CS446 Introduction to Machine Learning (Fall 2013)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURE 12: MIDTERM REVIEW

Prof. Julia Hockenmaier
juliahmr@illinois.edu

Today's class

Quick run-through of the material we've covered so far

The selection of slides in today's lecture doesn't mean that you don't need to look at the rest when prepping for the exam!

Midterm
(Thursday, Oct 10 in class)

Format

Closed book exam (during class):

- You are not allowed to use any cheat sheets, computers, calculators, phones etc. (you shouldn't have to anyway)
- Only the material covered in lectures (Assignments have gone beyond what's covered in class)
- Bring a pen (black/blue).

Sample questions

What is *n*-fold cross-validation, and what is its advantage over standard evaluation?

Good solution:

- Standard evaluation: split data into test and training data (optional: validation set)
- *n*-fold cross validation: split the data set into *n* parts, run *n* experiments, each using a different part as test set and the remainder as training data.
- Advantage of *n*-fold cross validation: because we can report expected accuracy, and variances/standard deviation, we get better estimates of the performance of a classifier.

Question types

- *Define* X:
  Provide a mathematical/formal definition of X
- *Explain* what X is/does:
  Use plain English to say what X is/does
- *Compute* X:
  Return X; Show the steps required to calculate it
- *Show/Prove* that X is true/false/…:
  This requires a (typically very simple) proof.

CS446 Introduction to Machine Learning (Fall 2013)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURES 1 & 2: INTRO/SUPERVISED LEARNING

Prof. Julia Hockenmaier
juliahmr@illinois.edu

CS446: Key questions

- What kind of tasks can we learn models for?
- What kind of models can we learn?
- What algorithms can we use to learn?
- How do we evaluate how well we have learned to perform a particular task?
- How much data do we need to learn models for a particular task?

Learning scenarios

Supervised learning:

The focus of CS446

Learning to predict labels from correctly labeled data

Unsupervised learning:

Learning to find hidden structure (e.g. clusters) in input data

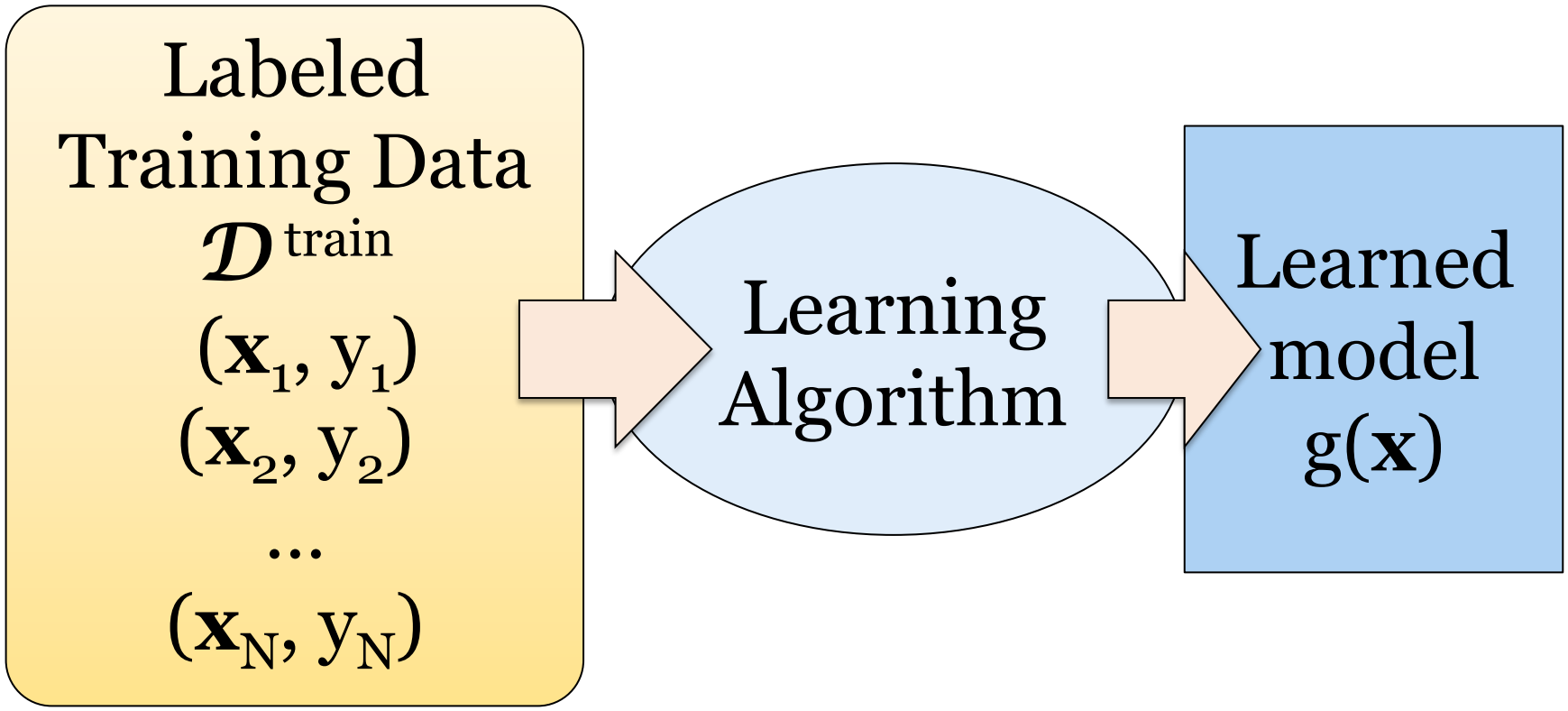
Semi-supervised learning:

Learning to predict labels from (a little) labeled and (a lot of) unlabeled data

Reinforcement learning:

Learning to act through feedback for actions (rewards/punishments) from the environment

