CS 446: Machine Learning

Dan Roth

University of Illinois, Urbana-Champaign

danr@illinois.edu

http://L2R.cs.uiuc.edu/~danr

3322 SC

CS446: Machine Learning

Participate, Ask Questions

- What do you need to know:
  - Theory of Computation
  - Probability Theory
  - Linear Algebra
  - Programming (Java; your favorite language; some Matlab)
- Homework 0 – on the web
  - No need to submit
- Who is the class for?
  - Future Machine Learning researchers/Advanced users

CS446: Policies

- Cheating

  No.

  We take it very seriously.

Info page

Note also the Schedule Page and our Notes

- Homework:
  - Collaboration is encouraged
  - But, you have to write your own solution/program.
  - (Please don't use old solutions)
- Late Policy:

  You have a credit of 4 days (4*24hours); That's it.
- Grading:
  - Possibly separate for grads/undergrads.
  - 5% Quizzes; 25% - homework; 30%-midterm; 40%-final;
  - Projects: 25% (4 hours)
- Questions?

CS446 Team

- **Dan Roth** (3323 Siebel)
  - Monday, 1-2 PM (or: appointment)
- **TAs**
  - Chase Duncan — Tues 12-1 — (3333 SC)
  - Subhro Roy — Wed 4-5 — (3333 SC)
  - Qiang Ning — Thur 3-4 — (3333 SC)
  - Hao Wu — Fri 1-2 — (3333 SC)
- **Discussion Sections**: (starting 3rd week) [times/locations not final]
  - **Tuesday:** 11 -12 [3405 SC] Subhro Roy [A-I]
  - **Wednesdays:** 5 -6 [3405 SC] Hao Wu [J-L]
  - **Thursdays:** 2 - 3 [3405 SC] Chase Duncan [M-S]
  - **Fridays:** 4 -5 [3405 SC] Qiang Ning [T-Z]

CS446 on the web

- Check our class website:
  - Schedule, slides, videos, policies
    - http://l2r.cs.uiuc.edu/~danr/Teaching/CS446-17/index.html
  - Sign up, participate in our Piazza forum:
    - Announcements and discussions
    - https://piazza.com/class#fall2017/cs446
  - Log on to Compass:
    - Submit assignments, get your grades
    - https://compass2g.illinois.edu
- Scribing the Class [Good writers; Latex; Paid Hourly]

What is Learning

- The Badges Game……
  - This is an example of the key learning protocol: supervised learning
- First question: Are you sure you got it?
  - Why?
- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm

Training data

+ Naoki Abe
- Myriam Abramson
+ David W. Aha
+ Kamal M. Ali
- Eric Allender
+ Dana Angluin
- Chidanand Apte
+ Minoru Asada
+ Lars Asker
+ Javed Aslam
+ Jose L. Balcazar
- Cristina Baroglio
+ Peter Bartlett
- Eric Baum
+ Welton Becket
- Shai Ben-David
+ George Berg
+ Neil Berkman
+ Malini Bhandaru
+ Bir Bhanu
+ Reinhard Blasig
- Avrim Blum
- Anselm Blumer
+ Justin Boyan
+ Carla E. Brodley
+ Nader Bshouty
- Wray Buntine
- Andrey Burago
+ Tom Bylander
+ Bill Byrne
- Claire Cardie
+ John Case
+ Jason Catlett
- Philip Chan
- Zhixiang Chen
- Chris Darken

The Badges game

+ Naoki Abe	- Eric Baum

- Conference attendees to the 1994 Machine Learning conference were given name badges labeled with + or −.

- What function was used to assign these labels?

Raw test data

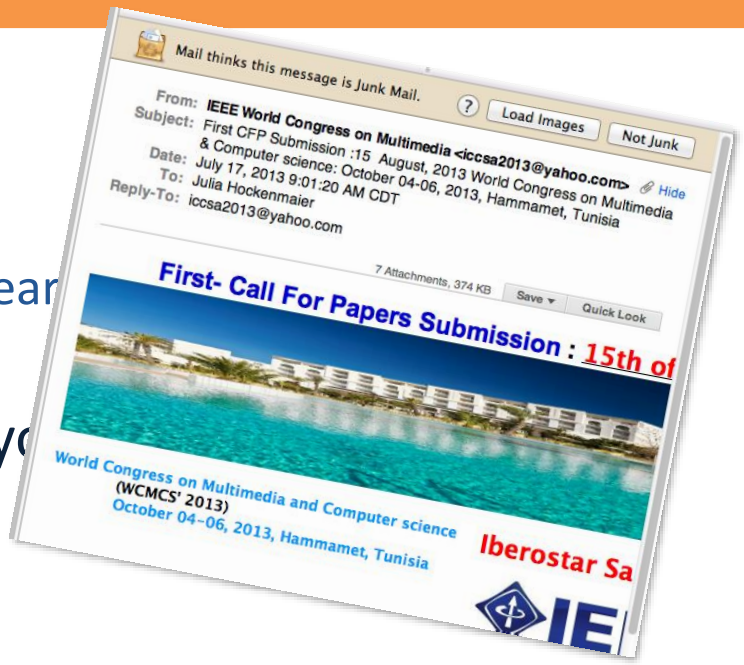
Gerald F. DeJong
Chris Drummond
Yolanda Gil
Attilio Giordana
Jiarong Hong
J. R. Quinlan
Priscilla Rasmussen
Dan Roth
Yoram Singer
Lyle H. Ungar

Labeled test data

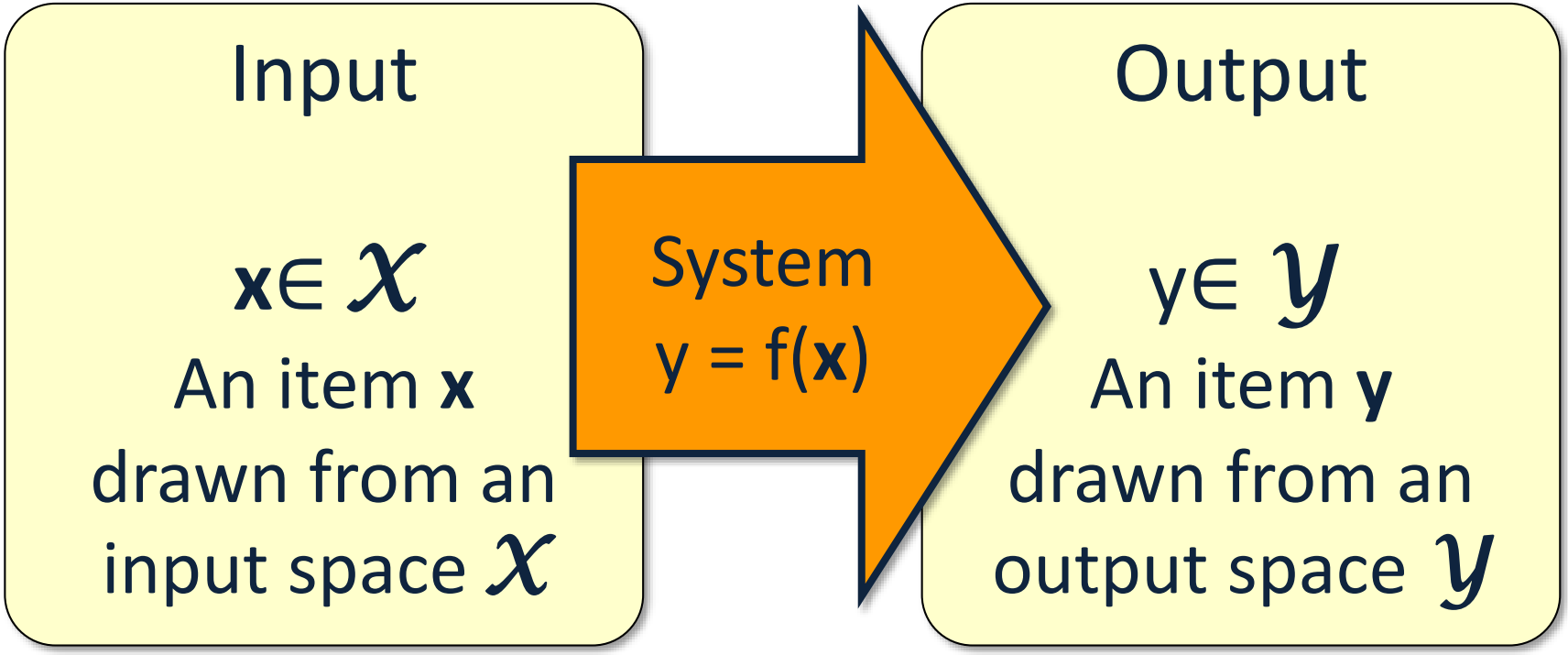
+ Gerald F. DeJong
- Chris Drummond
+ Yolanda Gil
- Attilio Giordana
+ Jiarong Hong
- J. R. Quinlan
- Priscilla Rasmussen
+ Dan Roth
+ Yoram Singer
- Lyle H. Ungar

What is Learning

- The Badges Game……
  - This is an example of the key lear
    learning
- First question: Are you sure yo
  - Why?
- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm

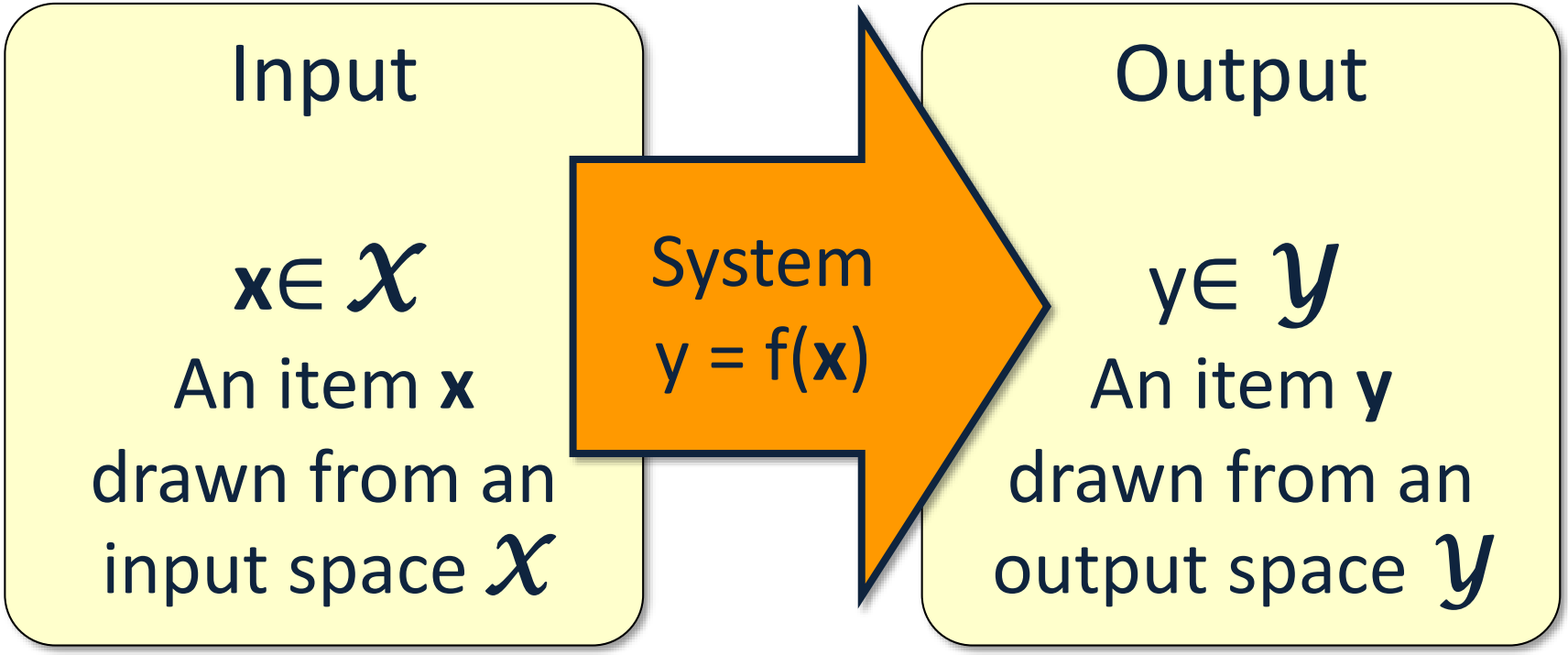


Supervised Learning



- We consider systems that apply a function f() to input items **x** and return an output **y** = f(**x**).

