classification error E directly (greedily)
    - Empirically it is found that this works poorly
  - **Much better is to use information gain (next slides)**
  - Other metrics are also used, e.g., Gini impurity, variance reduction
    - Often very similar results to information gain in practice

Entropy and Information

- **"Entropy" is a measure of randomness = amount of disorder**



https://www.youtube.com/watch?v=ZsY4WcQOrfk

Entropy, H(p), with only 2 outcomes

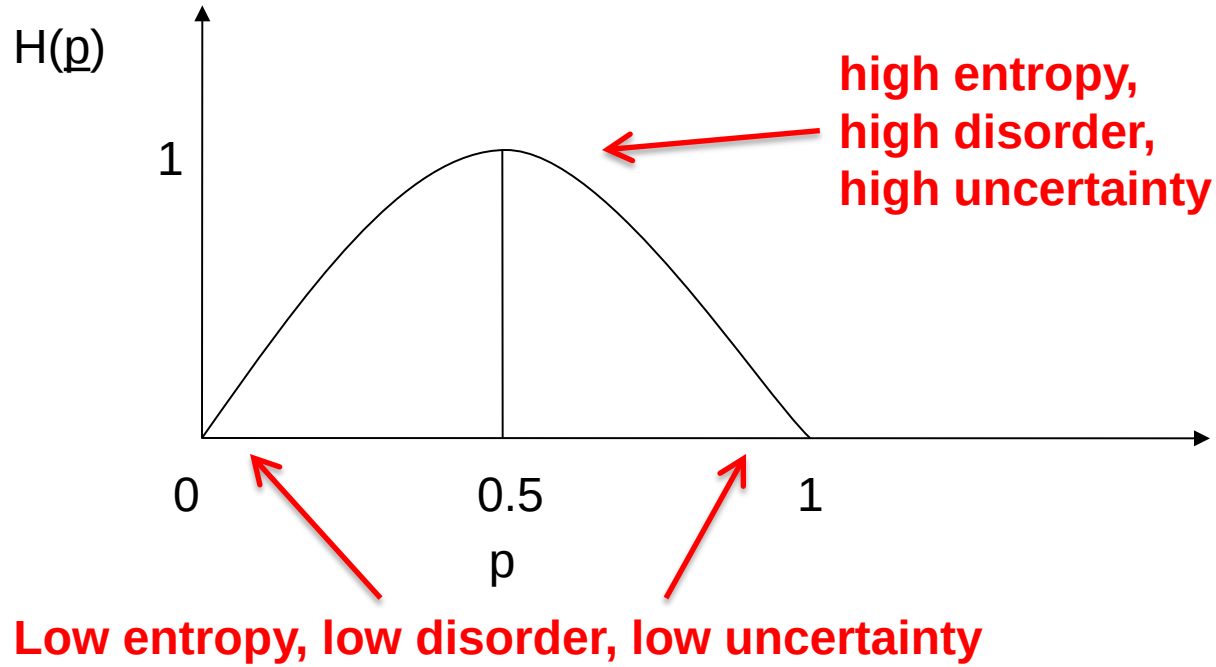
Consider 2 class problem:

p = probability of class #1,

1 – p = probability of class #2

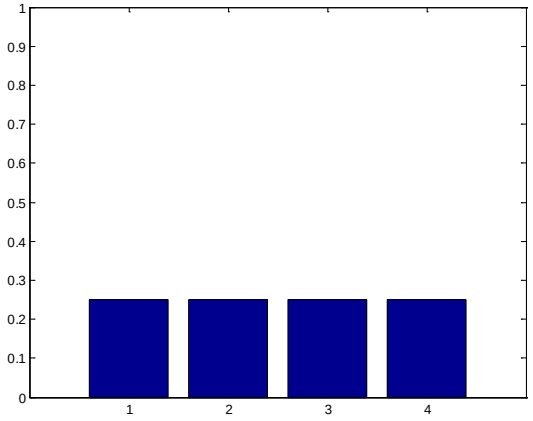
In binary case:

$$H(p) = -p \log p - (1-p) \log (1-p)$$

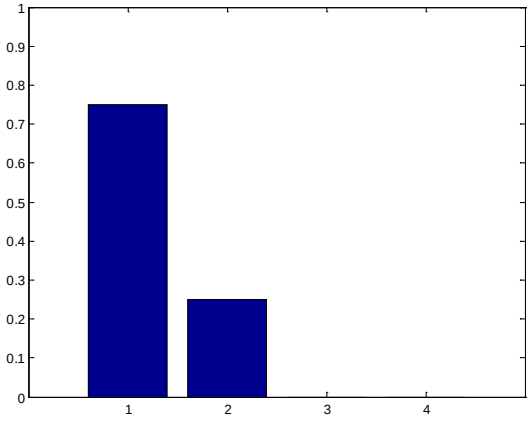


Entropy and Information

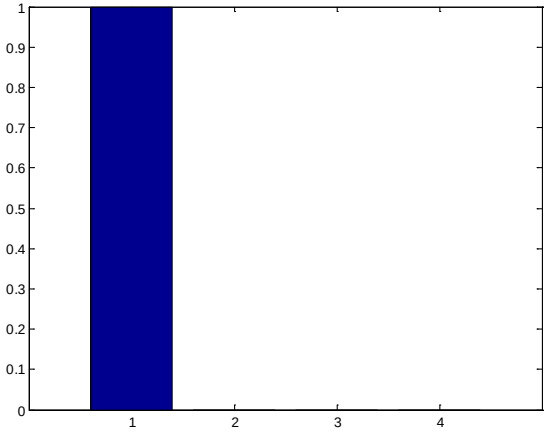
- **Entropy H(X) = E[log 1/P(X)] = $\sum_{x \in X}$ P(x) log 1/P(x)**
  = −$\sum_{x \in X}$ P(x) log P(x)
  - Log base two, units of entropy are "bits"
  - If only two outcomes: H(p) = – p log(p) – (1–p) log(1–p)
- Examples:



H(x) = .25 log 4 + .25 log 4 + .25 log 4 + .25 log 4 = log 4 = 2 bits



H(x) = .75 log 4/3 + .25 log 4 = 0.8133 bits



H(x) = 1 log 1 = 0 bits

Max entropy for 4 outcomes

Min entropy

Information Gain

- H(P) = current entropy of class distribution P at a particular node, before further partitioning the data

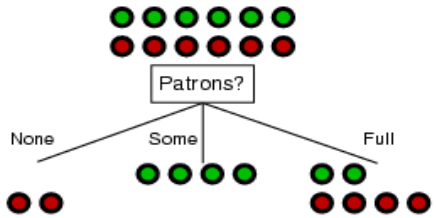
- H(P | A) = conditional entropy given attribute A
  = weighted average entropy of conditional class distribution, after partitioning the data according to the values in A

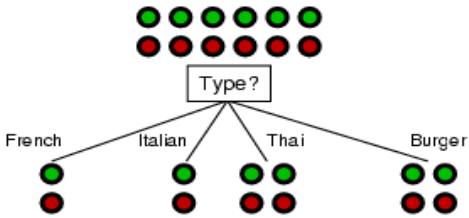
- Gain(A) = H(P) – H(P | A)
  - Sometimes written IG(A) = InformationGain(A)

- Simple rule in decision tree learning
  - **At each internal node, split on the node with the largest information gain [or equivalently, with smallest H(P|A)]**

- Note that by definition, conditional entropy can't be greater than the entropy, so Information Gain must be non-negative

Root Node Example





For the training set, *6 positives, 6 negatives, H(6/12, 6/12) = 1* bit

positive (p)

negative (1-p)