Supervised learning: Training

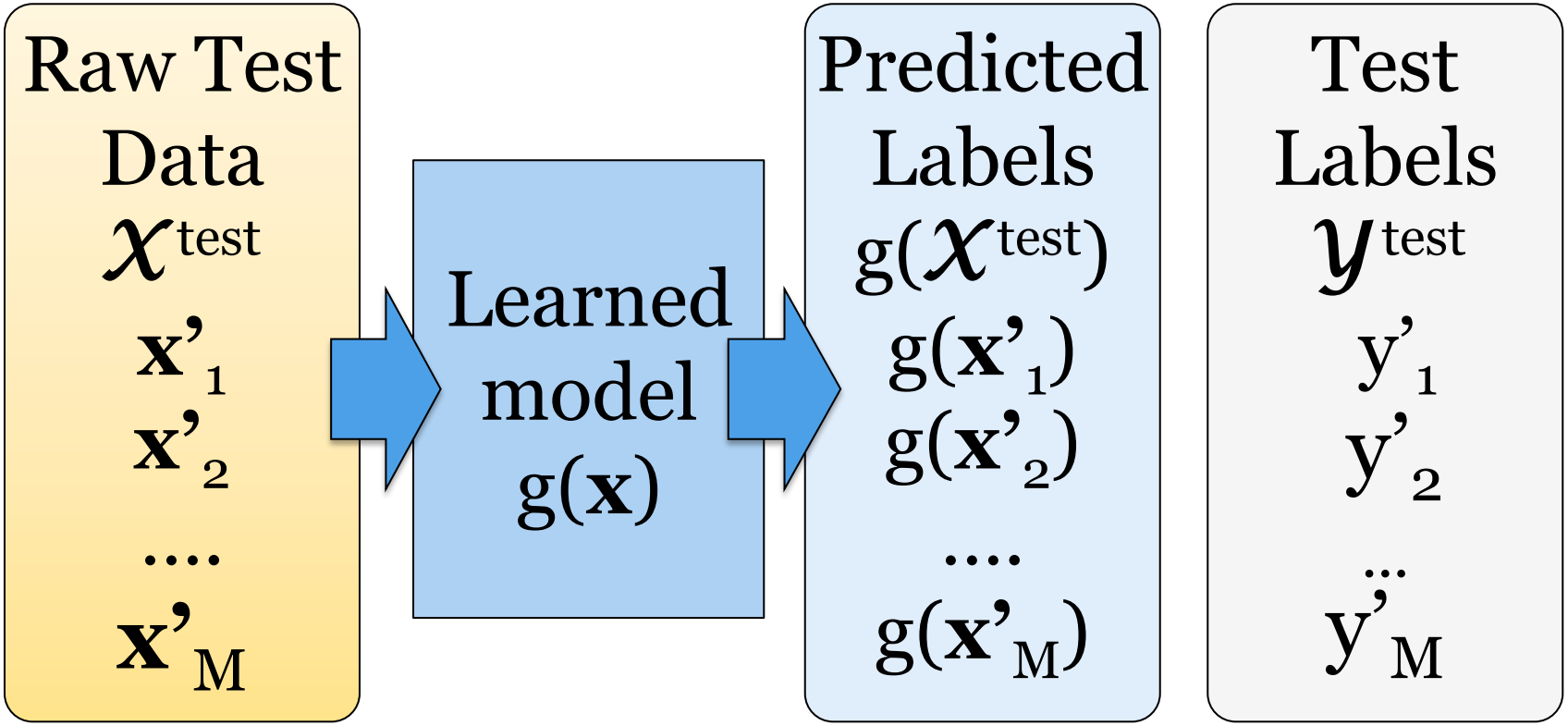


Give the learner examples in $\mathcal{D}^{\text{train}}$

The learner returns a model $g(\mathbf{x})$

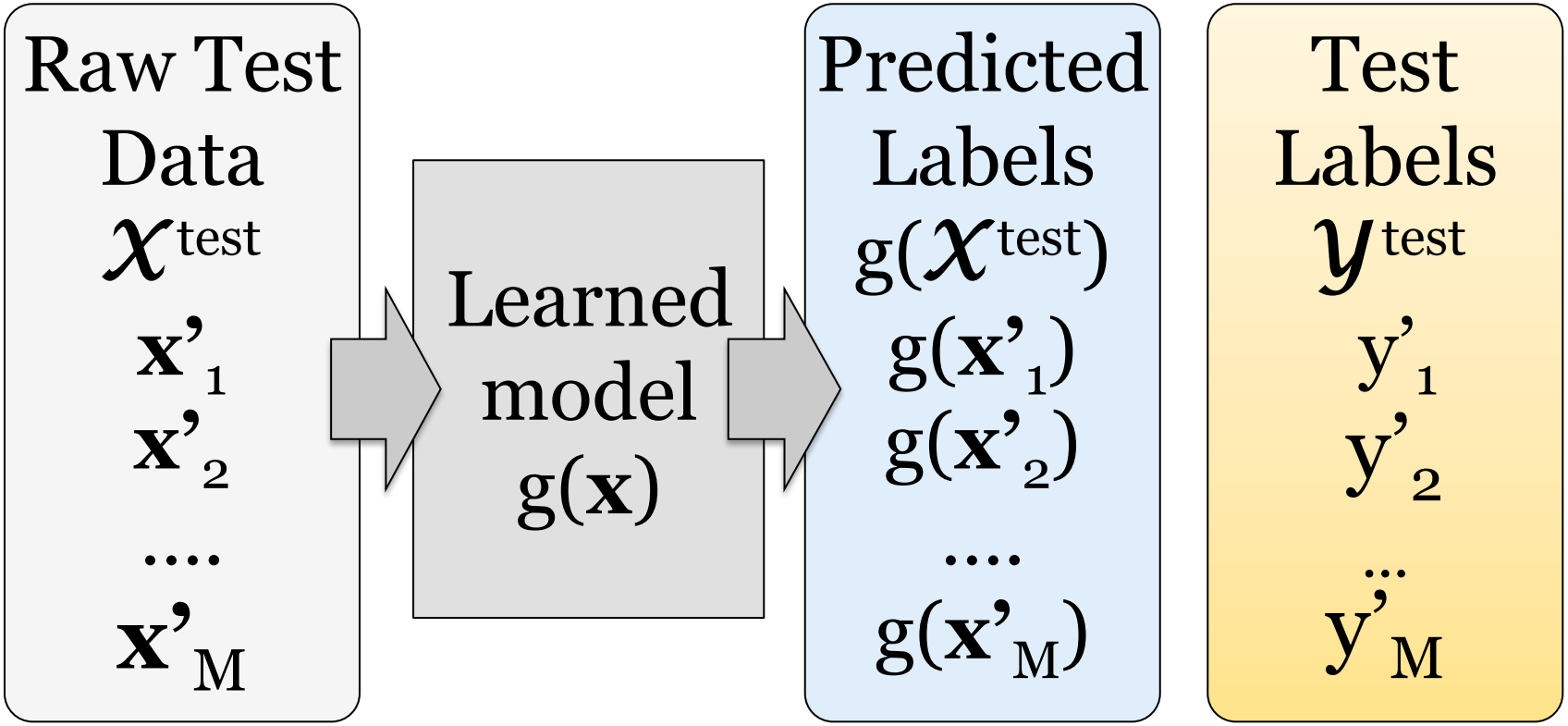
Supervised learning: Testing

Apply the model to the raw test data



Supervised learning: Testing

Evaluate the model by comparing the predicted labels against the test labels



Evaluating supervised learners

Use a test data set that is disjoint from $\mathcal{D}^{\text{train}}$

$\mathcal{D}^{\text{test}} = \{(\mathbf{x'}_1, y'_1), \ldots, (\mathbf{x'}_M, y'_M)\}$

The learner has not seen the test items during learning.

Split your labeled data into two parts: test and training.

Take all items $\mathbf{x'}_i$ in $\mathcal{D}^{\text{test}}$ and compare the predicted $f(\mathbf{x'}_i)$ with the correct y'_i .

This requires an evaluation metric (e.g. accuracy).

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 model 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?

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?

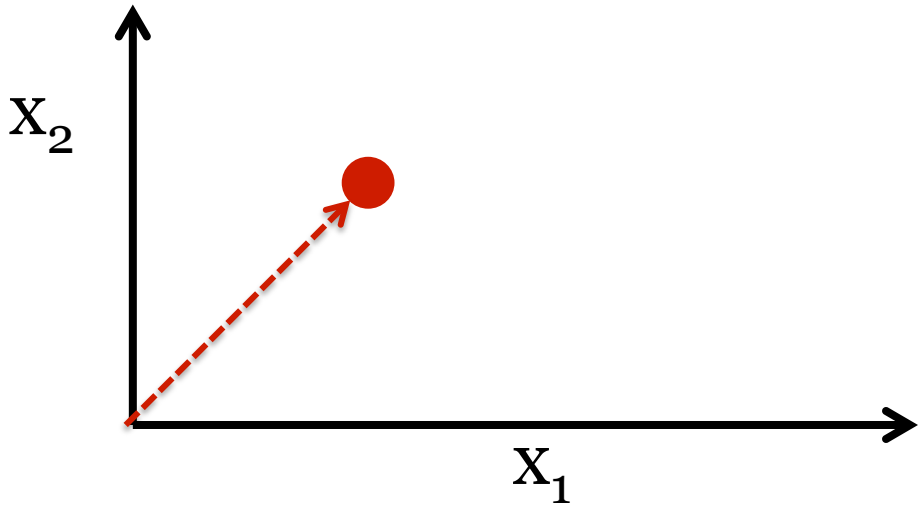
$\mathcal{X}$ as a vector space

$\mathcal{X}$ is an N-dimensional vector space (e.g. $\mathbb{R}^N$)