Supervised Learning

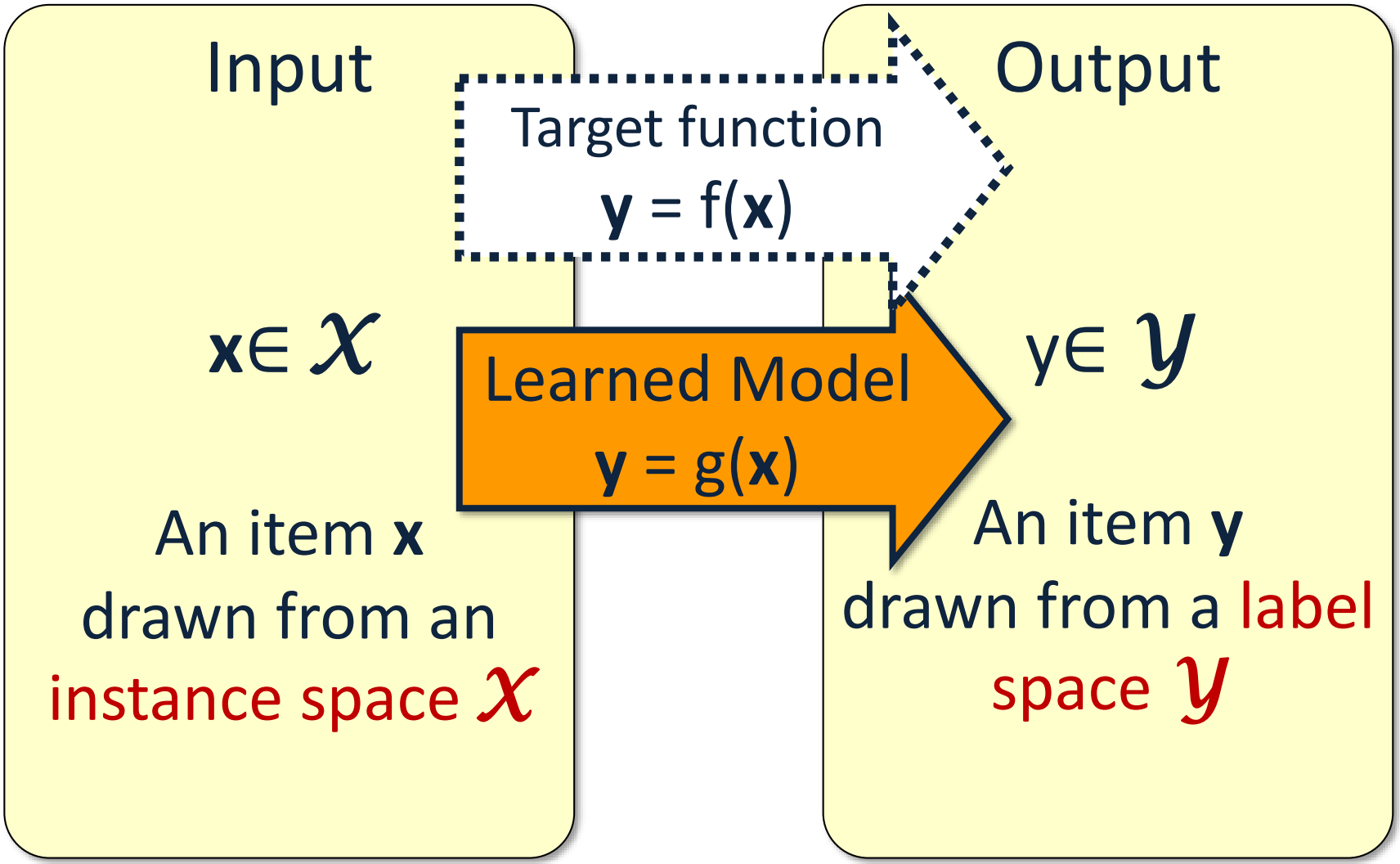


- In (supervised) machine learning, we deal with systems whose f(**x**) is learned from examples.

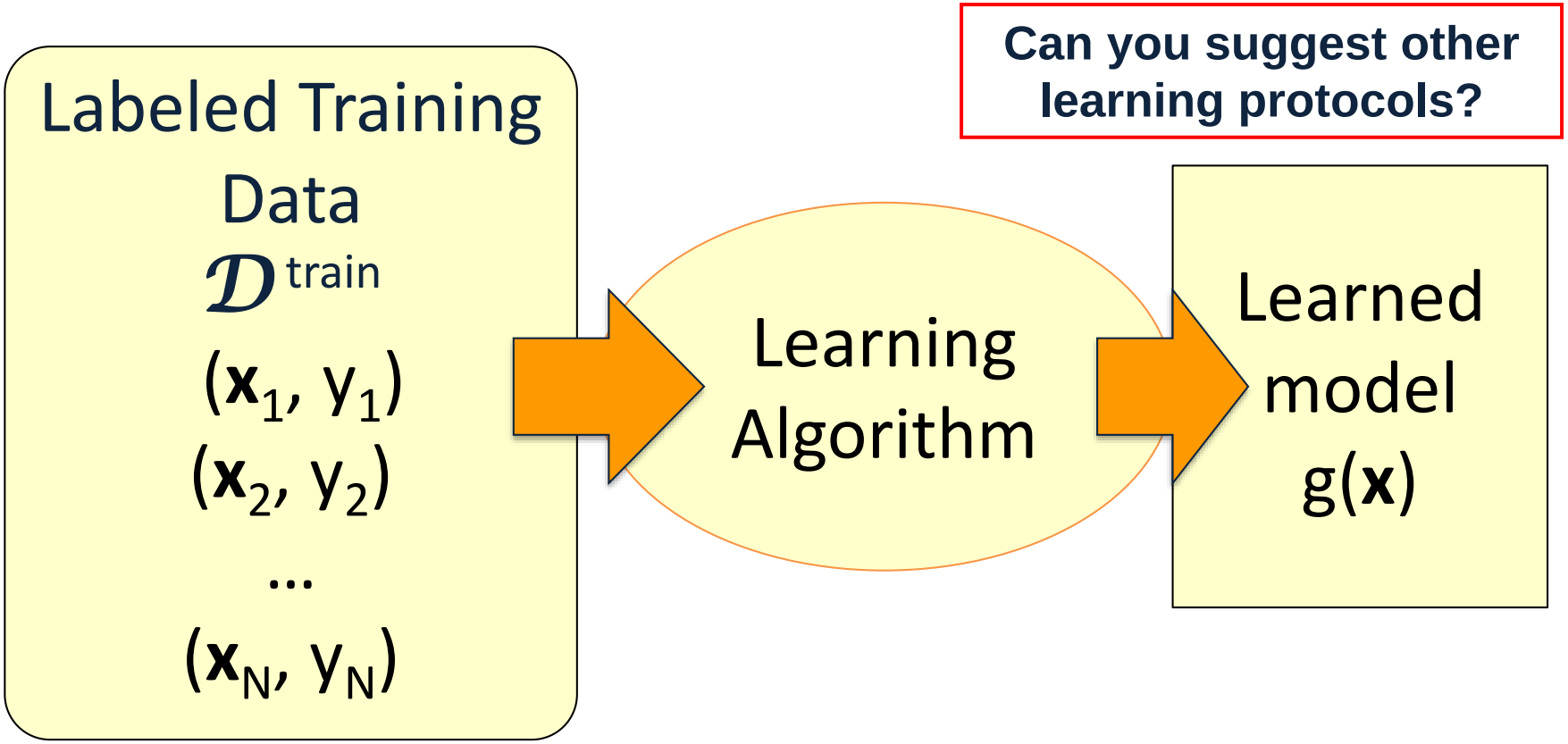
Why use learning?

- We typically use machine learning when the function f(**x**) we want the system to apply is unknown to us, and we cannot "think" about it. The function could actually be simple.

Supervised learning



Supervised learning: Training



- Give the learner examples in $\mathcal{D}^{\text{train}}$
- The learner returns a model $g(\mathbf{x})$