H(6/12,6/12) = -(6/12)*log2(6/12)-(6/12)*log2(6/12) = 1

Consider the attributes *Patrons* and *Type:*

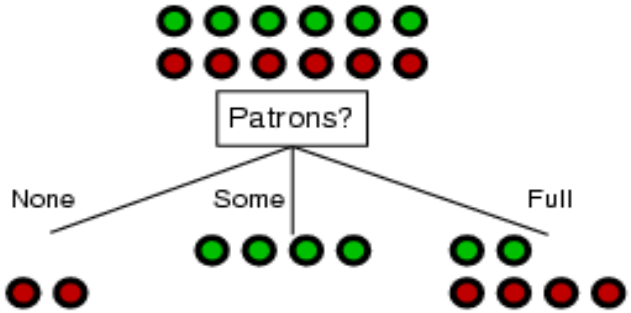
$$IG(\text{Patrons}) = 1 - \left[\frac{2}{12}H(0,1) + \frac{4}{12}H(1,0) + \frac{6}{12}H(\frac{2}{6},\frac{4}{6})\right] = 0.541 \text{ bits}$$

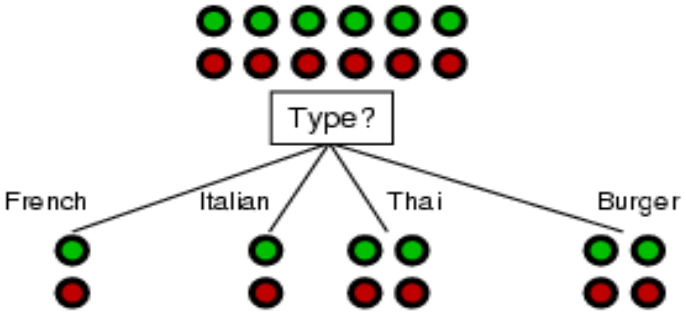
$$IG(\text{Type}) = 1 - \left[\frac{2}{12}H(\frac{1}{2},\frac{1}{2}) + \frac{2}{12}H(\frac{1}{2},\frac{1}{2}) + \frac{4}{12}H(\frac{2}{4},\frac{2}{4}) + \frac{4}{12}H(\frac{2}{4},\frac{2}{4})\right] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the learning algorithm as the root

Information gain is then repeatedly applied at internal nodes until all leaves contain only examples from one class or the other

Choosing an attribute



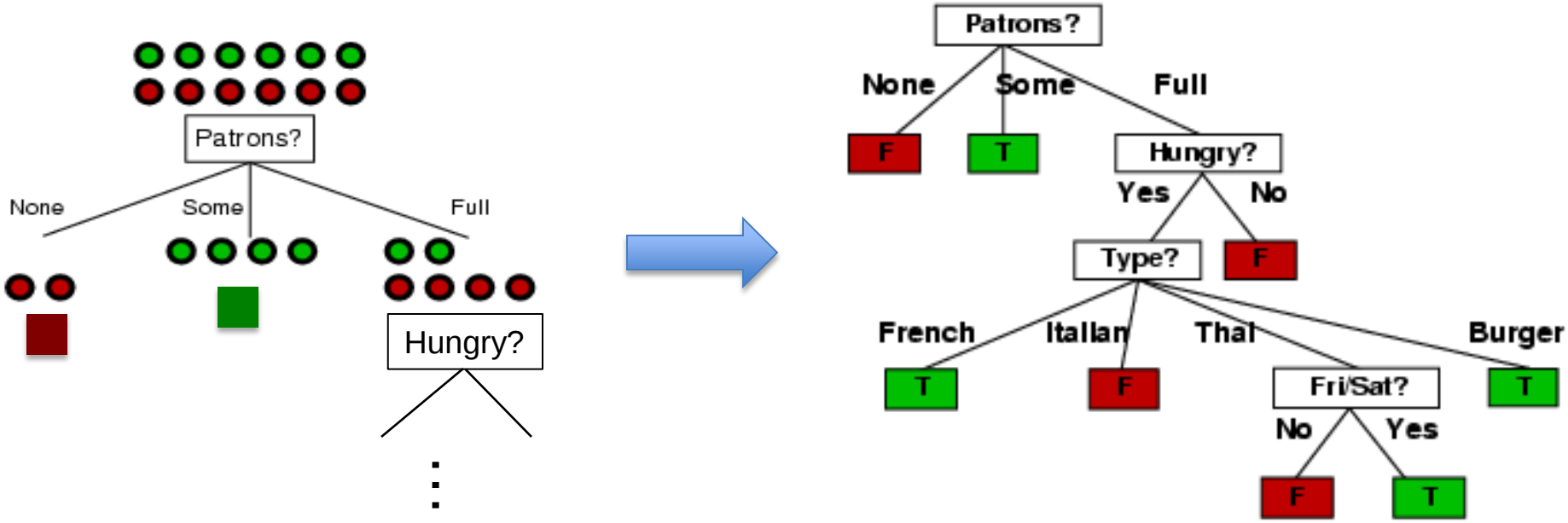


IG(Patrons) = 0.541 bits

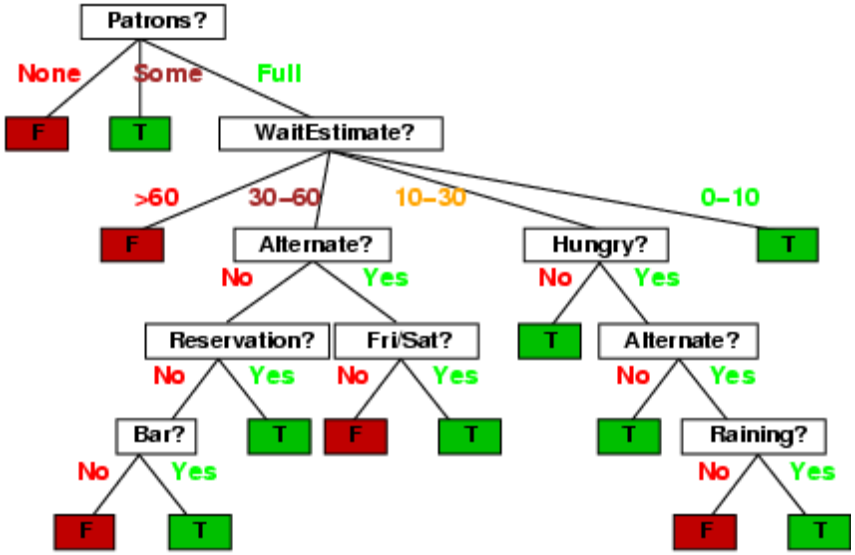
IG(Type) = 0 bits

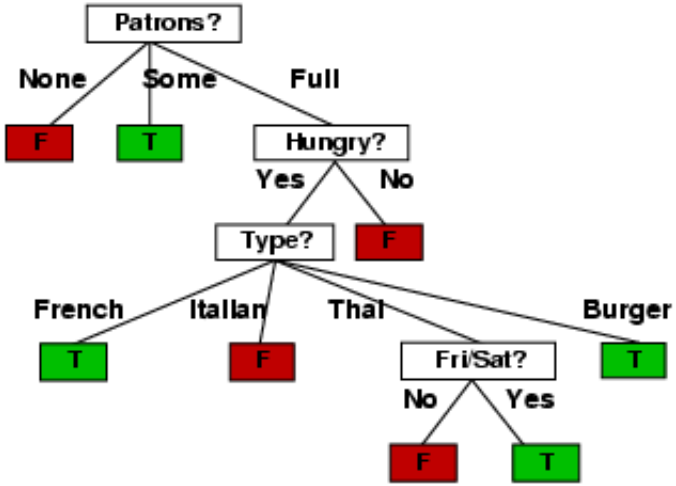
Decision Tree Learned

- Decision tree learned from the 12 examples:



R&N Tree (left) versus Learned Tree (right)





Assessing Performance

Training data performance is typically optimistic

e.g., error rate on training data

Reasons?