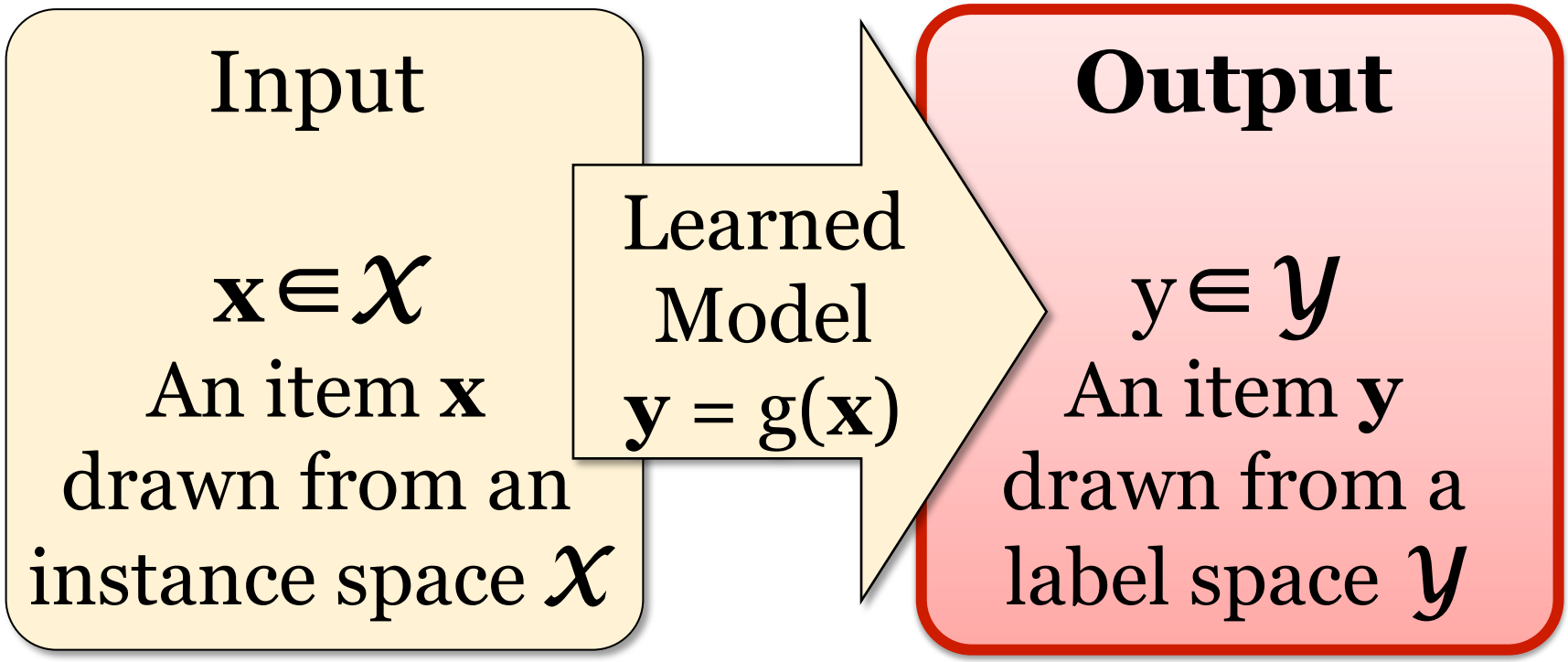
Each dimension = one feature.

Each $\mathbf{x}$ is a feature vector (hence the boldface $\mathbf{x}$).

Think of $\mathbf{x} = [x_1 \ldots x_N]$ as a point in $\mathcal{X}$:



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

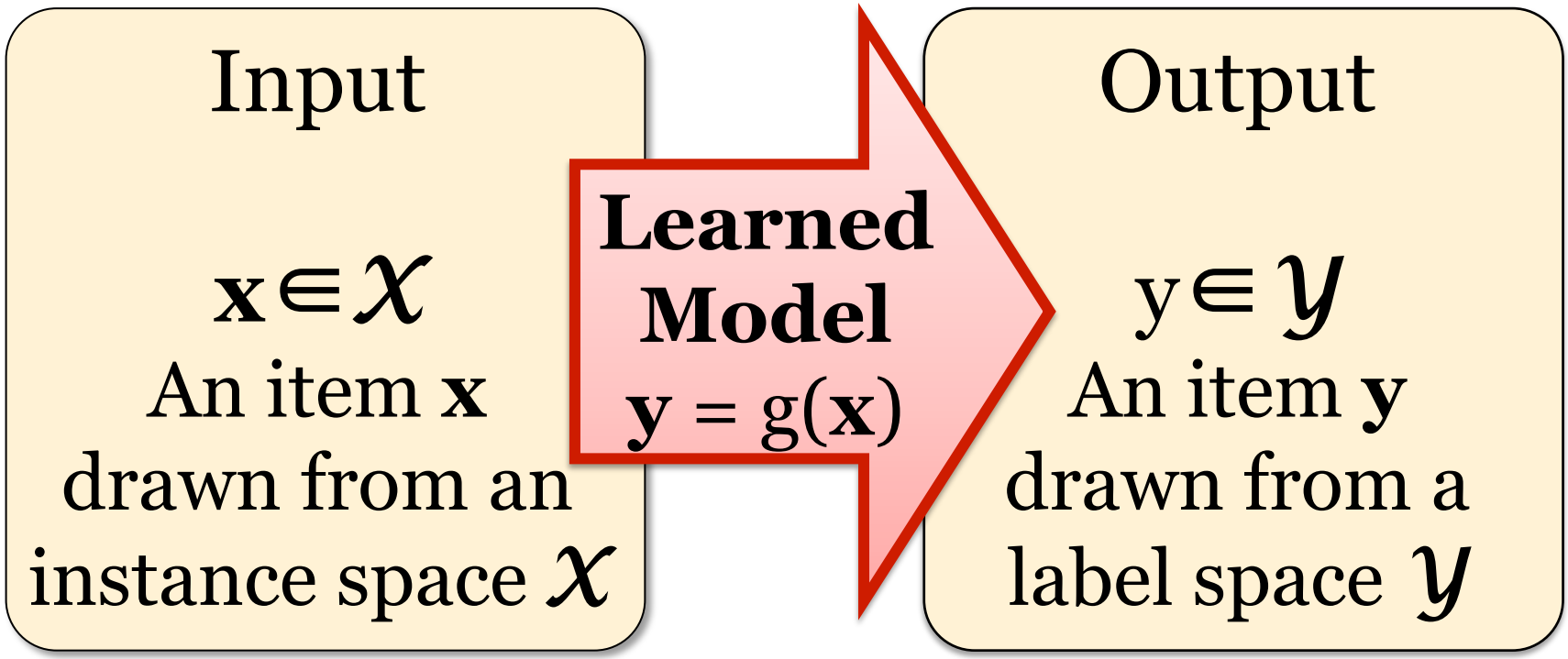
Output labels $y \in \mathcal{Y}$ are **categorical**: The focus of CS446

- Binary classification: Two possible labels
- Multiclass classification: k possible labels

Output labels $y \in \mathcal{Y}$ are **structured objects** (sequences of labels, parse trees, etc.)

- Structure learning (CS546 next semester)

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



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

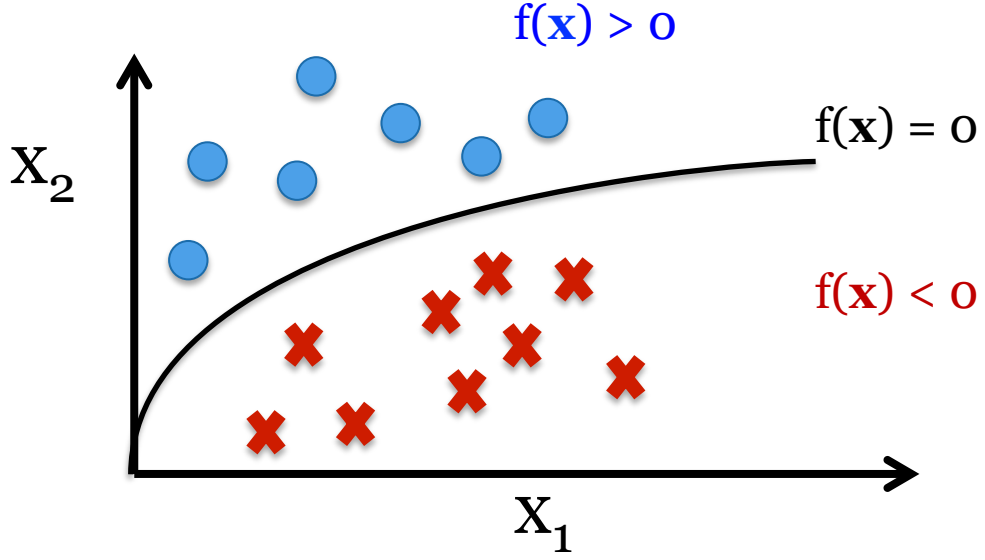
The hypothesis space $\mathcal{H}$

There are $|\mathcal{Y}|^{|\mathcal{X}|}$ possible functions f($\mathbf{x}$) from the instance space $\mathcal{X}$ to the label space $\mathcal{Y}$.