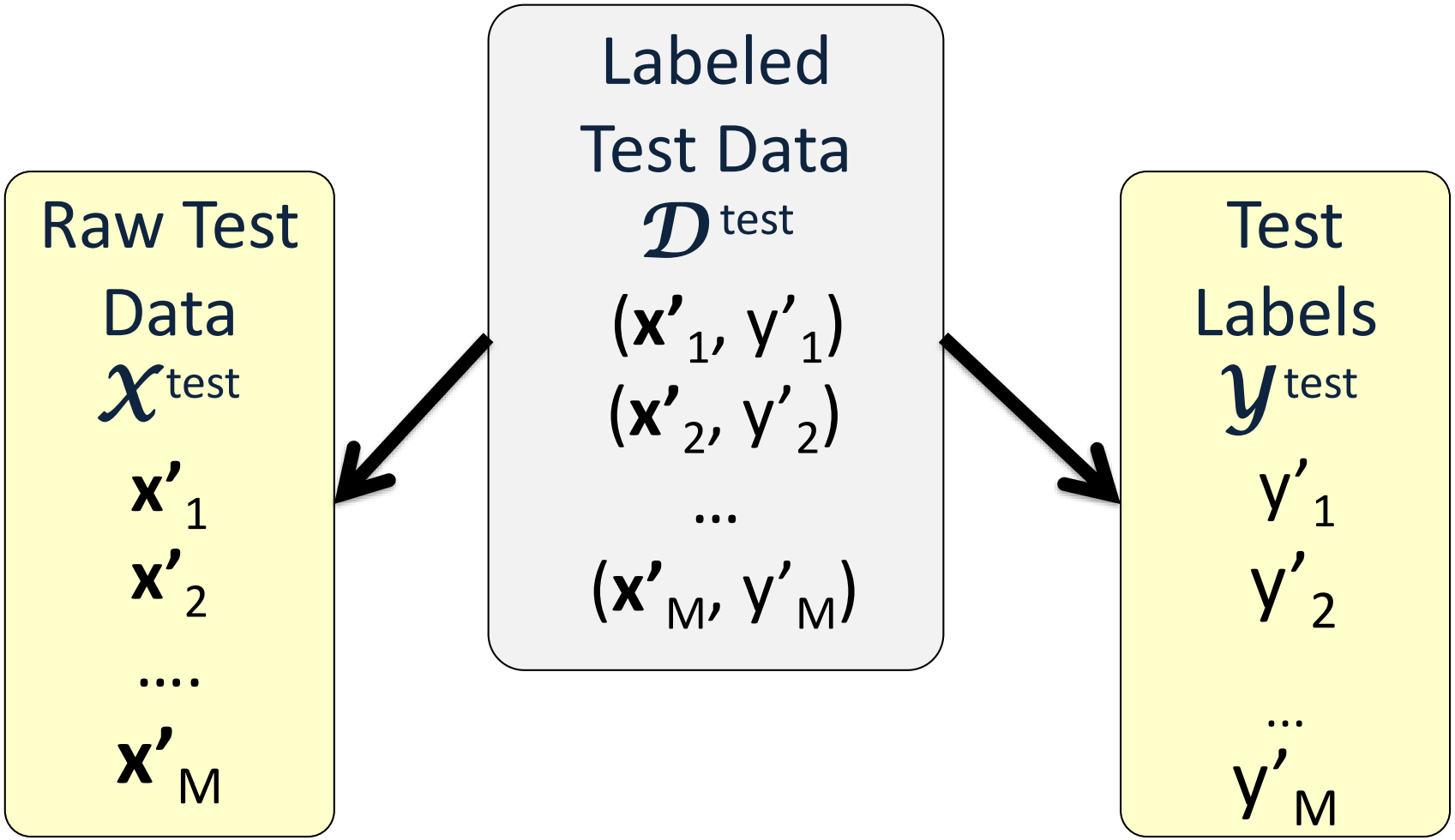
Supervised learning: Testing

Labeled
Test Data
$\mathcal{D}^{\text{test}}$

$(\mathbf{x'}_1, y'_1)$
$(\mathbf{x'}_2, y'_2)$
…
$(\mathbf{x'}_M, y'_M)$

- Reserve some labeled data for testing

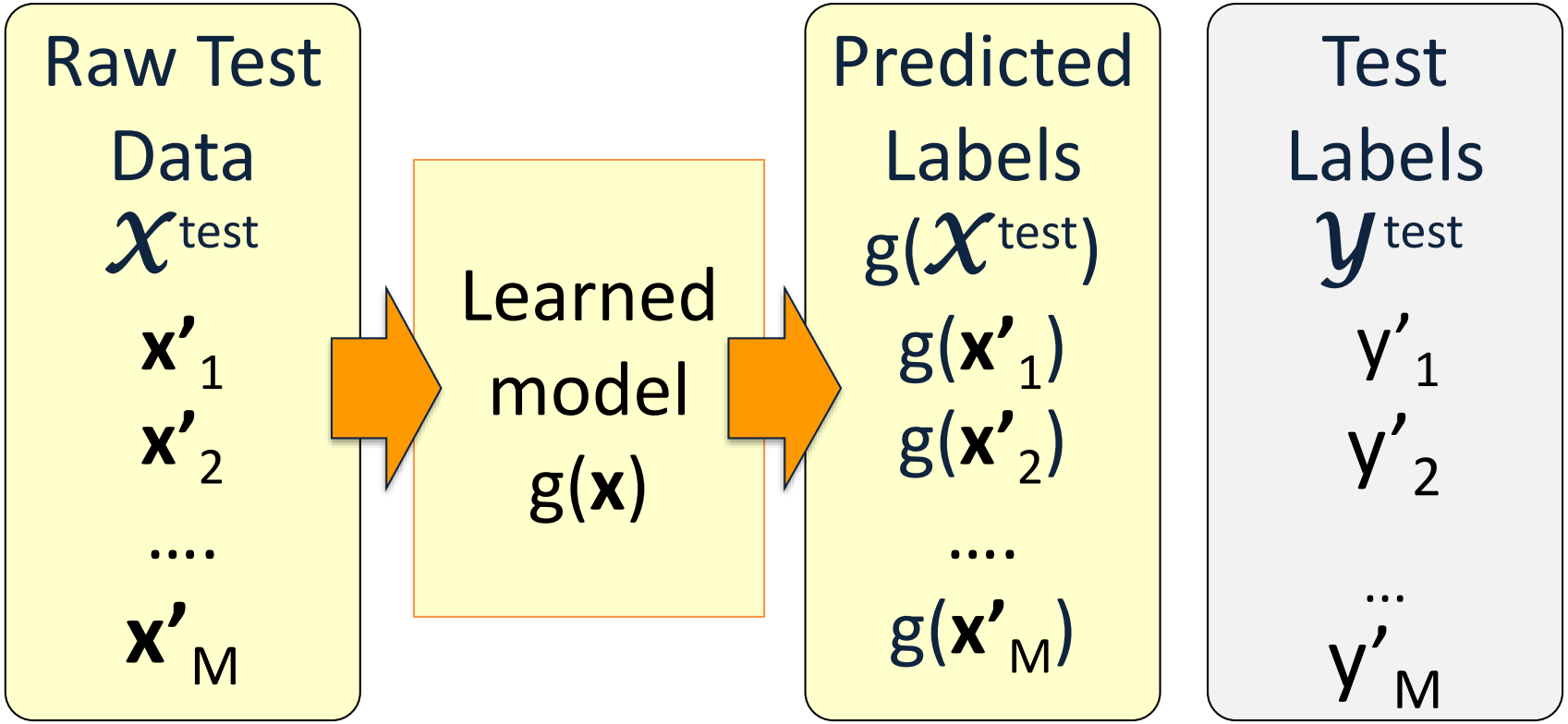
Supervised learning: Testing



Supervised learning: Testing

- Apply the model to the raw test data
- Evaluate by comparing predicted labels against the test labels

Can you use the test data otherwise?



What is Learning

- The Badges Game……
  - This is an example of the key learning protocol: supervised learning
- First question: Are you sure you got it?
  - Why?
- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm

Course Overview

- Introduction: Basic problems and questions
- A detailed example: Linear threshold units
  - Online Learning
- Two Basic Paradigms:
  - PAC (Risk Minimization)
  - Bayesian theory
- Learning Protocols:
  - Supervised; Unsupervised; Semi-supervised
- Algorithms
  - Decision Trees (C4.5)
  - [Rules and ILP (Ripper, Foil)]
  - Linear Threshold Units (Winnow; Perceptron; Boosting; SVMs; Kernels)
  - [Neural Networks (Backpropagation)]
  - Probabilistic Representations (naïve Bayes; Bayesian trees; Densities)
  - Unsupervised /Semi supervised: EM
- Clustering; Dimensionality Reduction

Supervised Learning

- Given: Examples (x,f(x)) of some unknown function f
- Find: A good approximation of f

- x provides some representation of the input
  - The process of mapping a domain element into a representation is called Feature Extraction. (Hard; ill-understood; important)
  - $x \in \{0,1\}^n$ or $x \in \Re^n$
- The target function (label)
  - $f(x) \in \{-1,+1\}$ Binary Classification
  - $f(x) \in \{1,2,3,..,k-1\}$ Multi-class classification
  - $f(x) \in \Re$ Regression

Supervised Learning : Examples

- Disease diagnosis
  - x: Properties of patient (symptoms, lab tests)
  - f : Disease (or maybe: recommended therapy)
- Part-of-Speech tagging
  - x: An English sentence (e.g., The can will rust)
  - f : The part of speech of a word in the sentence
- Face recognition
  - x: Bitmap picture of person's face
  - f : Name the person (or maybe: a property of)
- Automatic Steering
  - x: Bitmap picture of road surface in front of car
  - f : Degrees to turn the steering wheel

Many problems that do not seem like classification problems can be decomposed to classification problems. E.g, Semantic Role Labeling

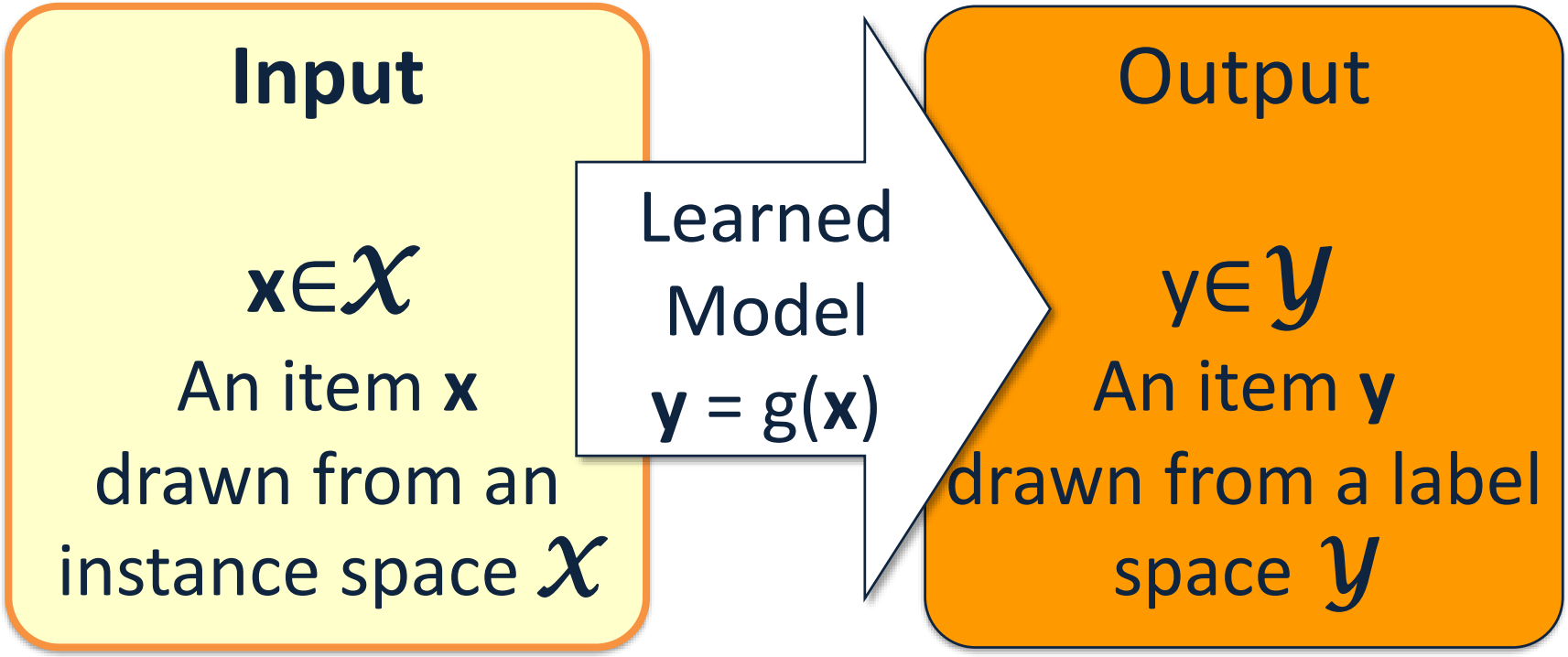
Key Issues in Machine Learning

- Modeling
  - How to formulate application problems as machine learning problems ? How to represent the data?
  - Learning Protocols (where is the data & labels coming from?)
- Representation
  - What functions should we learn (hypothesis spaces) ?
  - How to map raw input to an instance space?
  - Any rigorous way to find these? Any general approach?
- Algorithms
  - What are good algorithms?
  - How do we define success?
  - Generalization Vs. over fitting
  - The computational problem

Using supervised learning

- What is our instance space?
  - Gloss: What kind of features are we using?
- What is our label space?
  - Gloss: What kind of learning task are we dealing with?
- What is our hypothesis space?
  - Gloss: What kind of functions (models) are we learning?
- What learning algorithm do we use?
  - Gloss: How do we learn the model from the labeled data?
- What is our loss function/evaluation metric?
  - Gloss: How do we measure success? What drives learning?

1. The instance space $\mathcal{X}$



- Designing an appropriate instance space $\mathcal{X}$ is crucial for how well we can predict y.

1. The instance space $\mathcal{X}$

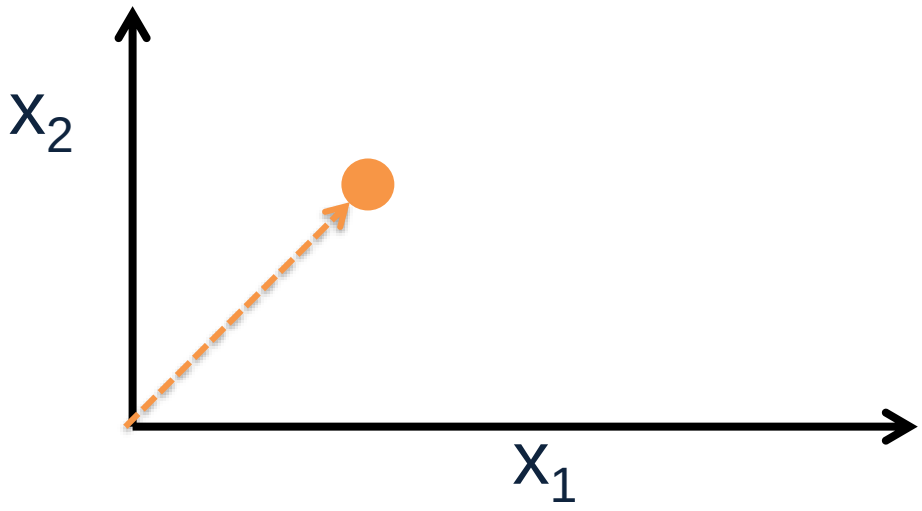
- When we apply machine learning to a task, we first need to *define* the instance space $\mathcal{X}$.
- Instances $\mathbf{x} \in \mathcal{X}$ are defined by features:
  - Boolean features:
    - Does this email contain the word 'money'?
  - Numerical features:
    - How often does 'money' occur in this email?
    - What is the width/height of this bounding box?
    - What is the length of the first name?

What's $\mathcal{X}$ for the Badges game?

- Possible features:
  - Gender/age/country of the person?
  - Length of their first or last name?
  - Does the name contain letter 'x'?
  - How many vowels does their name contain?
  - Is the n-th letter a vowel?

$\mathcal{X}$ as a vector space

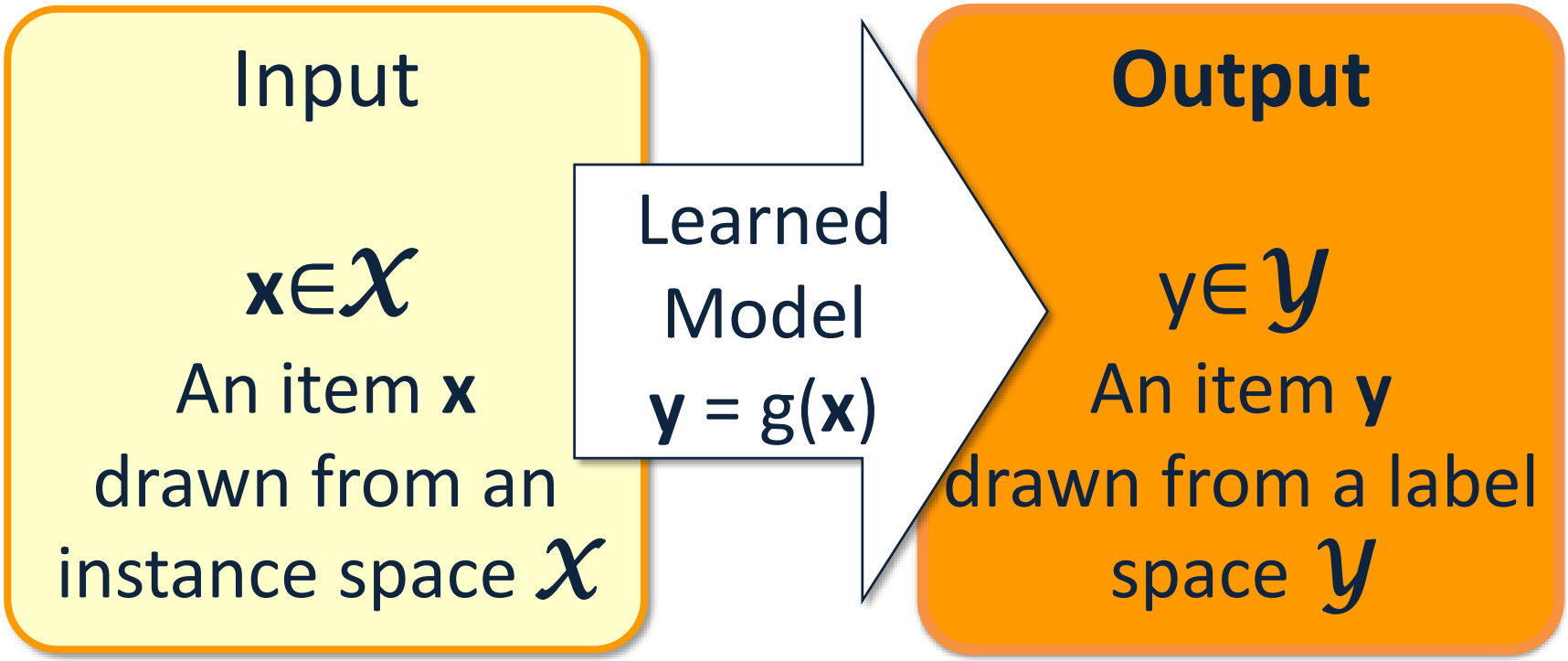
- $\mathcal{X}$ is an N-dimensional vector space (e.g. $\Re^N$)
  - Each dimension = one feature.
- Each **x** is a feature vector (hence the boldface **x**).
- Think of **x** = $[x_1 \ldots x_N]$ as a point in $\mathcal{X}$:



Good features are essential

- The choice of features is crucial for how well a task can be learned.
  - In many application areas (language, vision, etc.), a lot of work goes into designing suitable features.
  - This requires domain expertise.
- CS446 can't teach you what specific features to use for your task.
  - But we will touch on some general principles

2. The label space $\mathcal{Y}$



- The label space $\mathcal{Y}$ determines what *kind* of supervised learning task we are dealing with

Supervised learning tasks I

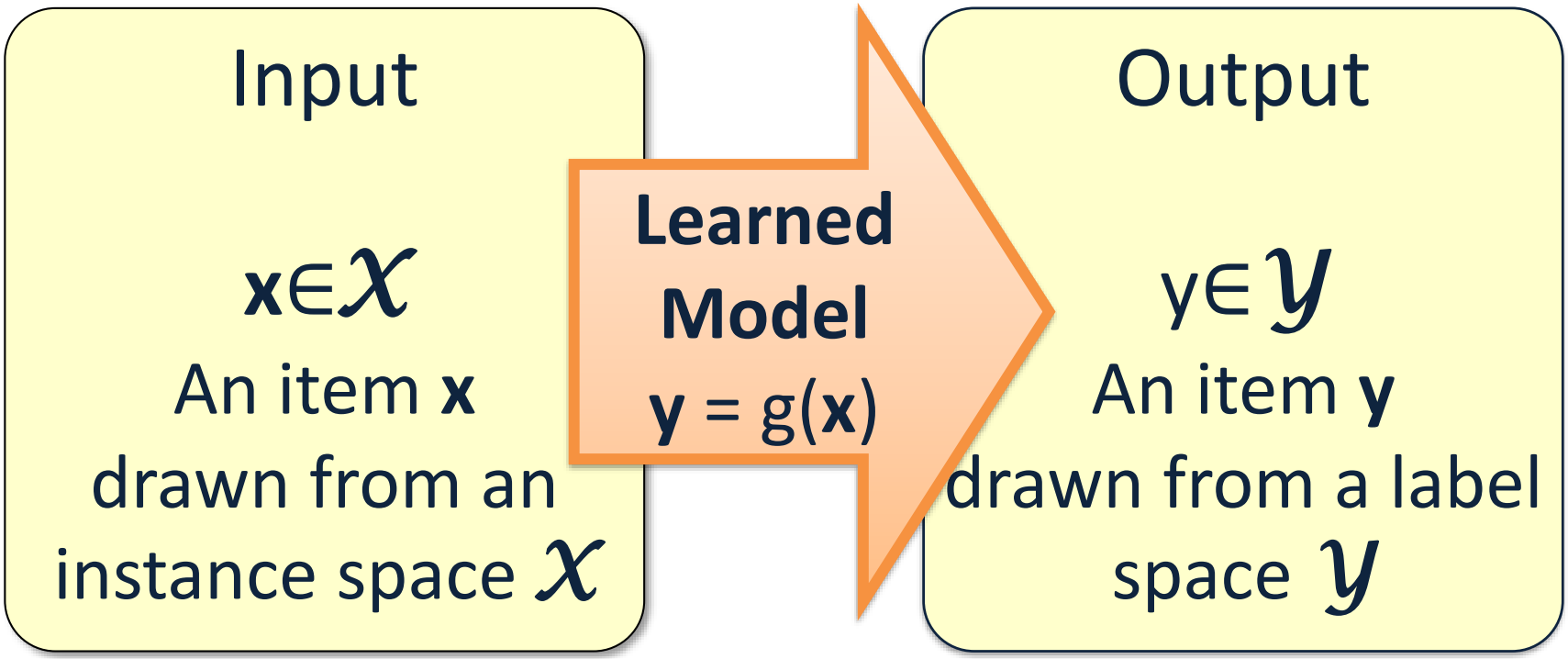
The focus of CS446. But…

- Output labels y∈Y are categorical:
  - Binary classification: Two possible labels
  - Multiclass classification: k possible labels
  - Output labels y∈Y are structured objects (sequences of labels, parse trees, etc.)
  - Structure learning

Supervised learning tasks II

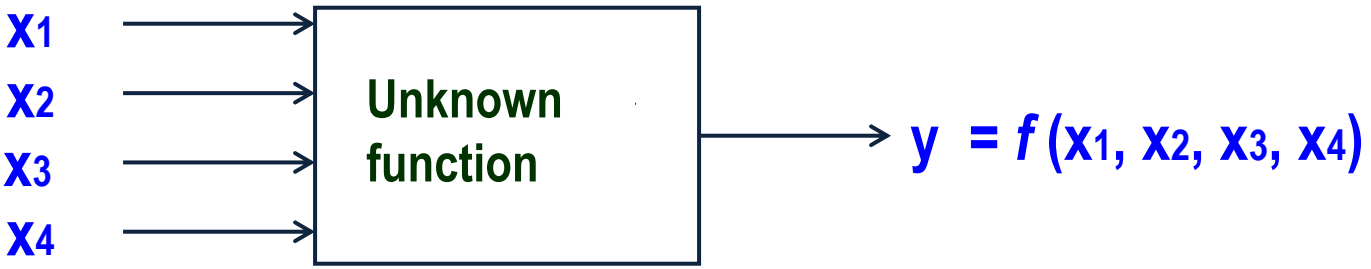
- Output labels $y \in Y$ are numerical:
  - Regression (linear/polynomial):
    - Labels are continuous-valued
    - Learn a linear/polynomial function f(x)
  - Ranking:
    - Labels are ordinal
    - Learn an ordering f(x1) > f(x2) over input

3. The model g(**x**)



- We need to choose what *kind* of model we want to learn

A Learning Problem



Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Can you learn this function?

What is it?

Hypothesis Space

Complete Ignorance:
There are 2^{16} = 65536 possible functions over four input features.

We can't figure out which one is correct until we've seen every possible input-output pair.

After observing seven exampl
have 2^9 possibilities for f

Is Learning Possible?

Example	x_1	x_2	x_3	x_4	y
	0	0	0	0	?
	0	0	0	1	?
	0	0	1	0	0
	0	0	1	1	1
	0	1	0	0	0
	0	1	0	1	0
	0	1	1	0	0

- There are $|\mathbf{Y}|^{|\mathbf{X}|}$ possible functions f(**x**) from the instance space **X** to the label space **Y.**
- Learners typically consider only a *subset* of the functions from **X** to **Y**, called the hypothesis space **H** . $\mathbf{H} \subseteq |\mathbf{Y}|^{|\mathbf{X}|}$

Hypothesis Space (2)

1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	1	0	0	1	1
5	0	1	1	0	0
6	1	1	0	0	0
7	0	1	0	1	0

Simple Rules: There are only 16 simple **conjunctive rules** of the form $y=x_i \wedge x_j \wedge x_k$

Rule	Counterexample	
$y=c$		
x_1	1100	0
x_2	0100	0
x_3	0110	0
x_4	0101	1
$x_1 \wedge x_2$	1100	0
$x_1 \wedge x_3$	0011	1
$x_1 \wedge x_4$	0011	1

Rule	Counterexample	
$x_2 \wedge x_3$	0011	1
$x_2 \wedge x_4$	0011	1
$x_3 \wedge x_4$	1001	1
$x_1 \wedge x_2 \wedge x_3$	0011	1
$x_1 \wedge x_2 \wedge x_4$	0011	1
$x_1 \wedge x_3 \wedge x_4$	0011	1
$x_2 \wedge x_3 \wedge x_4$	0011	1
$x_1 \wedge x_2 \wedge x_3 \wedge x_4$	0011	1

No simple rule explains the data. The same is true for **simple clauses**.

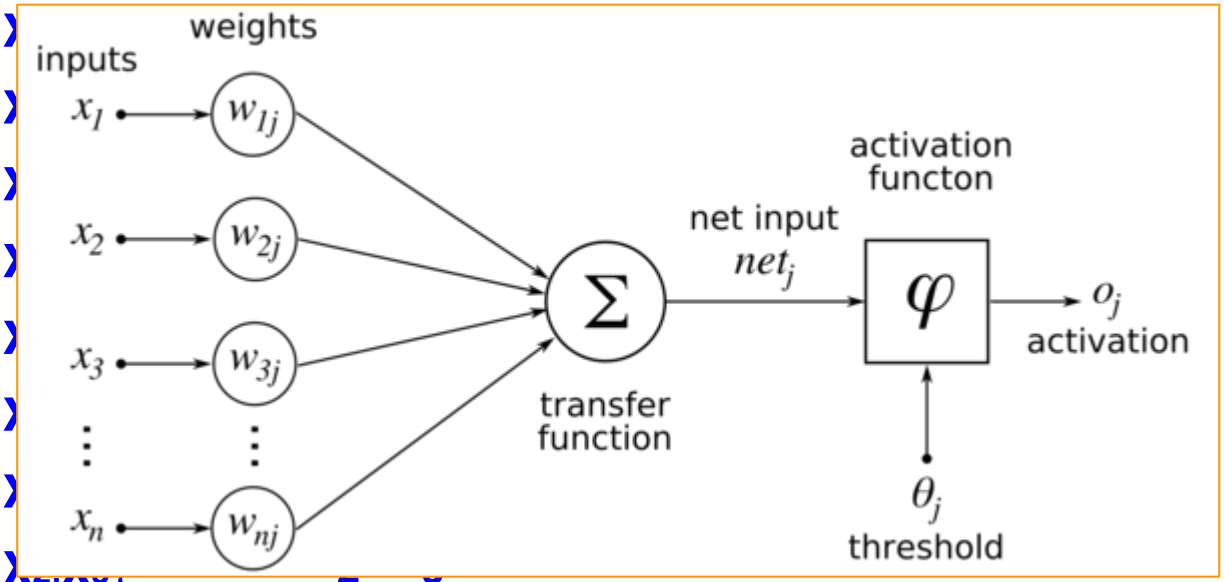
Hypothesis Space (3)

Don't worry, this function is actually a neural network…

Notation: 2 variables from the set on the left. **Value**: Index of the counterexample.

m-of-n rules: There are 32 possible rules of the form "y = 1 if and only if at least m of the following n variables are 1"

variables	1-of	2-of	3-of	4-of
{1,2}	2	3	-	-
{1,3}	4	4	-	-
{1,4}	1	3	3	-
{2,3}	2	3	3	-
{2,4}	1	* * *	3	-
{3,4}	1	5	3	-
{2,3,4}	1	5	3	3



Found a consistent hypothesis.

Views of Learning

- Learning is the removal of our remaining uncertainty:
  - Suppose we knew that the unknown function was an m-of-n Boolean function, then we could use the training data to infer which function it is.
- Learning requires guessing a good, small hypothesis class:
  - We can start with a very small class and enlarge it until it contains an hypothesis that fits the data.
- We could be wrong !
  - Our prior knowledge might be wrong:
    - y=x4 $\wedge$ one-of (x1, x3) is also consistent
  - Our guess of the hypothesis space could be wrong
- If this is the unknown function, then we will make errors when we are given new examples, and are asked to predict the value of the function

General strategies for Machine Learning

- Develop flexible hypothesis spaces:
  - Decision trees, neural networks, nested collections.
- Develop representation languages for restricted classes of functions:
  - Serve to limit the expressivity of the target models
  - E.g., Functional representation (n-of-m); Grammars; linear functions; stochastic models;
  - Get flexibility by augmenting the feature space

In either case:

- Develop algorithms for finding a hypothesis in our hypothesis space, that fits the data
- And hope that they will generalize well

CS446 Team

- **[updated 1/24/17]**
- **Dan Roth** (3323 Siebel)
  - Monday 1-2 (or: appointment)
- **TAs**
  - Chase Duncan Tues 12-1 (3333 SC)
  - Subhro Roy Wed 4-5 (3333 SC)
  - Qiang Ning Thur 3-4 (3333 SC)
  - Hao Wu Fri 1-2 (3333 SC)
- **Discussion Sections**: (starting next week)
  - **Monday:** 4 -5 [3405 SC] Chase Duncan[A-I]
  - **Wednesdays:** 5 -6 [3405 SC] Hao Wu [J-L]
  - **Thursdays:** 4 - 5 [3405 SC] Subhro Roy[M-S]
  - **Fridays:** 4 -5 [3405 SC] Qiang Ning [T-Z]

CS446 on the web

- Check our class website:
  - Schedule, slides, videos, policies
    - http://l2r.cs.uiuc.edu/~danr/Teaching/CS446-17/index.html
  - Sign up, participate in our Piazza forum:
    - Announcements and discussions
    - https://piazza.com/class#fall2017/cs446
  - Log on to Compass:
    - Submit assignments, get your grades
    - https://compass2g.illinois.edu
- Scribing the Class [Good writers; Latex; Paid Hourly]

Administration

- Registration [Ask NOW]
- Hw0: Solution will be released today
- Hw1 will be out tomorrow
  - Please start working on it as soon as possible;
    - Discussion sessions will start next week; come ready with questions
- Projects
  - Small (2-3) groups; your choice of a topic.
  - 25% of the grade → needs to be a substantial project
  - Extra credit for undergrads
- Quiz 1: Avg. score: 4.51/5
  - Only 165 of you attempted it (???)