- classifier may not have enough data to fully learn the concept (but on training data we don't know this)
- for noisy data, the classifier may overfit the training data

In practice we want to assess performance "out of sample"

how well will the classifier do on new unseen data? This is the true test of what we have learned (just like a classroom)

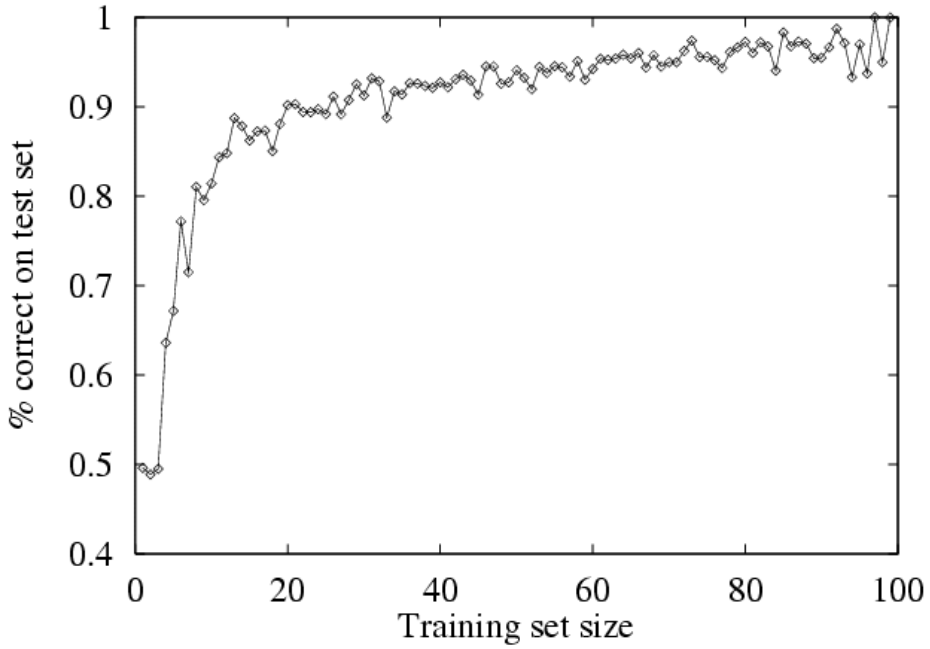
With large data sets we can partition our data into 2 subsets, train and test