Learners typically consider only a *subset* of the functions from $\mathcal{X}$ to $\mathcal{Y}$, called the hypothesis space $\mathcal{H}$.

$$\mathcal{H} \subseteq |\mathcal{Y}|^{|\mathcal{X}|}$$

Classifiers in vector spaces



Binary classification:

We assume f *separates* the positive and negative examples:

- Assign y = 1 to all **x** where f(**x**) > 0
- Assign y = 0 to all **x** where f(**x**) < 0

Criteria for choosing models

Accuracy:

Prefer models that make fewer mistakes

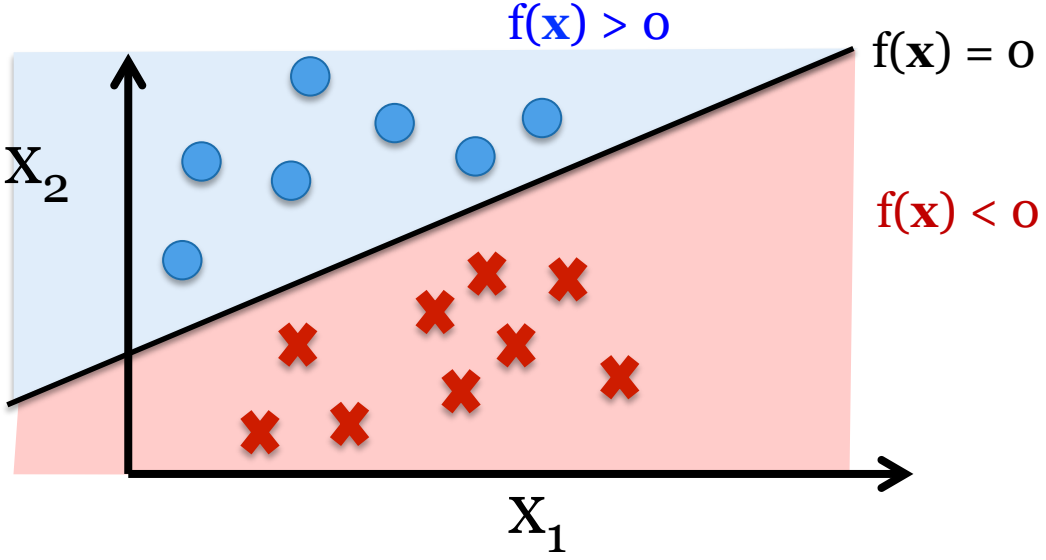
- We only have access to the training data
- But we care about accuracy on unseen (test) examples

Simplicity (Occam's razor):

Prefer simpler models (e.g. fewer parameters).

- These (often) generalize better, and need less data for training.

Linear classifiers



Many learning algorithms restrict the hypothesis space to **linear classifiers**:

$$f(\mathbf{x}) = w_0 + \mathbf{wx}$$

4. The learning algorithm

The learning task:

Given a labeled training data set

$$\mathcal{D}^{\text{train}} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$$

return a model (classifier) g: $\mathcal{X} \mapsto \mathcal{Y}$

from the hypothesis space $\mathcal{H} \subseteq |\mathcal{Y}|^{|\mathcal{X}|}$

The learning algorithm performs a search in the hypothesis space $\mathcal{H}$ for the model g.

Batch versus online training

Batch learning:

The learner sees the complete training data, and only changes its hypothesis when it has seen the entire training data set.

Online training:

The learner sees the training data one example at a time, and can change its hypothesis with every new example

CS446 Introduction to Machine Learning (Fall 2013)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURES 3 & 4: DECISION TREES

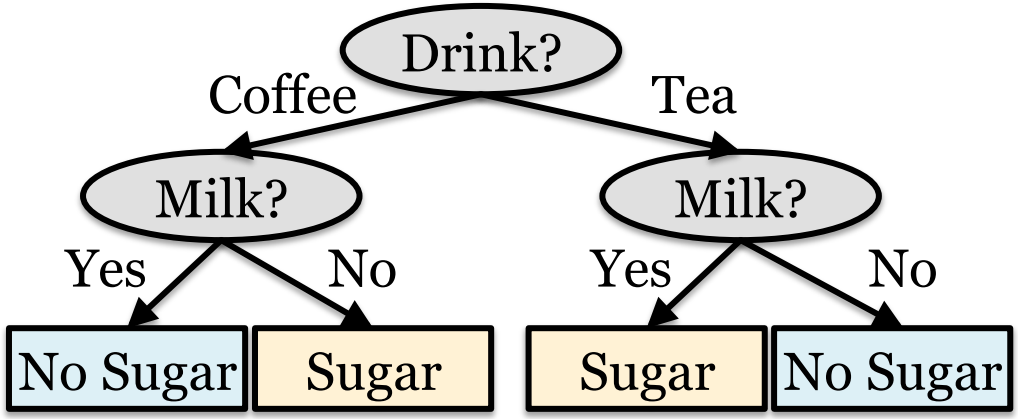
Prof. Julia Hockenmaier
juliahmr@illinois.edu

Decision trees are classifiers

Non-leaf nodes test the value of one feature

- Tests: yes/no questions; switch statements
- Each child = a different value of that feature

Leaf-nodes assign a class label



How expressive are decision trees?