Questions

An Example

Modeling

I don't know {whether, weather} to laugh or cry

This is the Modeling Step

How can we make this a learning problem?

What is the hypothesis space?

- We will look for a function

  F: Sentences → {whether, weather}
- We need to define the domain of this function better.

- **An option**: For each word w in English define a Boolean feature x_w :

  $[x_w = 1]$ iff w is in the sentence
- This maps a sentence to a point in $\{0,1\}^{50,000}$
- In this space: some points are whether points

  some are weather points

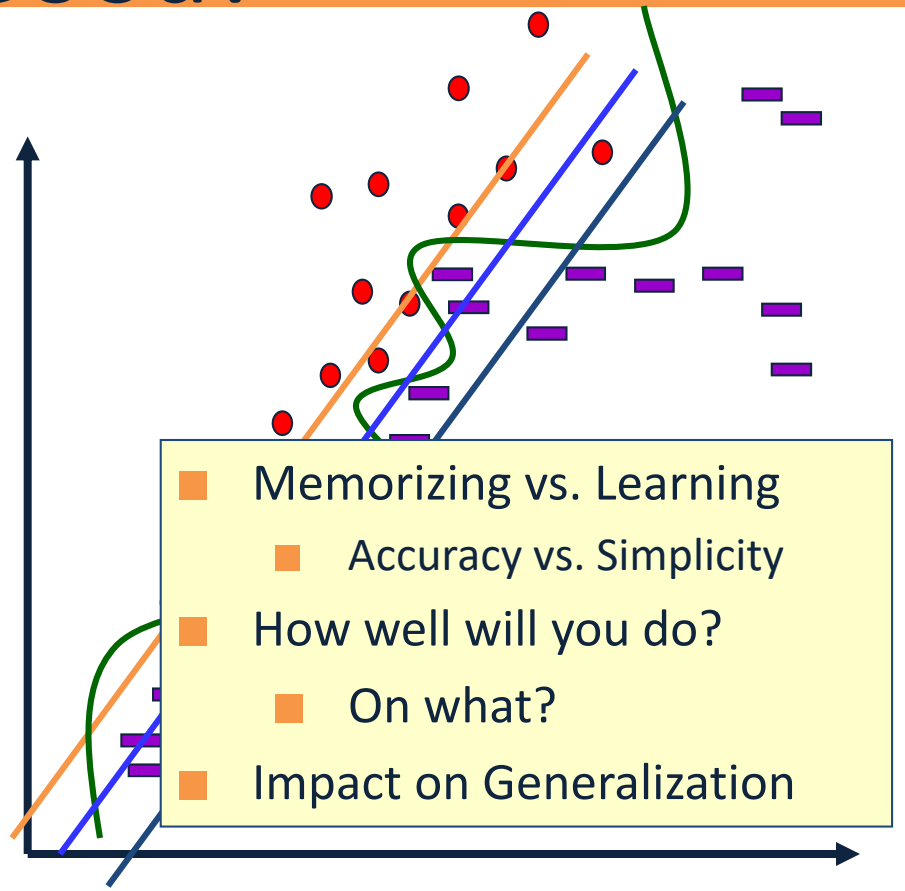
Learning Protocol?

Supervised? Unsupervised?

Representation Step: What's Good?

Representation

- Learning problem:
  - Find a function that
    - best separates the data
- What function?
- What's best?
- (How to find it?)



- Memorizing vs. Learning
  - Accuracy vs. Simplicity
- How well will you do?
  - On what?
- Impact on Generalization

Linear = linear in the feature space

x= data representation; w = the classifier

$y = \text{sgn}\{w^T x\}$

- A possibility: Define the learning problem to be:
  A (linear) function that best separates the data

Expressivity

Representation

$$f(x) = \text{sgn}\{x \cdot w - \theta\} = \text{sgn}\{\sum_{i=1}^{n} w_i x_i - \theta\}$$

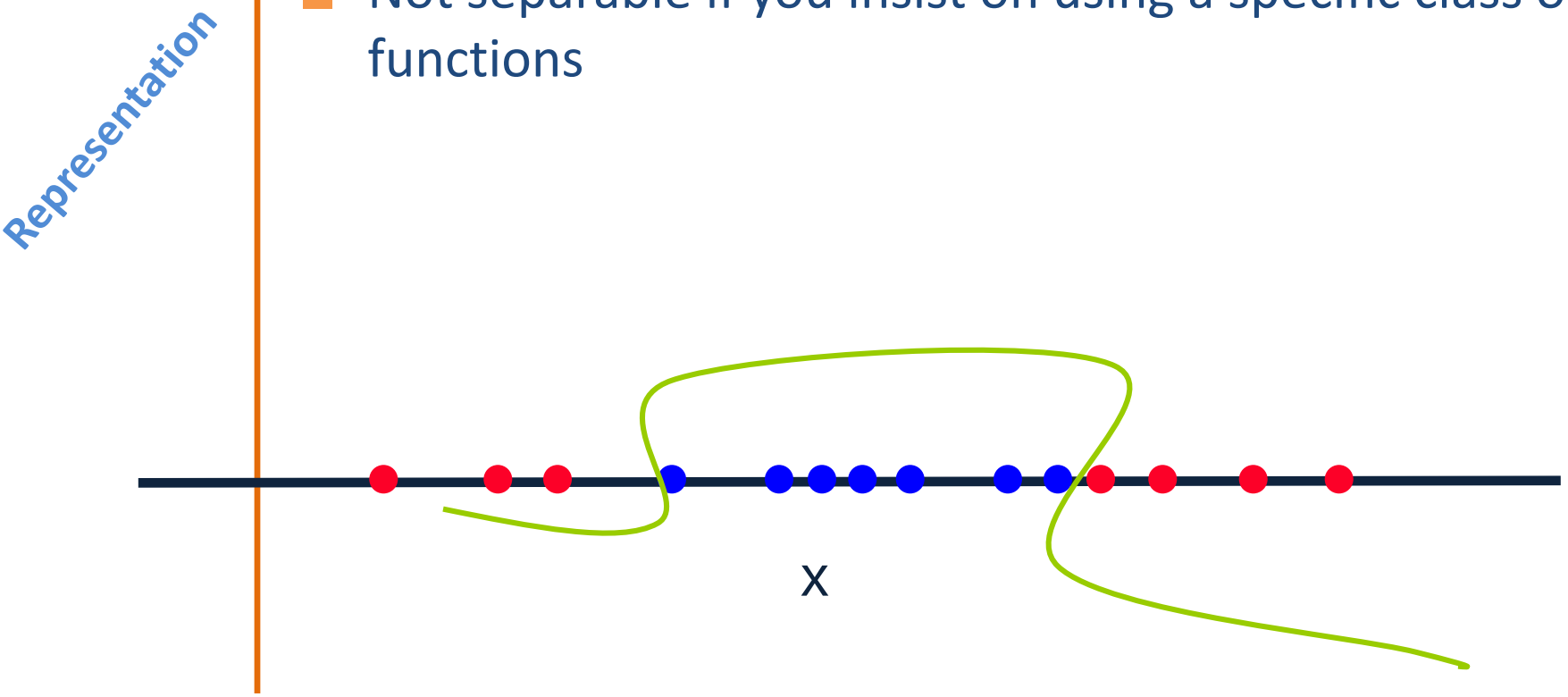
- Many functions are Linear

  Probabilistic Classifiers as well

  - Conjunctions:
    - $y = x_1 \wedge x_3 \wedge x_5$
    - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 3\}$; $w = (1, 0, 1, 0, 1)$ $\theta=3$
  - At least m of n:
    - y = at least 2 of $\{x_1, x_3, x_5\}$
    - $y = \text{sgn}\{1 \cdot x_1 + 1 \cdot x_3 + 1 \cdot x_5 - 2\}$; $w = (1, 0, 1, 0, 1)$ $\theta=2$
- Many functions are not
  - Xor: $y = x_1 \wedge x_2 \vee \neg x_1 \wedge \neg x_2$
  - Non trivial DNF: $y = x_1 \wedge x_2 \vee x_3 \wedge x_4$
- But can be made linear

Functions Can be Made Linear

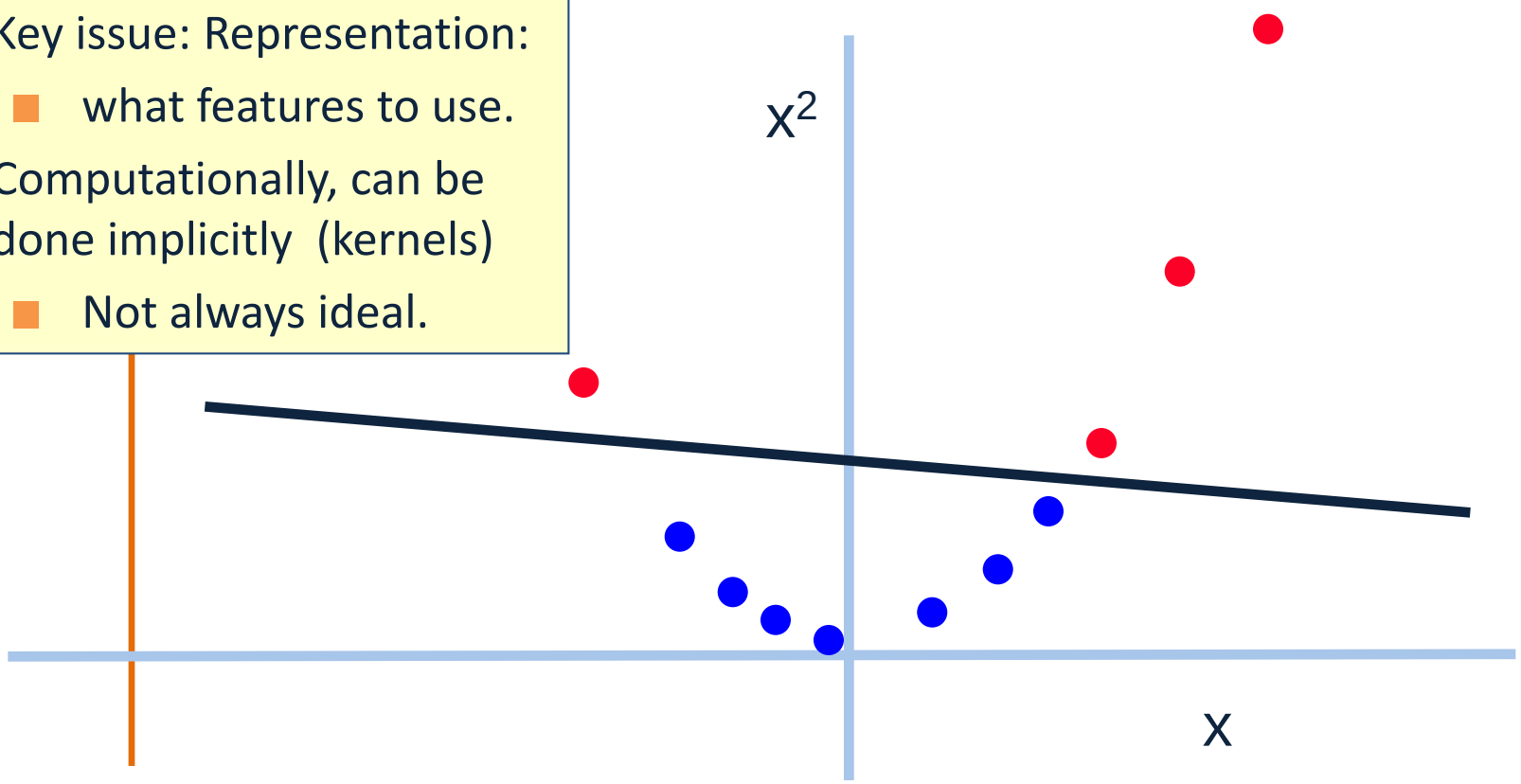
- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions



Blown Up Feature Space

- Data are separable in <x, x^2> space

- Key issue: Representation:
  - what features to use.
- Computationally, can be done implicitly (kernels)
  - Not always ideal.