- build a model on the training data
- assess performance on the test data

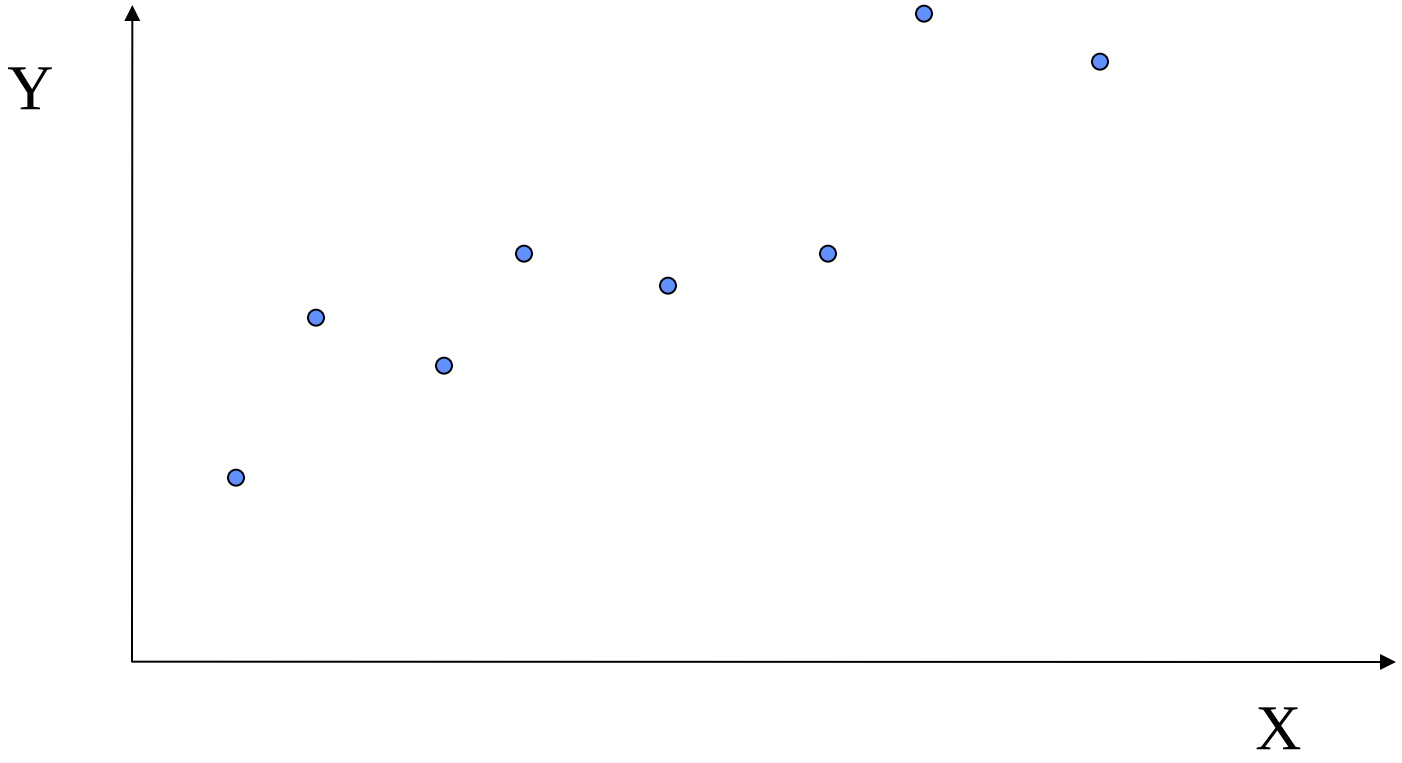
Example of Test Performance

Restaurant problem

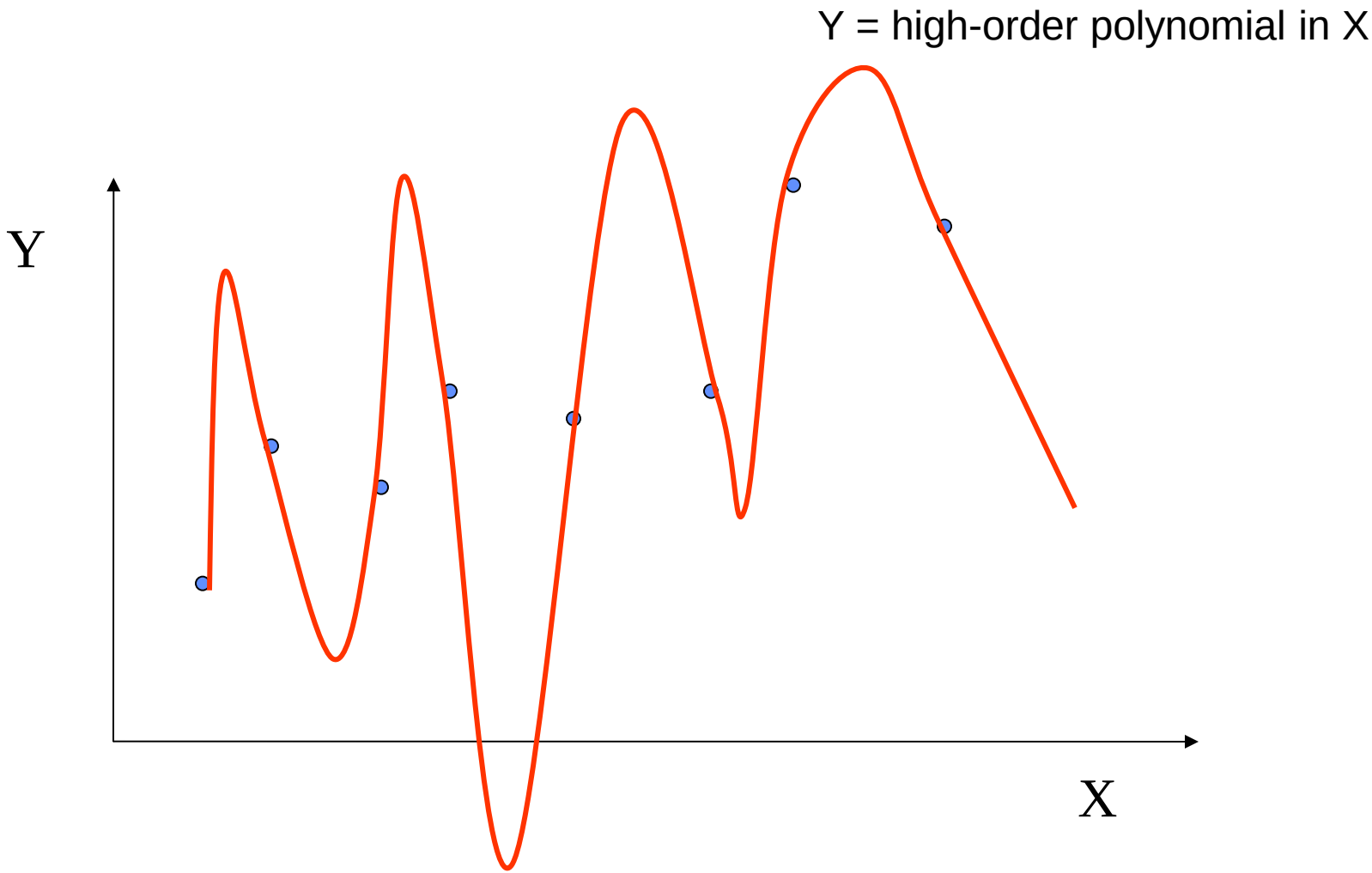
- simulate 100 data sets of different sizes
- train on this data, and assess performance on an independent test set
- learning curve = plotting accuracy as a function of training set size
- typical “diminishing returns” effect (some nice theory to explain this)