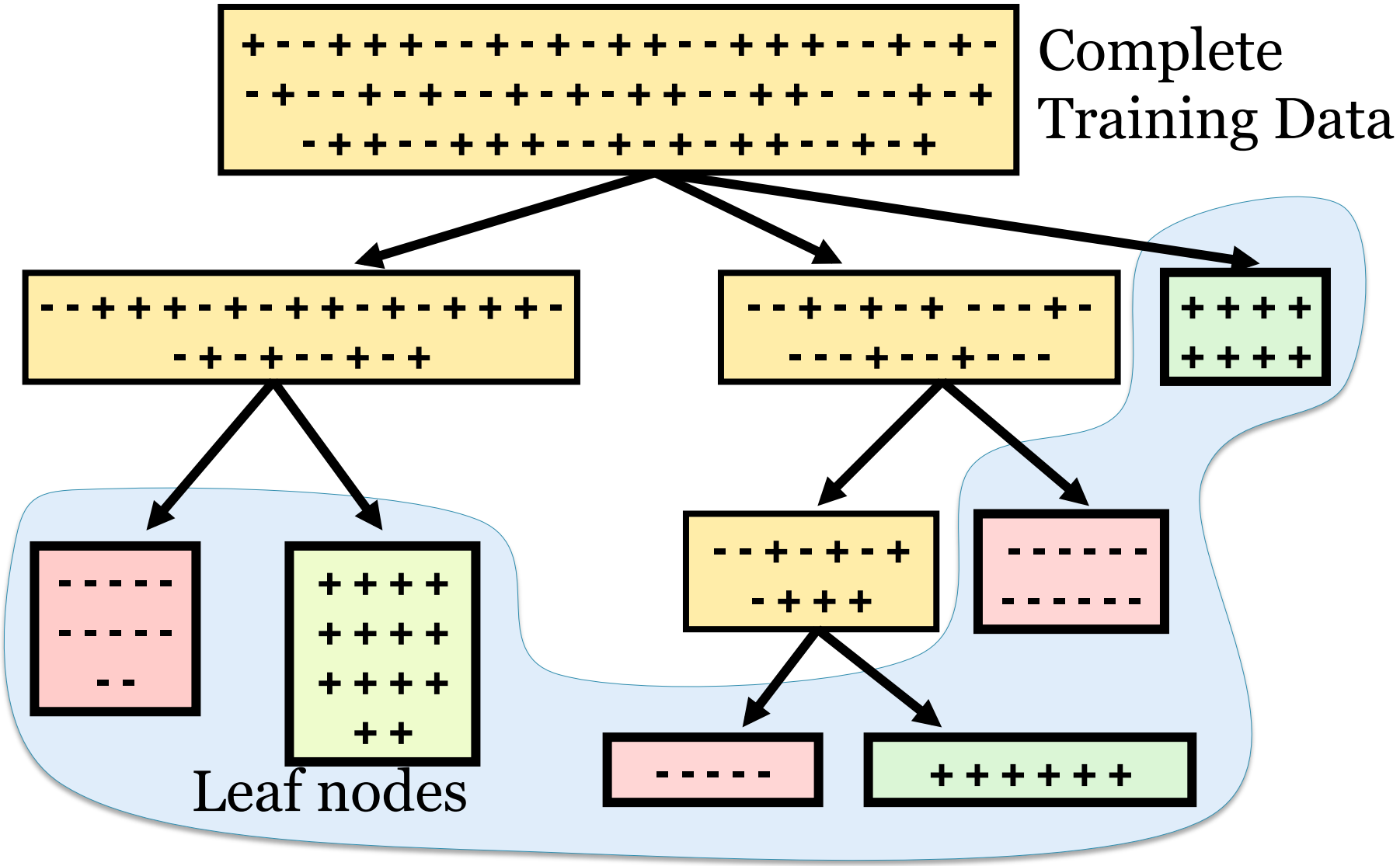
Hypothesis spaces for binary classification:

Each hypothesis h $\in \mathcal{H}$ assigns *true*
to one subset of the instance space $\mathcal{X}$

Decision trees do not restrict $\mathcal{H}$:

There is a decision tree for every hypothesis
Any subset of $\mathcal{X}$ can be identified via yes/no questions

Learning decision trees



How do we split a node **N**?

The node **N** is associated with a subset S of the training examples.

- If all items in S have the same class label, **N** is a leaf node
- Else, split on the values $V_F = \{v_1, \ldots, v_K\}$ of the most informative feature F :

  For each $v_k \in V_F$: add a new child $\mathbf{C}_k$ to **N**.

  $\mathbf{C}_k$ is associated with S_k, the subset of items in S where F takes the value v_k

Using entropy to guide decision tree learning

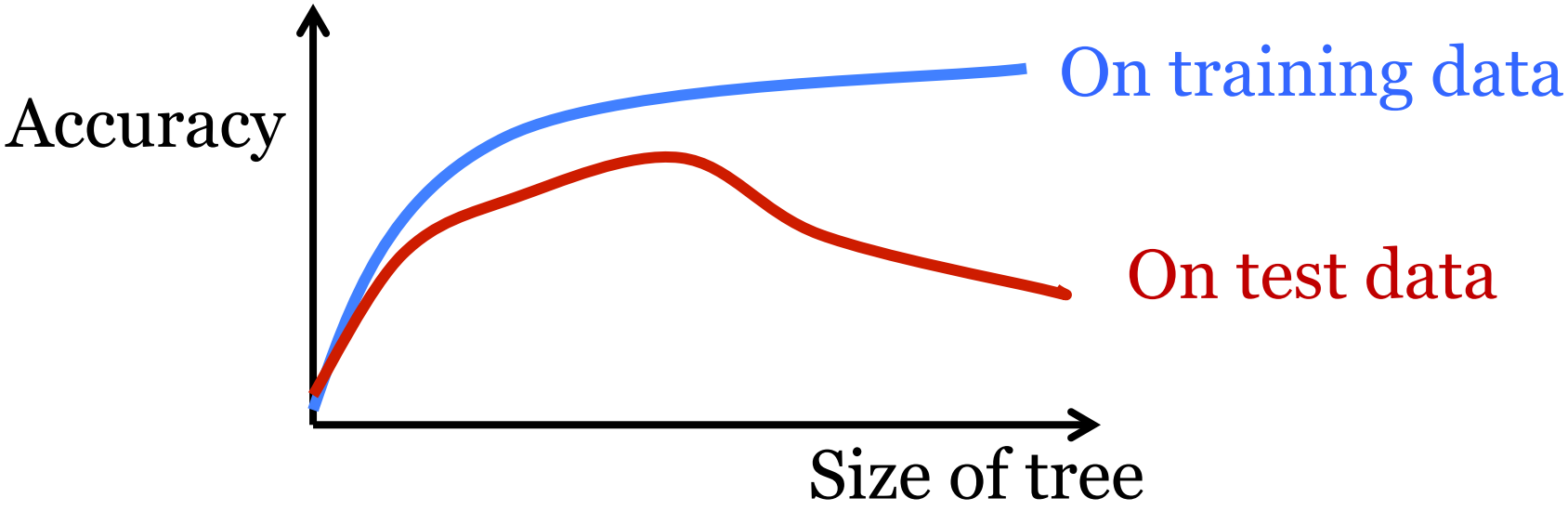
- The parent S has entropy $H(S)$ and size $|S|$
- Splitting S on feature $\mathbf{X}_i$ with values 1,…,k yields k children S_1, …, S_k with entropy $H(S_k)$ & size $|S_k|$
- After splitting S on $\mathbf{X}_i$ the expected entropy is

$$\sum_k \frac{|S_k|}{|S|} H(S_k)$$

- When we split S on $\mathbf{X}_i$, the information gain is:

$$Gain(S, X_i) = H(S) - \sum_k \frac{|S_k|}{|S|} H(S_k)$$

Overfitting



A decision tree overfits the training data when its accuracy on the training data goes up but its accuracy on unseen data goes down

Reasons for overfitting

Too much variance in the training data

- Training data is not a representative sample of the instance space
- We split on features that are actually irrelevant

Too much noise in the training data

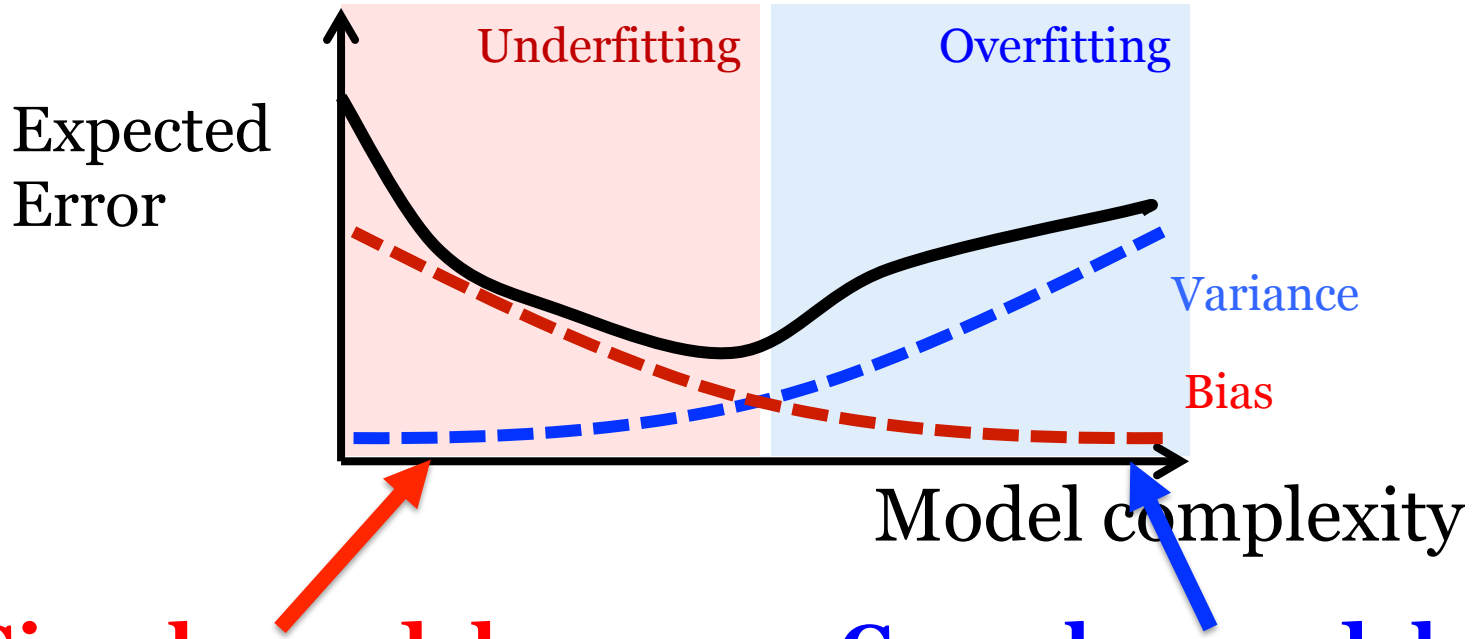
- Noise = some feature values or class labels are incorrect
- We learn to predict the noise