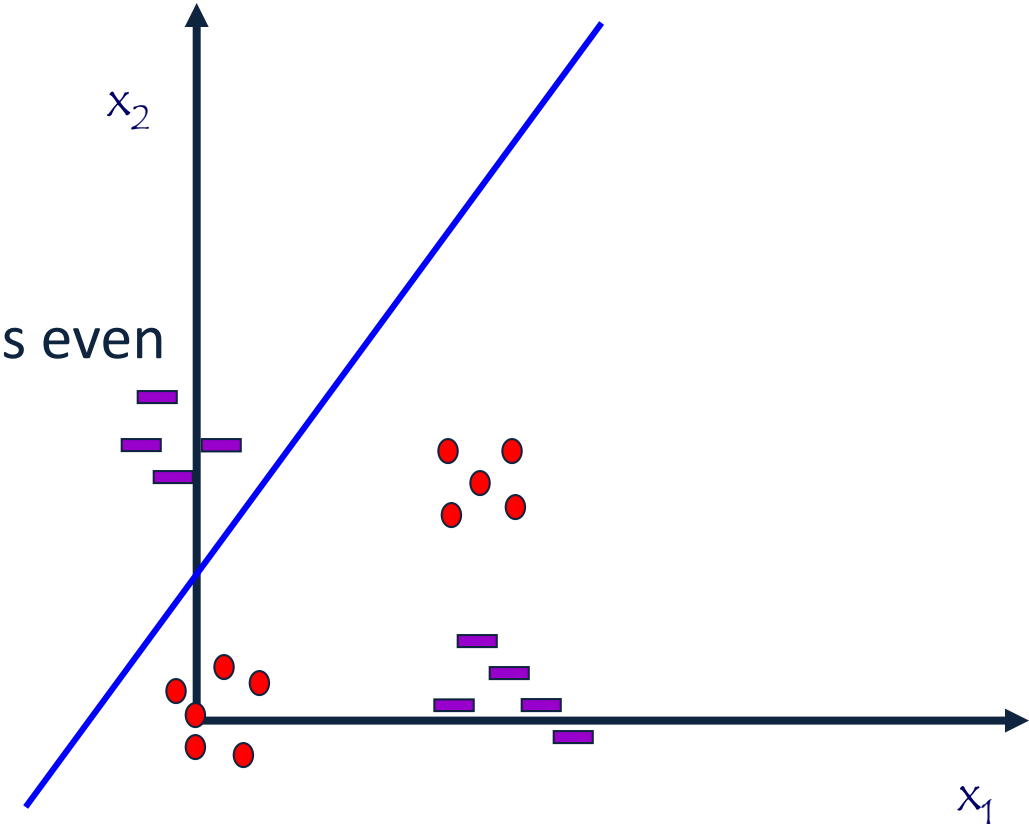
Exclusive-OR (XOR)

Representation

- $(x_1 \wedge x_{2)} \vee (\neg\{x_1\} \wedge \neg\{x_2\})$
- In general: a parity function.

- $x_i \in \{0,1\}$
- $f(x_1, x_2, \ldots, x_n) = 1$ iff $\sum x_i$ is even

This function is not linearly separable.



Functions Can be Made Linear

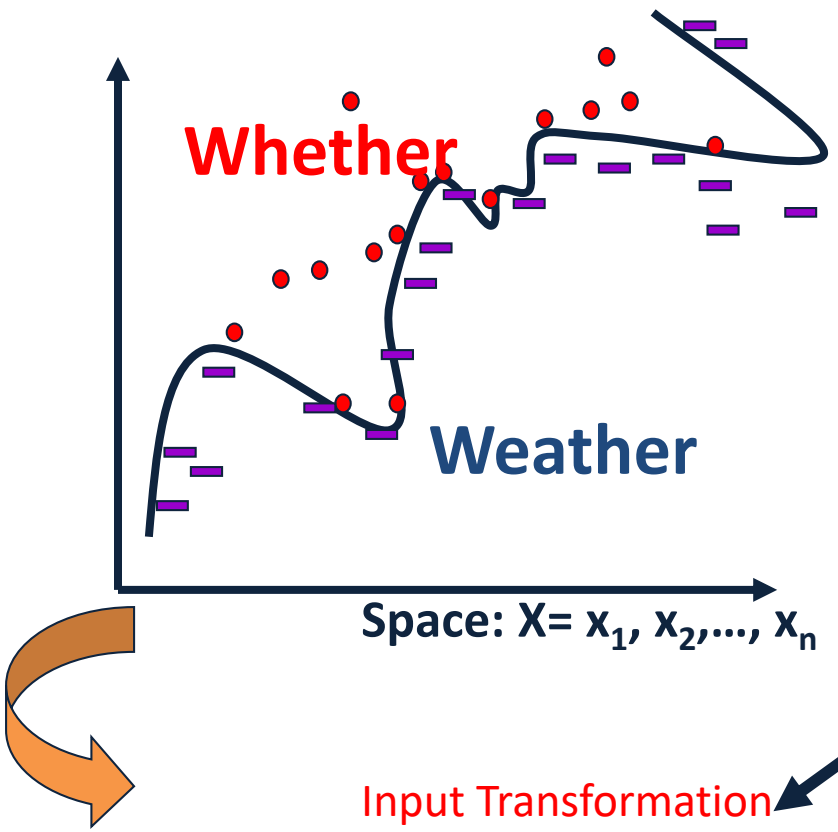
Discrete Case

A real Weather/Whether example

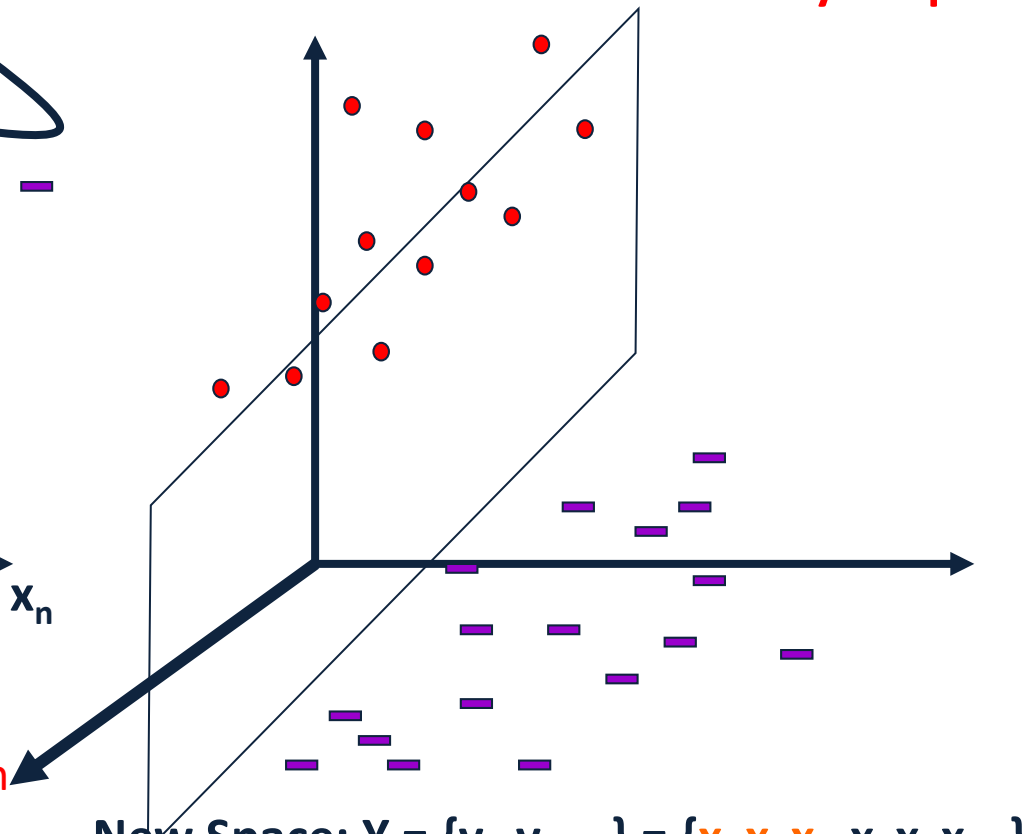
$x_1 x_2 x_4 \vee x_2 x_4 x_5 \vee x_1 x_3 x_7$

$y_3 \vee y_4 \vee y_7$

New discriminator is functionally simpler



Input Transformation

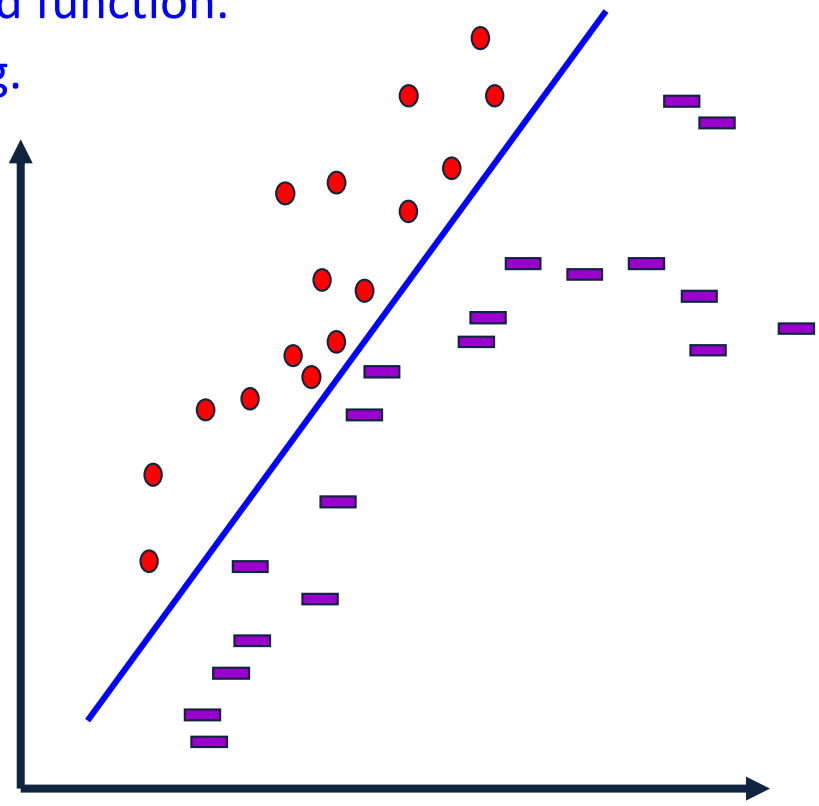


New Space: Y = {y_1,y_2,…} = {x_i,x_i x_j, x_i x_j $x_{j,\ldots}$}

Third Step: How to Learn?

Algorithms

- A possibility: Local search
  - Start with a linear threshold function.
  - See how well you are doing.
  - Correct
  - Repeat until you converge.
- There are other ways that do not search directly in the hypotheses space
  - Directly compute the hypothesis



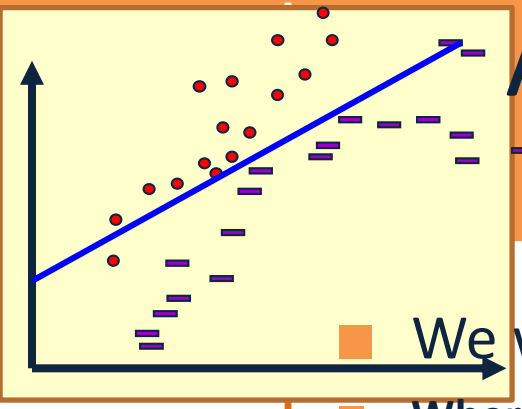
A General Framework for Learning

Algorithms

- Goal: predict an unobserved output value $y \in Y$ based on an observed input vector $x \in X$
- Estimate a functional relationship $y \sim f(x)$ from a set $\{(x,y)_i\}_{i=1,n}$
- Most relevant - Classification: $y \in \{0,1\}$ (or $y \in \{1,2,\ldots k\}$)
  - (But, within the same framework can also talk about Regression, $y \in \Re$)

Simple **loss function**: # of mistakes
[…] is a indicator function

- What do we want f(x) to satisfy?
  - We want to minimize the **Risk**: $L(f()) = E_{X,Y}([f(x) \neq y])$
  - Where: $E_{X,Y}$ denotes the expectation with respect to the true distribution.



A General Framework for Learning (II)

Algorithms

- We want to minimize the Loss: $L(f()) = E_{X,Y}([f(X) \neq Y])$
- **Where: $E_{X,Y}$ denotes the expectation with respect to the true distribution.**

Side note: If the distribution over $X \times Y$ is known, predict: $\mathbf{y = argmax_y P(y|x)}$
This is the best possible (the optimal Bayes' error).