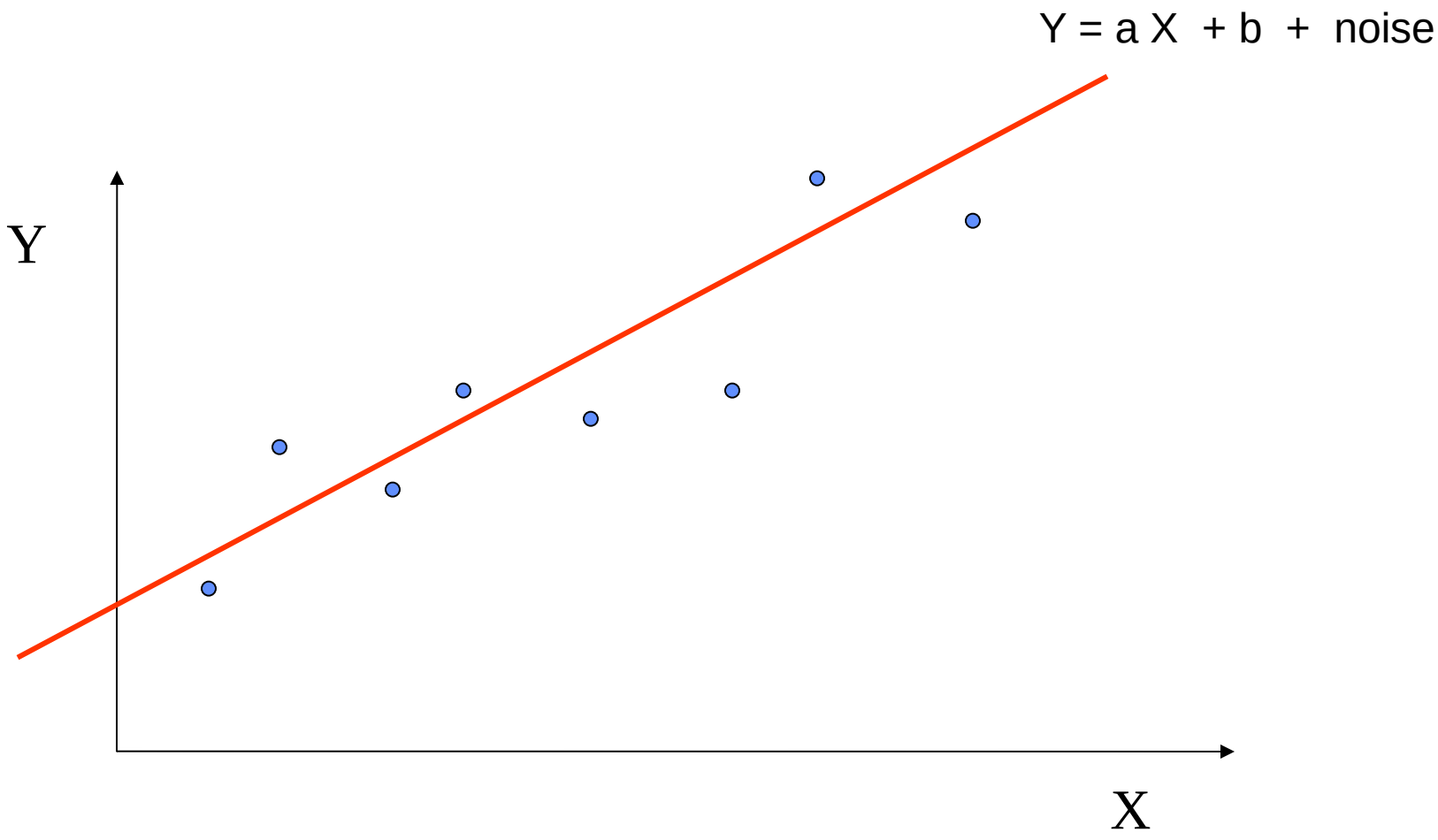
Overfitting and Underfitting



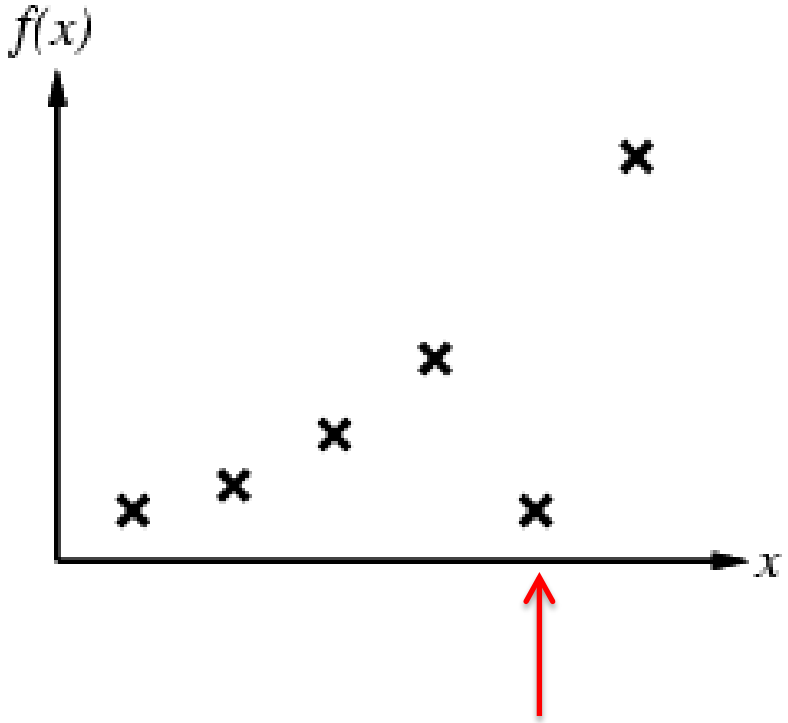
A Complex Model



A Much Simpler Model

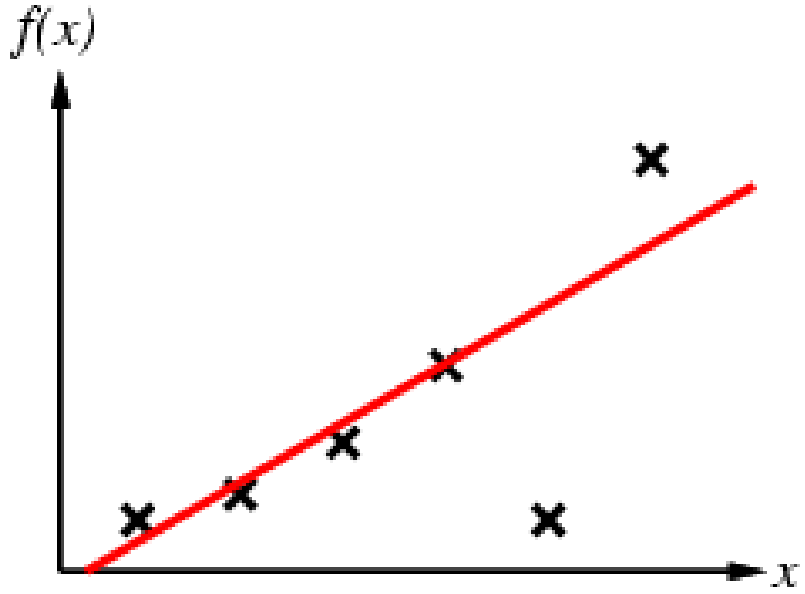


Example 2

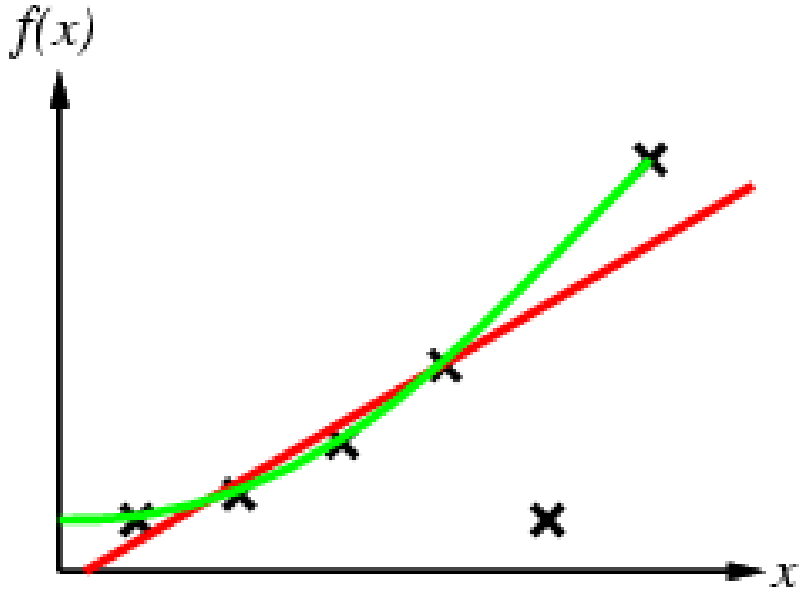


My biologist colleagues say, "Oh, that's the sample that we dropped on the floor!"