Reducing overfitting

Various heuristics are commonly used:

- Limit the depth of the tree
- Require a minimum number of examples per node used to select a split
- Learn a complete tree and prune, using validation (held-out) data

Underfitting and Overfitting

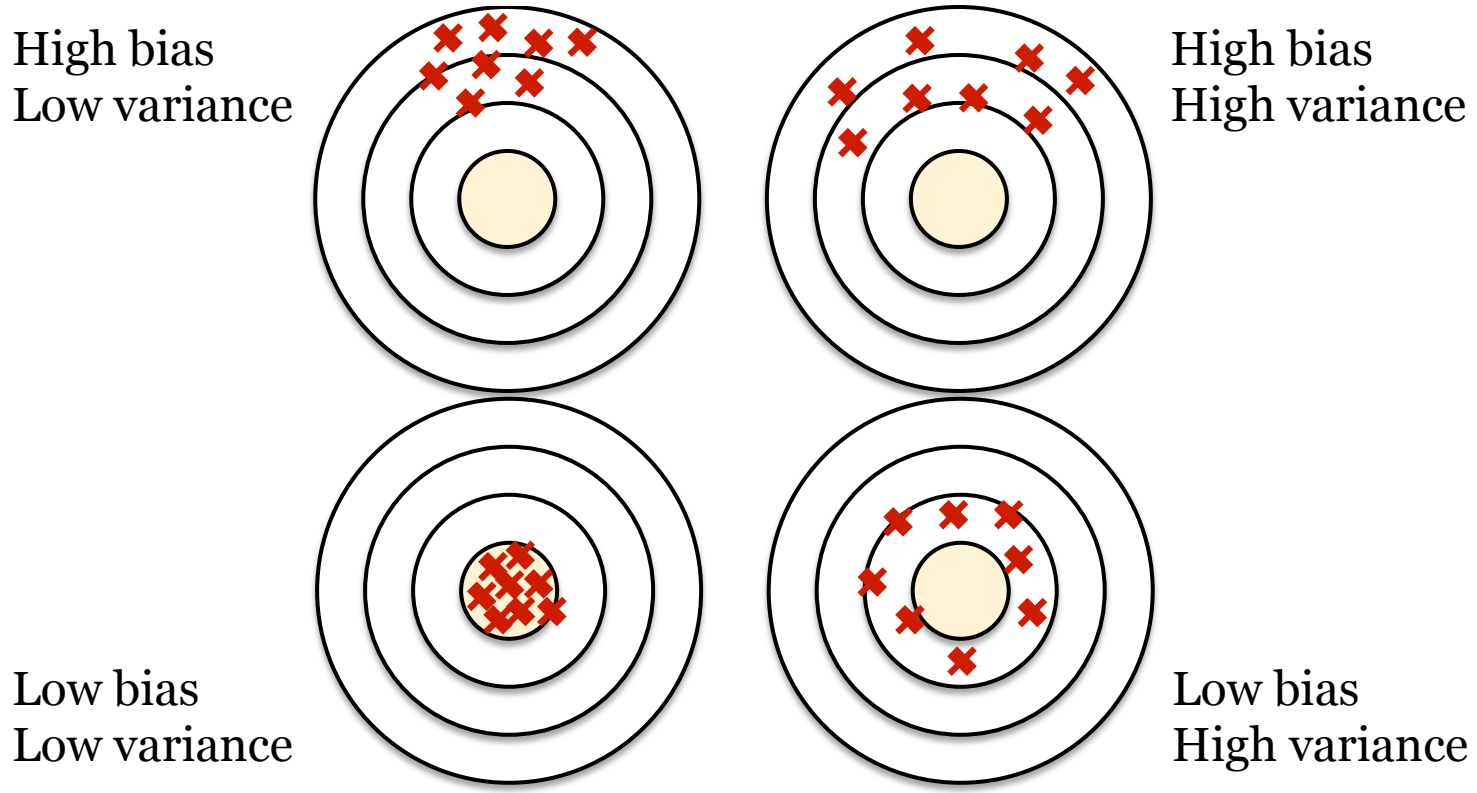


Simple models: High bias and low variance

Complex models: High variance and low bias

Bias-variance tradeoffs

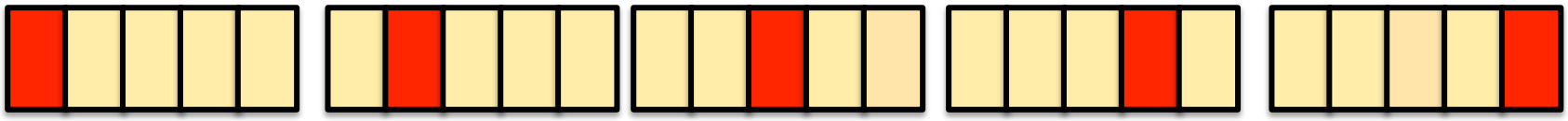
Dartboard = hypothesis space
Bullseye = target function
Darts = learned models



N-fold cross validation

Instead of a single test-training split:

- Split data into N equal-sized parts



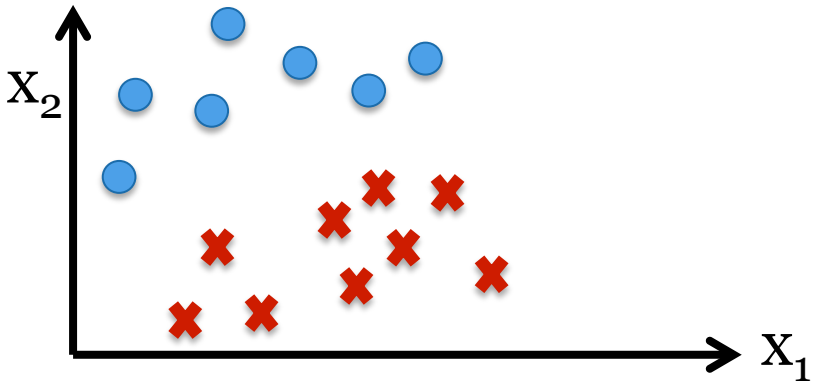
- Train and test N different classifiers
- Report average accuracy and standard deviation of the accuracy

CS446 Introduction to Machine Learning (Fall 2013)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