- We cannot minimize this loss
- Instead, we try to minimize the empirical classification error.
- For a set of training examples $\{(x_i, y_i)\}_{i=1,n}$
- Try to minimize: $L'(f()) = 1/n \sum_i [f(x_i) \neq y_i]$
  - (Issue I: why/when is this good enough? Not now)
- This minimization problem is typically NP hard.
- **To alleviate this computational problem, minimize a new function – a convex upper bound of the classification error function**

$I(f(x),y) = [f(x) \neq y] = \{1$ when $f(x) \neq y$; 0 otherwise$\}$

Algorithmic View of Learning: an Optimization Problem

Algorithms

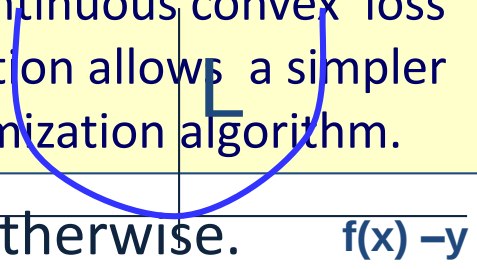
- A **Loss Function** L(f(x),y) measures the penalty incurred by a classifier f on example (x,y).
- There are many different loss functions one could define:
  - Misclassification Error:

    $L(f(x),y) = 0$ if $f(x) = y$; 1 otherwise
  - Squared Loss:

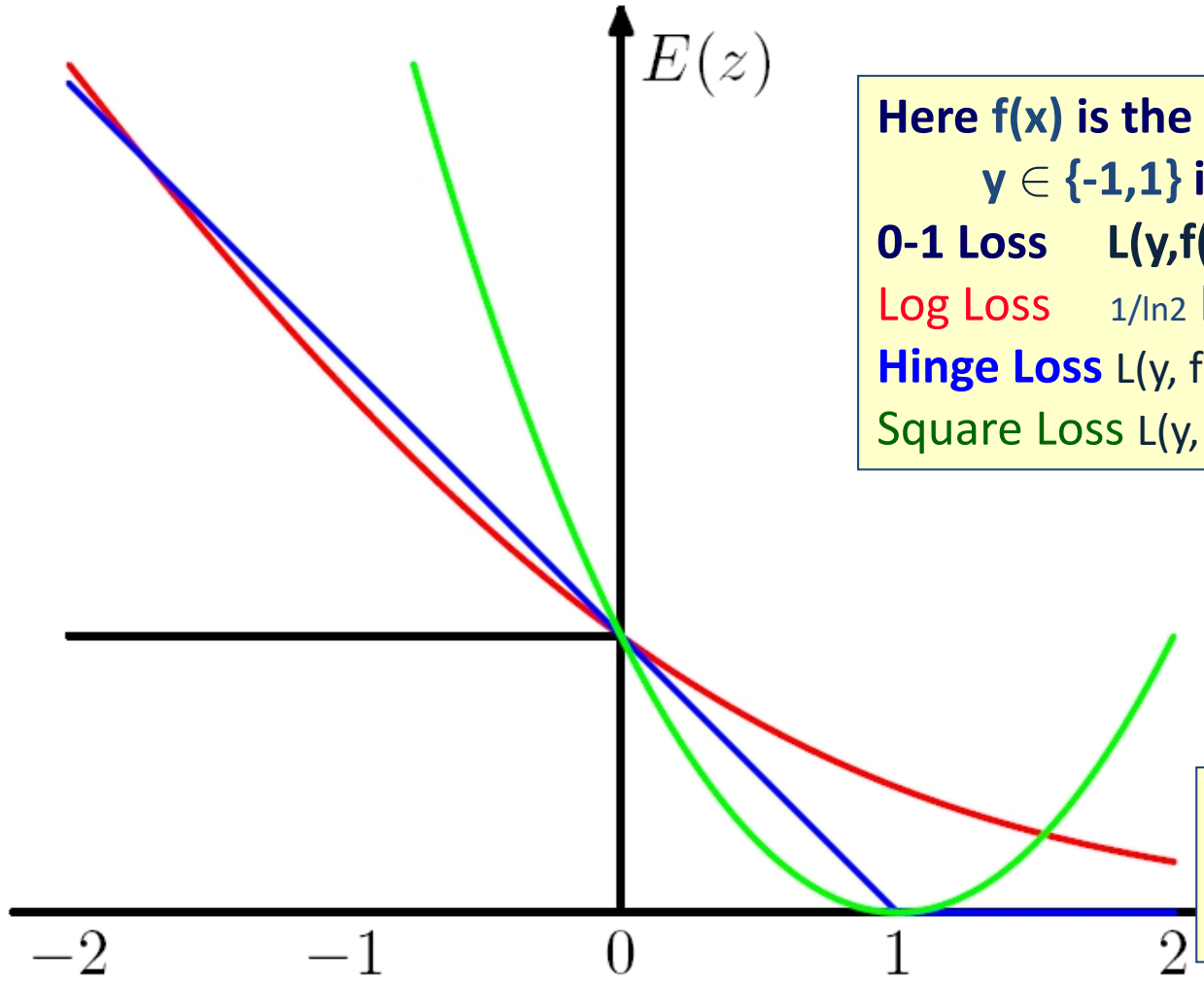
    $L(f(x),y) = (f(x) - y)^2$
  - Input dependent loss:

    $L(f(x),y) = 0$ if $f(x) = y$; $c(x)$ otherwise.

A continuous convex loss function allows a simpler optimization algorithm.



Loss



Here f(x) is the prediction $\in \Re$
y $\in$ {-1,1} is the correct value
0-1 Loss **L(y,f(x))= ½ (1-sgn(yf(x)))**
Log Loss $1/\ln 2 \log(1+\exp\{-yf(x)\})$
Hinge Loss $L(y, f(x)) = \max(0, 1 - y\ f(x))$
Square Loss $L(y, f(x)) = (y - f(x))^2$

0-1 Loss **x axis = yf(x)**
Log Loss = x axis = yf(x)
Hinge Loss: x axis = yf(x)
Square Loss: x axis = (y - f(x)+1)

Example

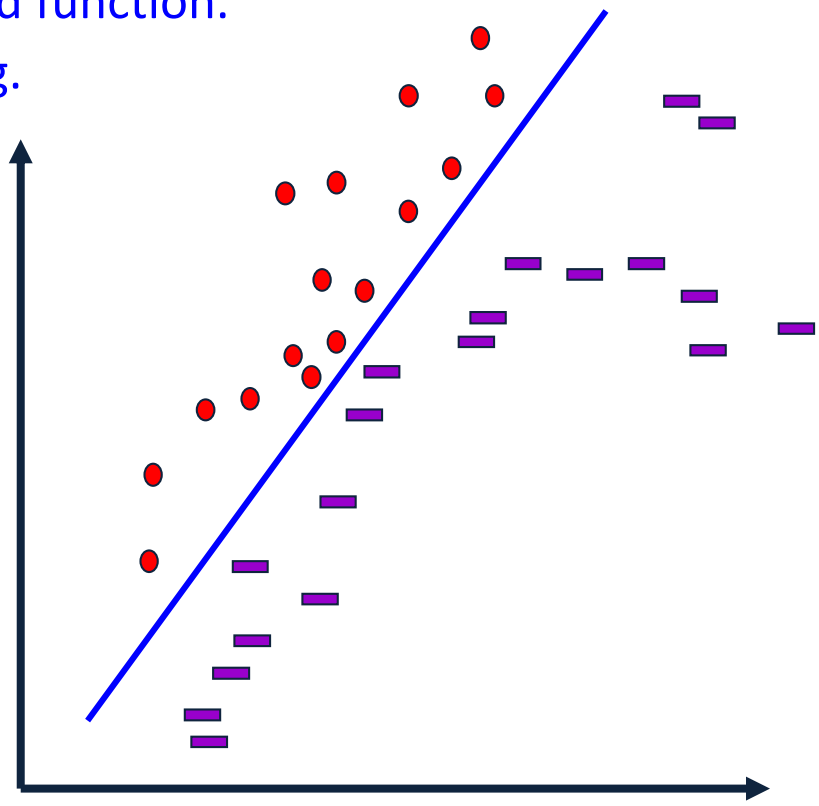
Putting it all together:

A Learning Algorithm

Third Step: How to Learn?

Algorithms

- A possibility: Local search
  - Start with a linear threshold function.
  - See how well you are doing.
  - Correct
  - Repeat until you converge.
- There are other ways that do not search directly in the hypotheses space
  - Directly compute the hypothesis

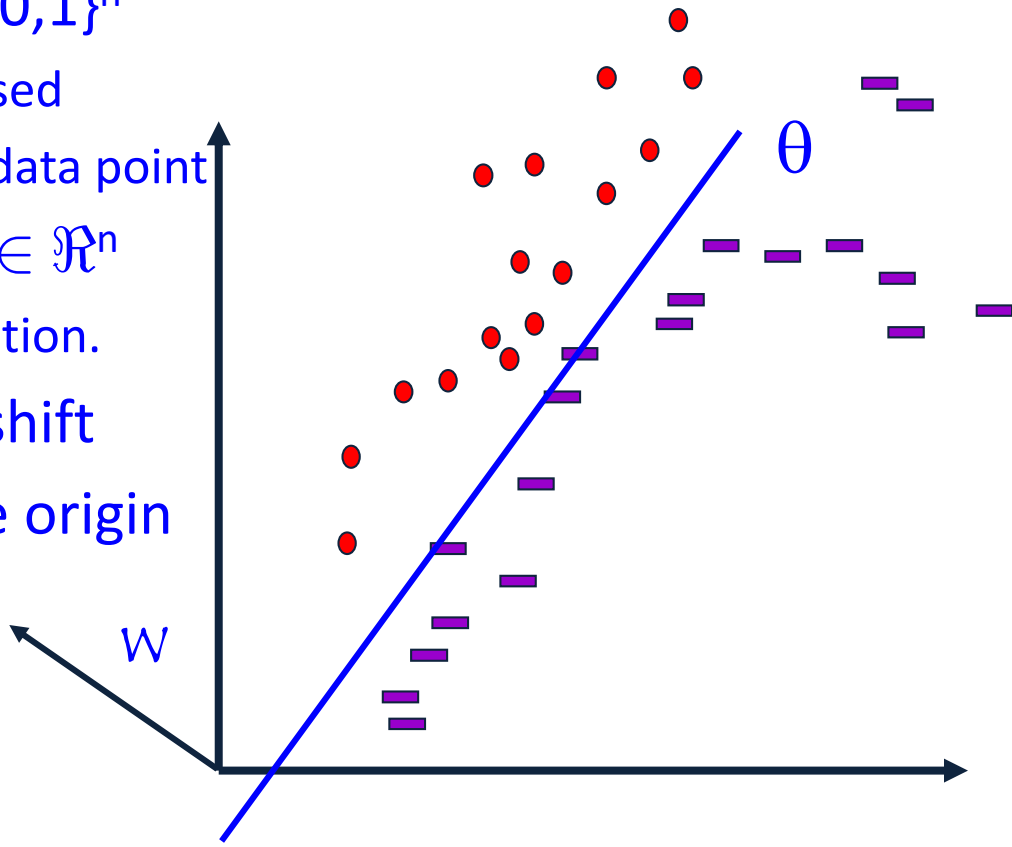


Learning Linear Separators (LTU)

Algorithms

$$f(x) = \text{sgn}\{x^T \cdot w - \theta\} = \text{sgn}\{\sum_{i=1}^{n} w_i x_i - \theta\}$$

- $x^T = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$
  is the feature based encoding of the data point
- $w^T = (w_1, w_2, \ldots, w_n) \in \Re^n$
  is the target function.
- θ determines the shift with respect to the origin



Canonical Representation

$$f(x) = \text{sgn}\{w^T \cdot x - \theta\} = \text{sgn}\{\sum_{i=1}^{n} w_i x_i - \theta\}$$

Algorithms

- $\text{sgn}\{w^T \cdot x - \theta\} \equiv \text{sgn}\{(w')^T \cdot x'\}$
- Where:
  - $x' = (x, -1)$ and $w' = (w, \theta)$
- Moved from an n dimensional representation to an (n+1) dimensional representation, but now can look for hyperplanes that go through the origin.