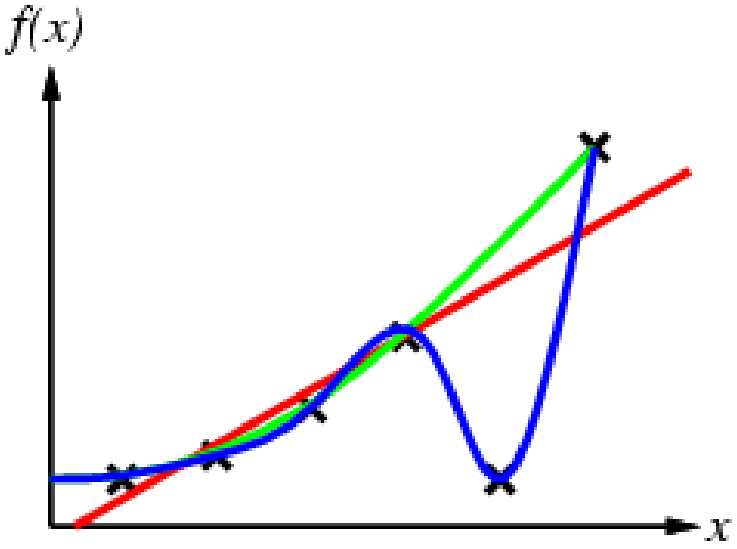
Example 2



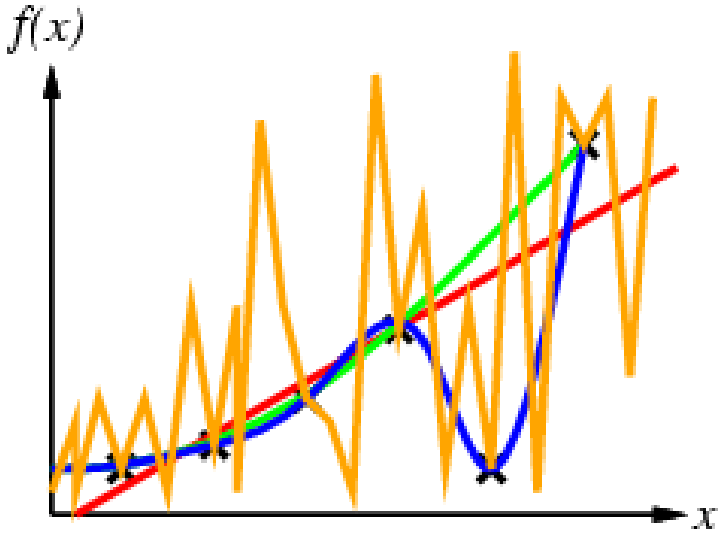
Example 2



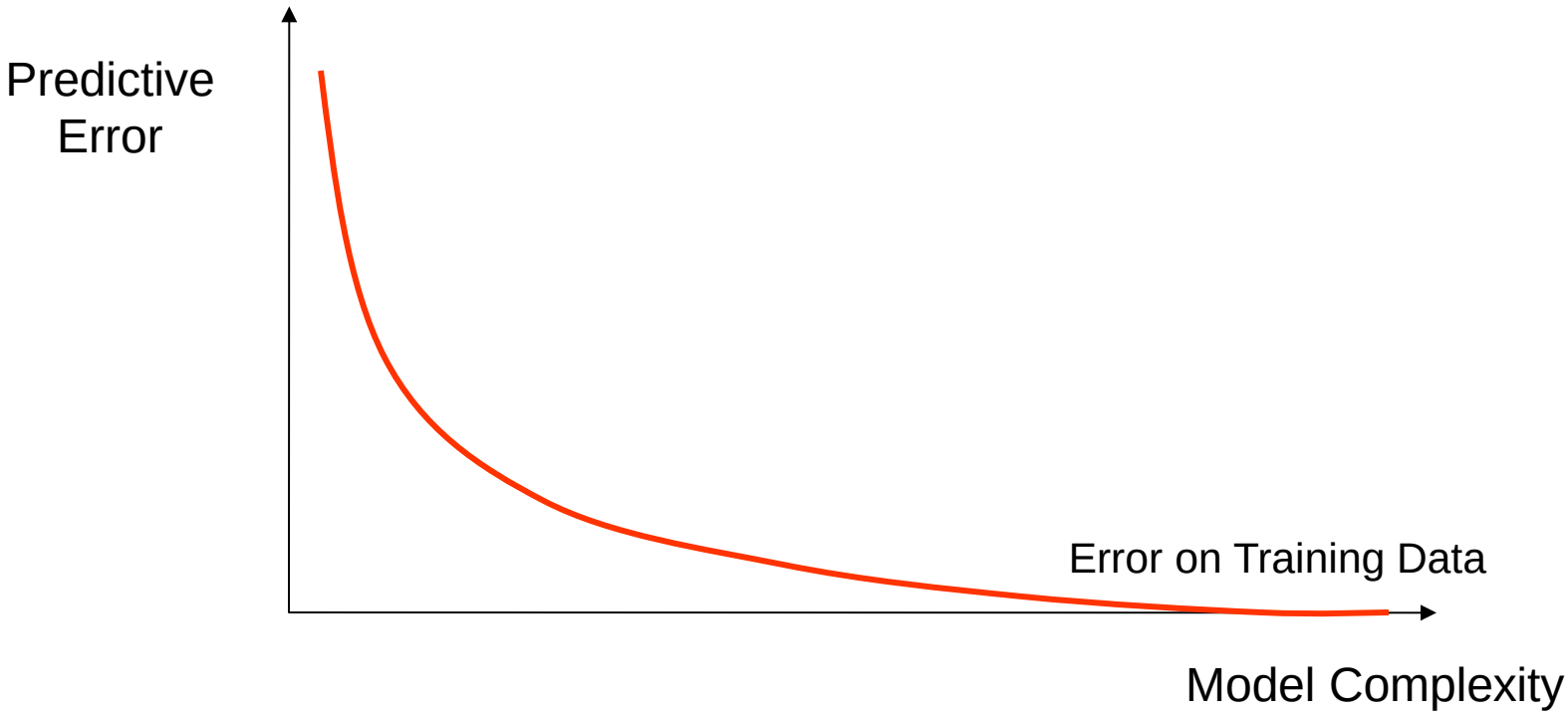
Example 2



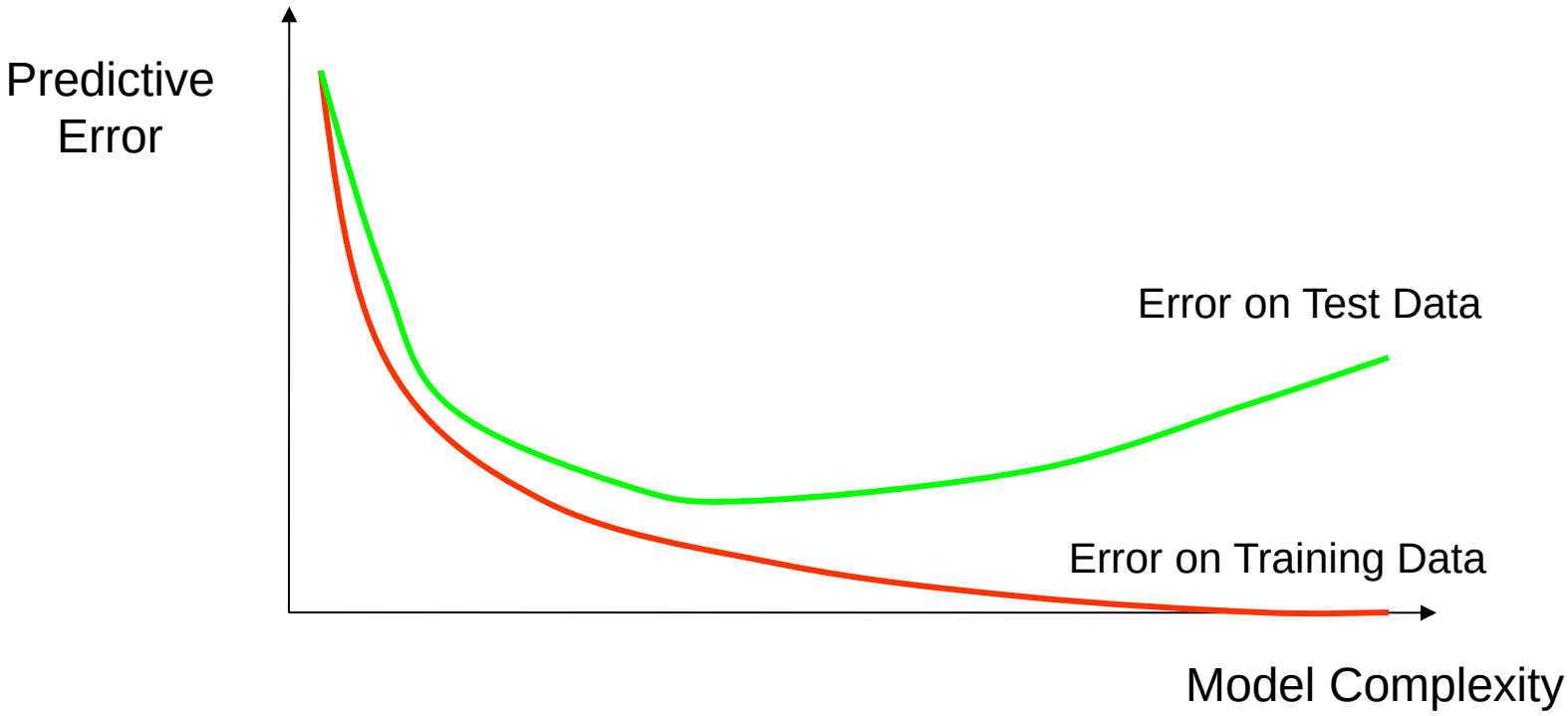
Example 2



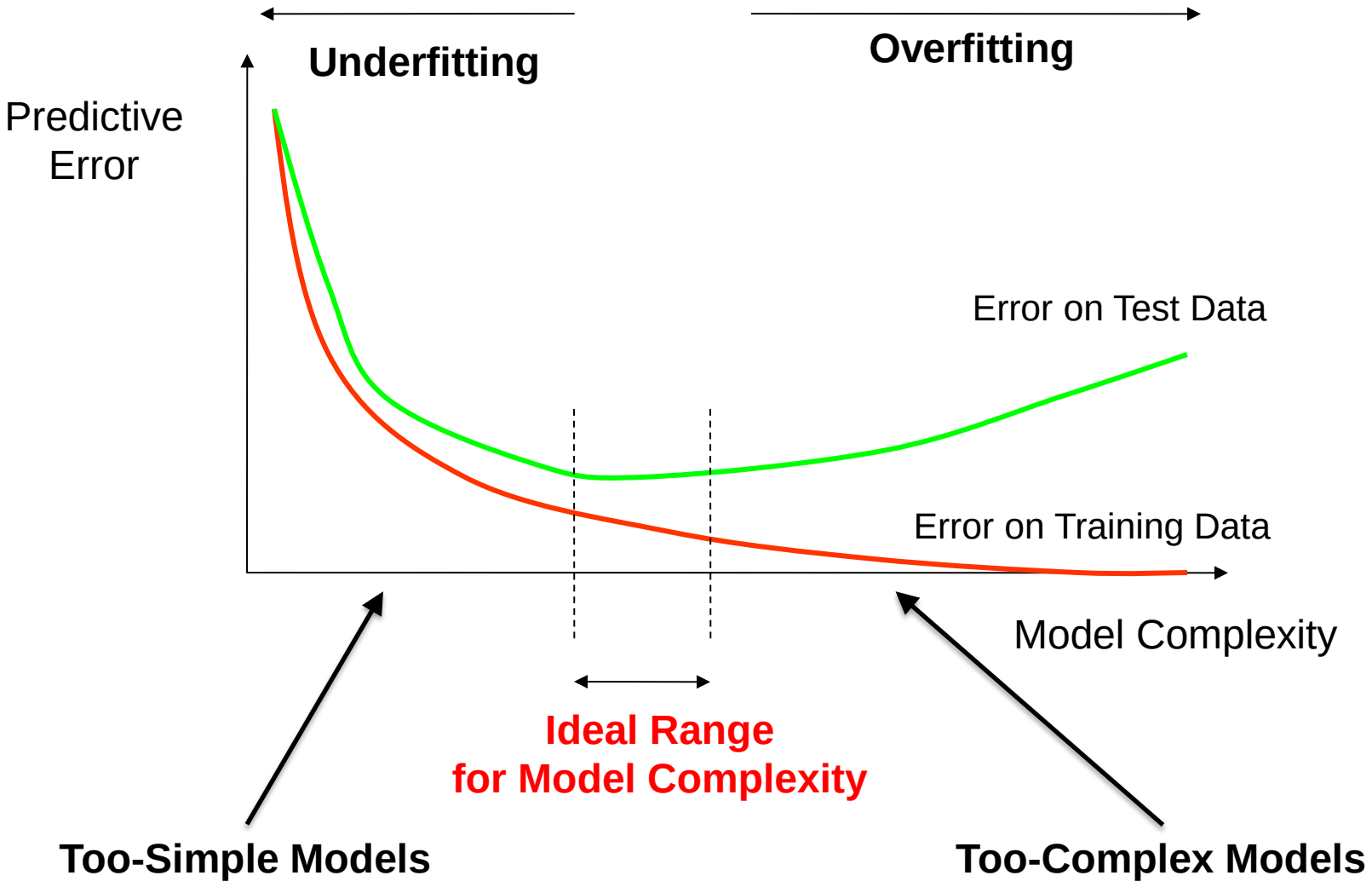
How Overfitting affects Prediction