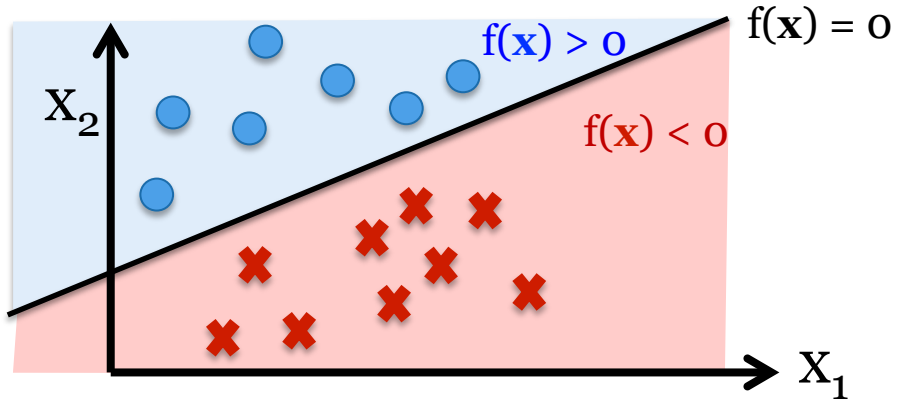
LECTURE 5: LINEAR CLASSIFIERS

Prof. Julia Hockenmaier
juliahmr@illinois.edu

Learning a linear classifier

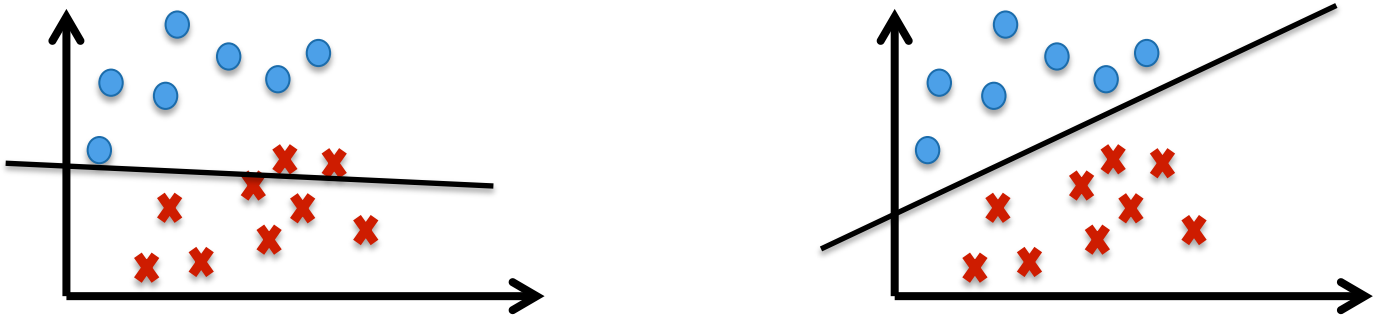


Input: Labeled training data
$\mathcal{D} = \{(\mathbf{x}^1, y^1),\ldots,(\mathbf{x}^D, y^D)\}$
plotted in the sample space $\mathcal{X} = \mathbf{R}^2$
with ●: $y^i = +1$, ✖: $y^i = -1$



Output: A decision boundary $f(\mathbf{x}) = 0$ that separates the training data
$y^i \cdot f(\mathbf{x}^i) > 0$

Which model should we pick?



We need a metric (aka an objective function)

We would like to minimize the probability of misclassifying *unseen* examples, but we can't measure that probability.

Instead: minimize the number of misclassified training examples

The **empirical risk** of f(**x**)

The empirical risk of a classifier f(**x**) = **w**·**x**
on data set $\mathcal{D}$ = {(**x**^{1}, y^1),…,(**x**^{D}, y^D)}
is its average loss on the items in $\mathcal{D}$

$$R_{\mathcal{D}}(\mathrm{f}) = \frac{1}{D}\sum_{i=1}^{D} L(y^i, \mathrm{f}(\mathbf{x}^i))$$

Realistic learning objective:
Find an f that minimizes empirical risk

(Note that the learner can ignore the constant 1/D)

Empirical risk minimization

Learning:
Given training data $\mathcal{D} = \{(\mathbf{x}^1, y^1),\ldots,(\mathbf{x}^D, y^D)\}$, return the classifier $f(\mathbf{x})$ that minimizes the empirical risk $R_{\mathcal{D}}(f)$

Loss functions for classification

L($y, f(\mathbf{x})$) is the loss (aka cost) of classifier f on example $\mathbf{x}$ when the true label of $\mathbf{x}$ is y.

We assign label $\hat{y}$ = sgn(f($\mathbf{x}$)) to $\mathbf{x}$

Loss = what penalty do we incur if we misclassify $\mathbf{x}$?

Plots of L($y, f(\mathbf{x})$):
x-axis is typically y·f($\mathbf{x}$)

Today: 0-1 loss and square loss
(more loss functions later)

Gradient Descent

Iterative batch learning algorithm:

- Learner updates the hypothesis based on the entire training data
- Learner has to go multiple times over the training data

Goal: Minimize training error/loss

- At each step: move **w** in the direction of *steepest descent* along the error/loss surface

Stochastic Gradient Descent

Online learning algorithm:

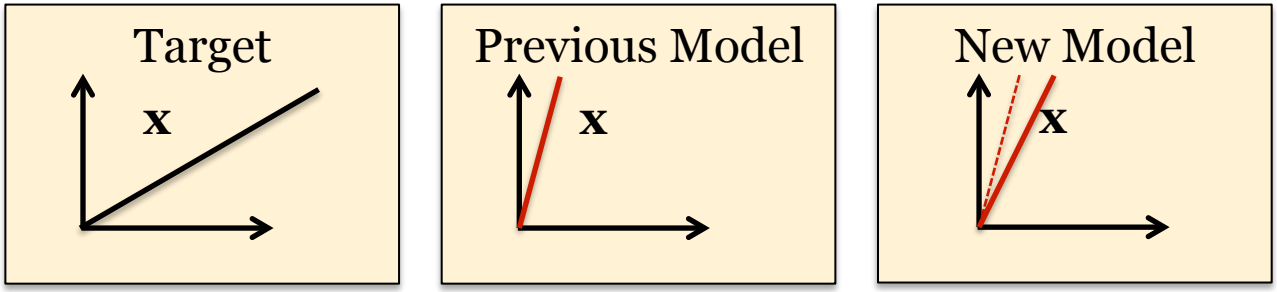
- Learner updates the hypothesis with each training example
- No assumption that we will see the same training examples again
- Like batch gradient descent, except we update after seeing each example

Perceptron and Winnow

The Perceptron rule

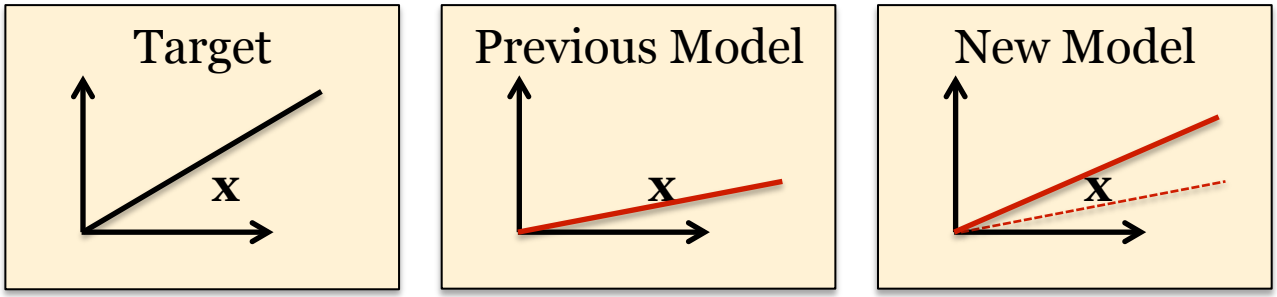
If y = +1: **x** should be above the decision boundary

Raise the decision boundary's slope: $\mathbf{w}^{i+1} := \mathbf{w}^{i} + \mathbf{x}$



If y = -1: **x** should be below the decision boundary

Lower the decision boundary's slope: $\mathbf{w}^{i+1} := \mathbf{w}^{i} - \mathbf{x}$