General Learning Principle

The Risk: a function of w

The loss: a function of x^T, w and y

Algorithms

- **Our goal** is to find a w that minimizes the expected risk

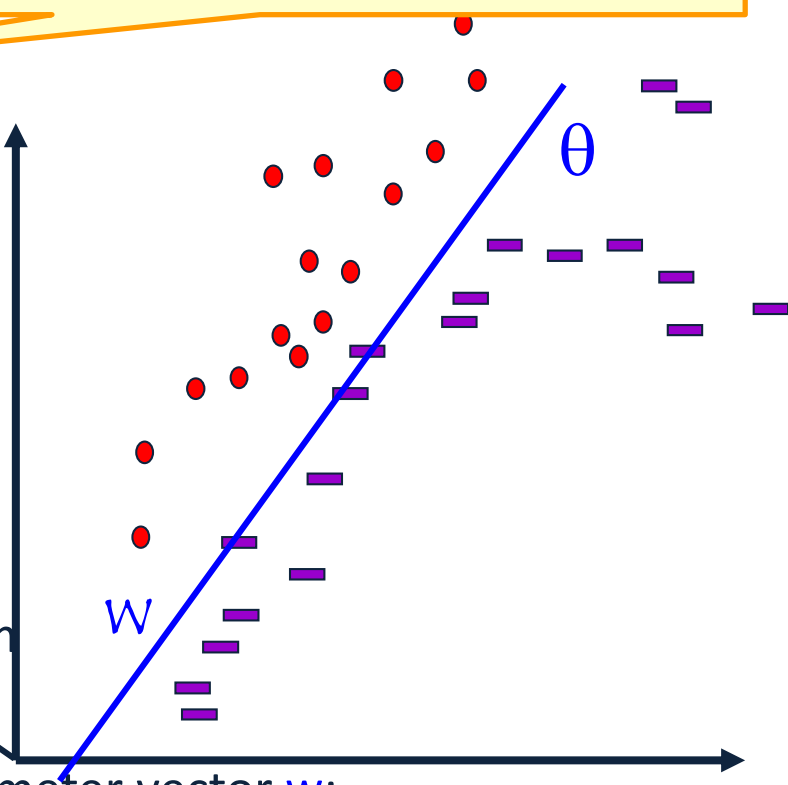
  $J(w) = E_{X,Y}\, Q(x, y, w)$

- We cannot do it.
- **Instead,** we approximate J(w) using a finite training set of independent samples (x_i, y_i)

  $J(w) \sim= \frac{1}{m} \sum_{1,m} Q(x_i, y_i, w)$

- To find the minimum, we use a **batch gradient descent** algorithm
- That is, we successively compute estimates w^t of the optimal parameter vector w:

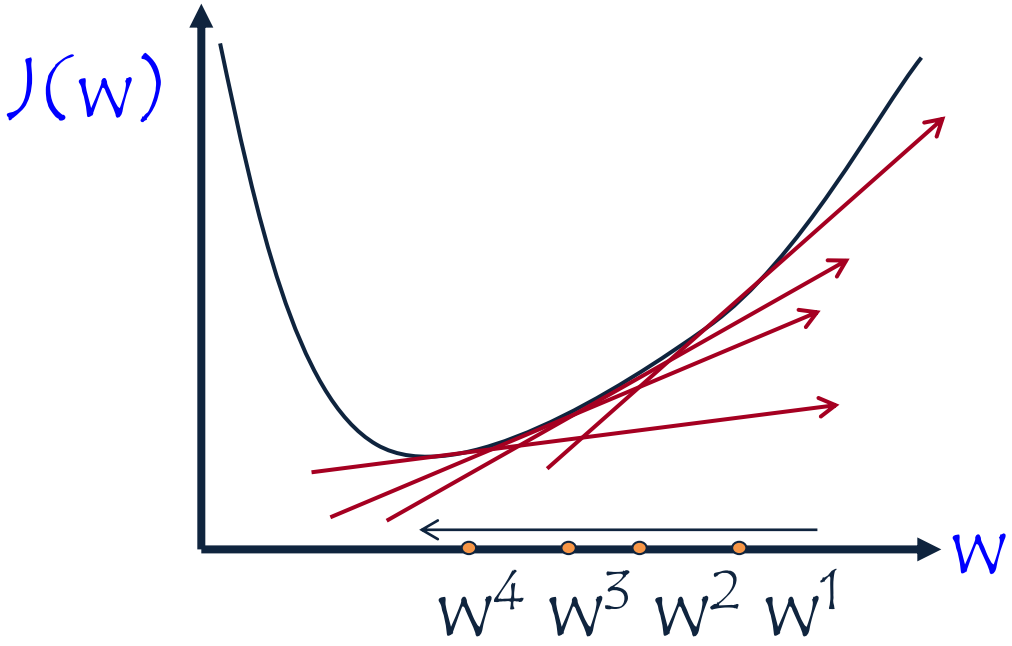
$$w^{t+1} = w^t - \nabla J(w) = w^t - \frac{1}{m} \sum_{1,m} \nabla Q(x_i, y_i, w)$$



Gradient Descent

Algorithms

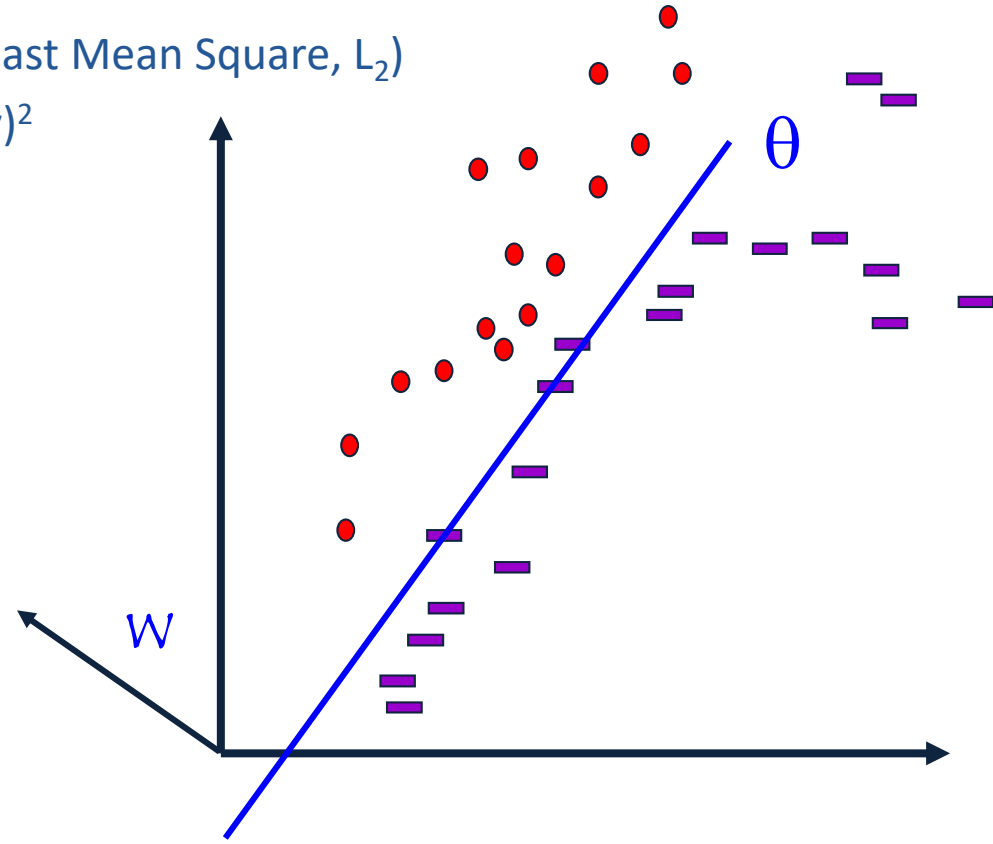
- We use gradient descent to determine the weight vector that minimizes J(w) = Err (w) ;
- Fixing the set D of examples, J=Err is a function of w^j
- At each step, the weight vector is modified in the direction that produces the steepest descent along the error surface.



LMS: An Optimization Algorithm

Algorithms

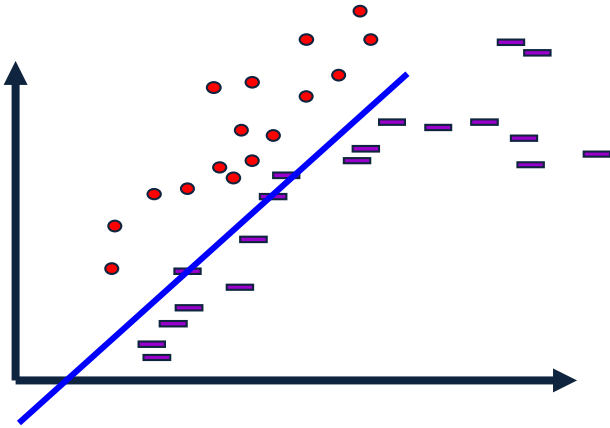
- Our Hypothesis Space is the collection of **Linear Threshold Units**
- Loss function:
  - Squared loss: LMS (Least Mean Square, L_2)
  - $Q(x, y, w) = \frac{1}{2} (w^T x - y)^2$



LMS: An Optimization Algorithm

- (i (subscript) – vector component; j (superscript) - time; d – example #)

Assumption: $x \in R^n$; $u \in R^n$ is the target weight vector; the target (label) is $t_d = u \cdot x$ Noise has been added; so, possibly, no weight vector is consistent with the data.



- Let $\mathbf{w}^{(j)}$ be the current weight vector we have
- Our prediction on the d-th example **x** is:

$$o_d = \sum_i w_i^j \bullet x_i = \vec{w}^{(j)} \bullet \vec{x}$$

- Let t_d be the target value for this example (real value; represents $u \cdot x$)
- The error the current hypothesis makes on the data set is:

$$J(w) = Err(\vec{w}^{(j)}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

Gradient Descent

Algorithms

- To find the best direction in the weight space $\vec{w}$ we compute the gradient of E with respect to each of the components of

$$\nabla E(\vec{w}) \equiv [\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_n}]$$

- This vector specifies the direction that produces the steepest increase in E;
- We want to modify $\vec{w}$ in the direction of $-\nabla E(\vec{w})$

$$\vec{w} = \vec{w} + \Delta \vec{w}$$

- Where (with a fixed step size R):

$$\Delta \vec{w} = -R \nabla E(\vec{w})$$

Gradient Descent: LMS

- We have: $$Err(\vec{w}^{(j)}) = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$
- Therefore: $$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d \in D}(t_d - o_d)^2 =$$

$$= \frac{1}{2}\sum_{d \in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 =$$

$$= \frac{1}{2}\sum_{d \in D} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}_d \bullet \vec{x}_d) =$$

$$= \sum_{d \in D}(t_d - o_d)(-x_{id})$$

Algorithms

Gradient Descent: LMS

Algorithms

- Weight update rule:

$$\Delta w_i = R \sum_{d \in D} (t_d - o_d) x_{id}$$

- Gradient descent algorithm for training linear units:
  - Start with an initial random weight vector
  - For every example d with target value t_d do:
  - Evaluate the linear unit $o_d = \sum_i w_i \bullet x_{id} = \vec{w} \bullet \vec{x}_d$
  - Update $\vec{w}$ by adding Δw_i to each component
  - Continue until E below some threshold

This algorithm always converges to a local minimum of J(w), for small enough steps. Here (LMS for linear regression), the surface contains only a single global minimum, so the algorithm converges to a weight vector with minimum error, regardless of whether the examples are linearly separable.

The surface may have local minimum if the loss function is different.

Incremental (Stochastic) Gradient Descent: LMS

Algorithms

- Weight update rule:

$$\Delta w_i = R(t_d - o_d)x_{id}$$

Dropped the averaging operation. Instead of averaging the gradient of the loss over the complete training set, choose at random a sample (x,y) (or a subset of examples) and update w^t

- Gradient descent algorithm for training linear units:
  - Start with an initial random weight vector
  - For every example d with target value t_d do:
    - Evaluate the linear unit $o_d = \sum_i w_i \bullet x_{id} = \vec{w} \bullet \vec{x}_d$
    - update $\vec{w}$ by incrementally by adding Δw_i to each component (update without summing over all data)
  - Continue until E below some threshold

Incremental (Stochastic) Gradient Descent: LMS

Algorithms

- Weight update rule:

$$\Delta w_i = R(t_d - o_d)x_{id}$$

Dropped the averaging operation. Sometimes called "**on-line**" since we don't need a reference to a training set: **observe – predict – get feedback.**

- Gradient descent algorithm for training linear units:
  - Start with an initial random weight vector
  - For every example d with target value: t_d
    - Evaluate the linear unit $o_d = \sum_i w_i \bullet x_{id} = \vec{w} \bullet \vec{x}_d$
    - update $\vec{w}$ by incrementally adding Δw_i to each component (update without summing over all data)
  - Continue until E below some threshold
- In general - does not converge to global minimum
- Decreasing R with time guarantees convergence
- But, on-line algorithms are sometimes advantageous…

Learning Rates and Convergence

Algorithms

- In the general (non-separable) case the learning rate R must decrease to zero to guarantee convergence.
- The learning rate is called the *step size.* There are more sophisticated algorithms that choose the step size automatically and converge faster.
- Choosing a better starting point also has impact.

- The gradient descent and its stochastic version are very simple algorithms, but almost all the algorithms we will learn in the class can be traced back to gradient decent algorithms for different loss functions and different hypotheses spaces.

Computational Issues

Algorithms

- Assume the data is linearly separable.
- Sample complexity:
  - Suppose we want to ensure that our LTU has an error rate (on new examples) of less than ε with high probability (at least $(1-\delta)$)
  - How large does m (the number of examples) must be in order to achieve this ? It can be shown that for n dimensional problems

$$m = O(1/\varepsilon [\ln(1/\delta) + (n+1) \ln(1/\varepsilon)].$$

- Computational complexity: What can be said?
  - It can be shown that there exists a polynomial time algorithm for finding consistent LTU (by reduction from linear programming).
  - [Contrast with the NP hardness for 0-1 loss optimization]
  - (On-line algorithms have inverse quadratic dependence on the margin)

Other Methods for LTUs

Algorithms

- Fisher Linear Discriminant:
  - A direct computation method
- Probabilistic methods (naïve Bayes):
  - Produces a stochastic classifier that can be viewed as a linear threshold unit.
- Winnow/Perceptron
  - A multiplicative/additive update algorithm with some sparsity properties in the function space (a large number of irrelevant attributes) or features space (sparse examples)
- Logistic Regression, SVM…many other algorithms