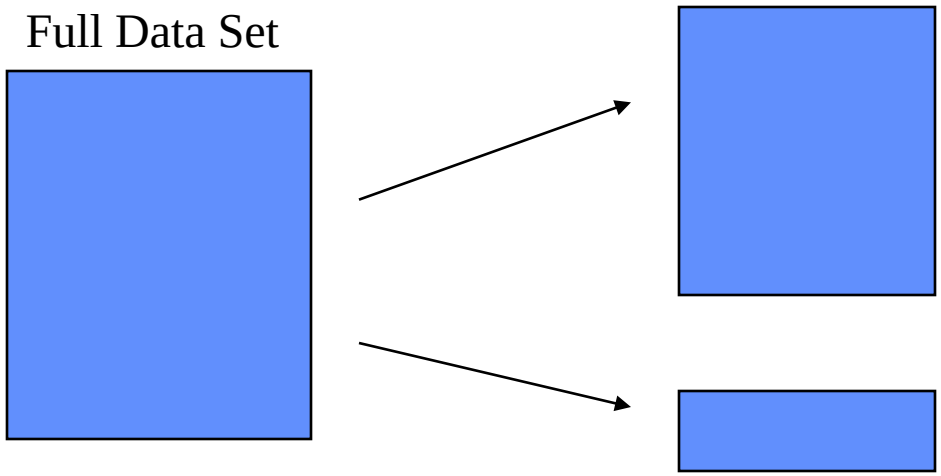
How Overfitting affects Prediction



How Overfitting affects Prediction



Training and Validation Data



Idea: train each model on the "training data"

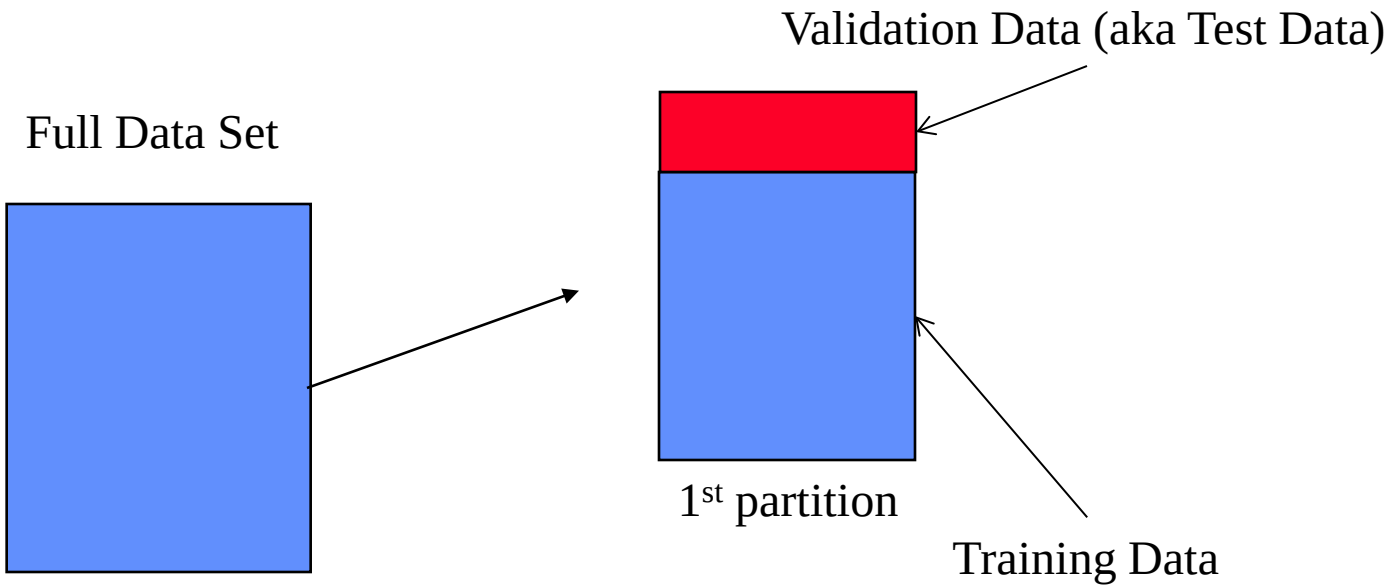
and then test each model's accuracy on the validation data

The k-fold Cross-Validation Method

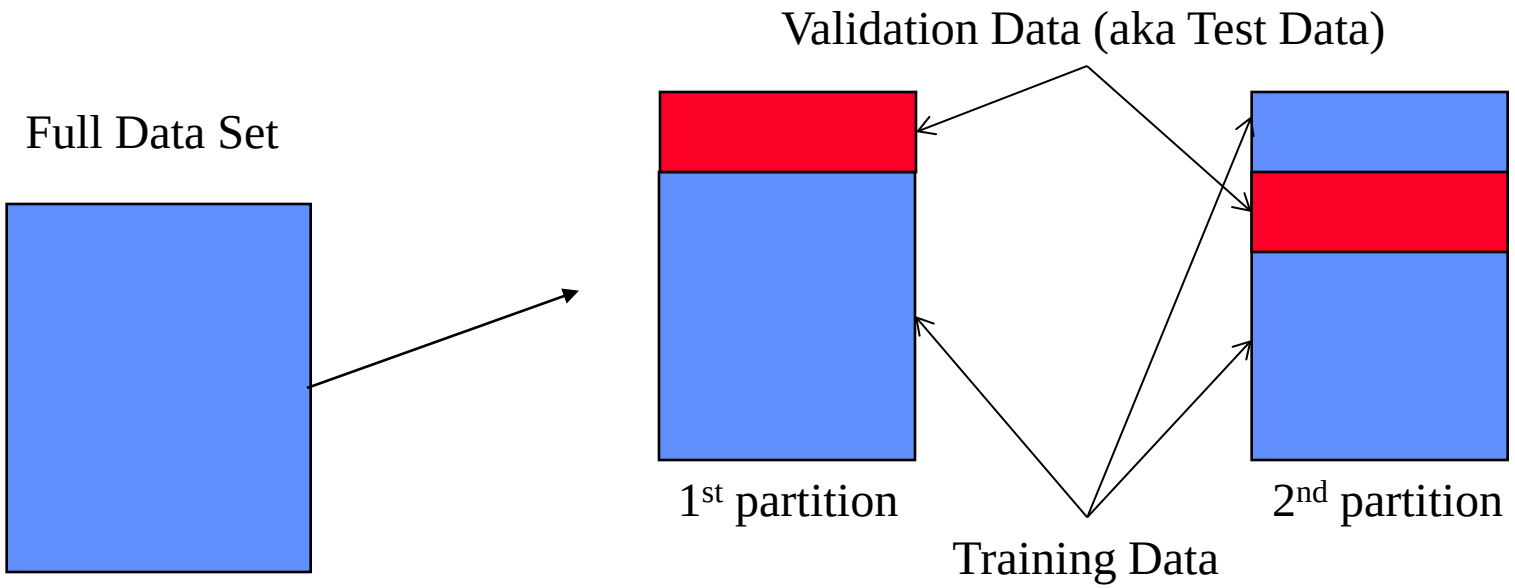
- Why just choose one particular 90/10 “split” of the data?
  - In principle we could do this multiple times

- “k-fold Cross-Validation” (e.g., k=10)
  - randomly partition our full data set into k disjoint subsets (each roughly of size n/k, n = total number of training data points)
    - for i = 1:10 (here k = 10)
      - train on 90% of data,
      - Acc(i) = accuracy on other 10%
    - end
    - Cross-Validation-Accuracy = $1/k \sum_i$ Acc(i)
  - choose the method with the highest cross-validation accuracy
  - common values for k are 5 and 10
  - Can also do “leave-one-out” where k = n

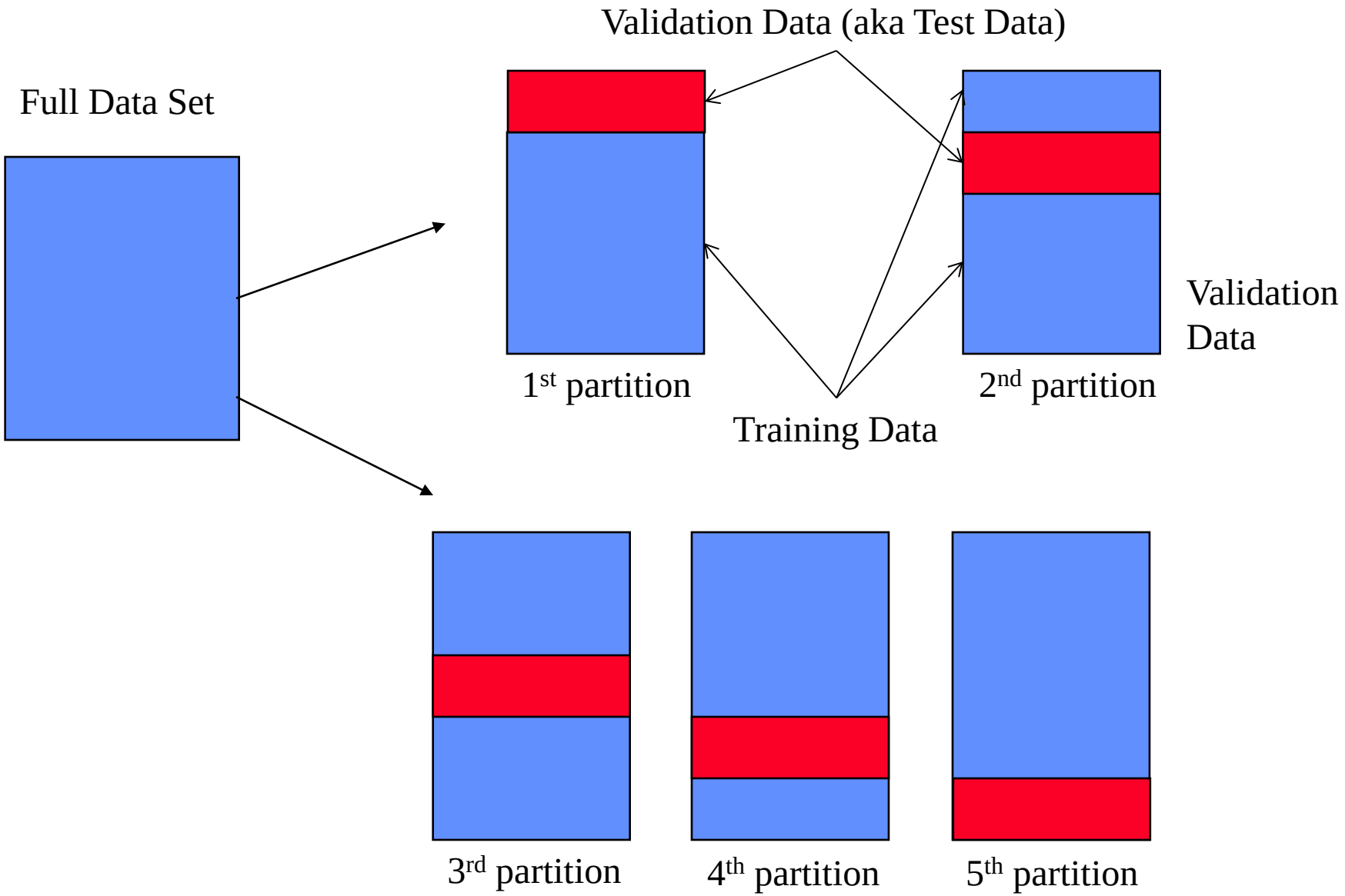
Disjoint Validation Data Sets



Disjoint Validation Data Sets



Disjoint Validation Data Sets



More on Cross-Validation

- Notes
  - cross-validation generates an approximate estimate of how well the learned model will do on “unseen” data
  - by averaging over different partitions it is more robust than just a single train/validate partition of the data
  - “k-fold” cross-validation is a generalization
    - partition data into disjoint validation subsets of size n/k
    - train, validate, and average over the v partitions
    - e.g., k=10 is commonly used
  - k-fold cross-validation is approximately k times computationally more expensive than just fitting a model to all of the data

You will be expected to know

- Understand Attributes, Error function, Classification, Regression, Hypothesis (Predictor function)
- What is Supervised Learning?
- Decision Tree Algorithm
- Entropy
- Information Gain
- Tradeoff between train and test with model complexity
- Cross validation

Summary

- Inductive learning
  - Error function, class of hypothesis/models {h}
  - Want to minimize E on our training data
  - Example: decision tree learning

- Generalization
  - Training data error is over-optimistic
  - We want to see performance on test data
  - Cross-validation is a useful practical approach

- Learning to recognize faces
  - Viola-Jones algorithm: state-of-the-art face detector, entirely learned from data, using boosting+decision-stumps