Comparison: Perceptron

Perceptron

$\mathbf{w}^{i+1} = \mathbf{w}^i + y^i\mathbf{x}^i$ if $\mathbf{w}^k$ misclassifies $\mathbf{x}^i$,

$\mathbf{w}^{i+1} = \mathbf{w}^i$ otherwise

Converges after a finite number of mistakes
if the data are linearly separable
(otherwise, it cycles)

Rate of convergence depends on the number of
active features in each item
(good when the input $\mathbf{x}$ is sparse)

Winnow update

if $\mathbf{w}^k$ misclassifies $\mathbf{x}^i$:

if $y^i = +1$:

double the weights of the features that are active in $\mathbf{x}^i$

if $y^i = -1$:

halve the weights of the features that are active in $\mathbf{x}^i$

(don't touch weights of inactive features)

Comparison: Winnow

if $\mathbf{w}^k$ misclassifies $\mathbf{x}^i$:

    if $y^i = +1$:

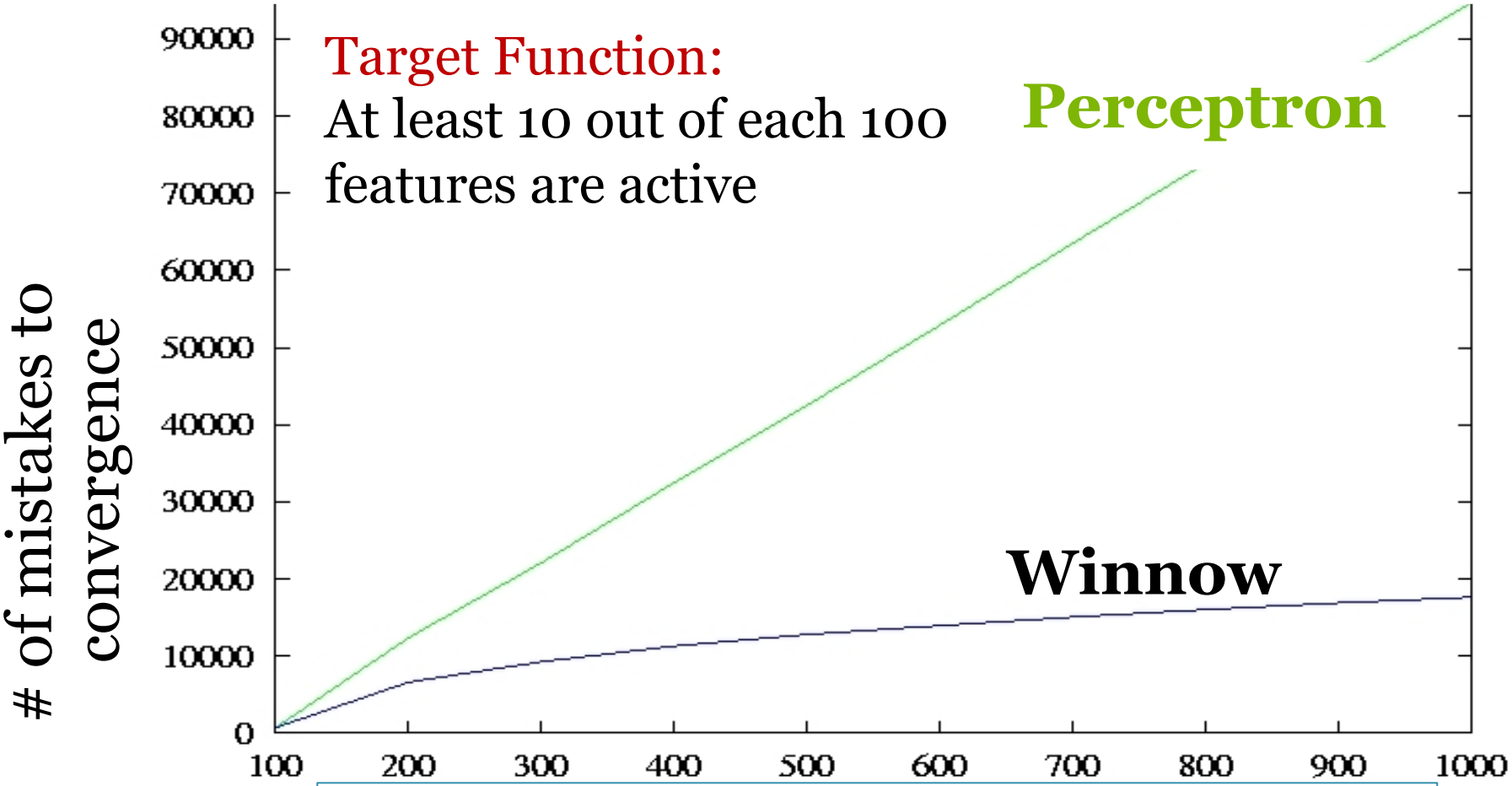
        double the weights of the features that are active in $\mathbf{x}^i$

    if $y^i = -1$:

        halve the weights of the features that are active in $\mathbf{x}^i$

Scales well when many features are irrelevant for the target concept (good when the target weight vector $\mathbf{w}$ is sparse)

Mistakes bounds comparison



Concept learning

Learning Conjunctions

Task: learn a hidden (monotone) conjunction

$$f = x_2 \wedge x_3 \wedge x_4 \wedge x_5 \wedge x_{100}$$

How many examples are needed to learn it? How?

- **Protocol I:** The learner proposes instances as queries to the teacher
- **Protocol II:** The teacher (who knows f) provides training examples
- **Protocol III:** Some random source (e.g., Nature) provides training examples; the Teacher (Nature) provides the labels (f(**x**))

Which Boolean functions can be captured by a linear classifier?

Disjunctions: $y = x_j \vee \neg x_k$

Monotone disjunctions: no literal (x_i) is negated

Linear classifier: $f(\mathbf{x}) = 1 \text{ iff } \sum_i w_i x_i \geq \theta$

Disjunctions with a linear classifier:

$w_j = 1$, $w_k = -1$ (x_k is negated), all other $w_i = 0$

$$f(\mathbf{x}) = 1 \text{ iff } \sum_i w_i x_i \geq 1$$

Which Boolean functions can be captured by a linear classifier?

At least *m* of *n*

y = at least 2 of $(x_j, x_k x_l)$

At least *m* of *n* with a linear classifier:

$w_j = 1$, $w_k = 1$, $w_l = 1$, and all other $w_k = 0$

$f(\mathbf{x}) = 1$ iff $\sum_i w_i x_i \geq 2$

CS446 Introduction to Machine Learning (Fall 2013)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURE 9 - 11:
DUALS, KERNELS, LARGE MARGIN CLASSIFIERS

Prof. Julia Hockenmaier
juliahmr@illinois.edu

Dual representation

Recall the Perceptron update rule:

If $\mathbf{x_m}$ *is misclassified, add* $y_m \cdot \mathbf{x_m}$ to $\mathbf{w}$

if $y_m \cdot f(\mathbf{x_m}) = y_m \cdot \mathbf{w} \cdot \mathbf{x_m} < 0$:

$\mathbf{w} := \mathbf{w} + y_m \cdot \mathbf{x_m}$

Dual representation:

Write $\mathbf{w}$ as a weighted sum of training items:

$\mathbf{w} = \sum_n \alpha_n y_n \mathbf{x}_n$

α_n: how often was $\mathbf{x}_n$ misclassified?

$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_n \alpha_n y_n \mathbf{x}_n \cdot \mathbf{x}$

Making data linearly separable

It is common for data to be not linearly separable in the original feature space.

We can often introduce new features to make the data linearly separable in the new space:

- *transform* the original features (e.g. $x \rightarrow x^2$)
- include transformed features in addition to the original features
- capture *interactions* between features (e.g. $x_3 = x_1 x_2$)

But this may blow up the number of features

The kernel trick

- Define a feature function $\varphi(\mathbf{x})$ which maps items $\mathbf{x}$ into a higher-dimensional space.
- The kernel function $K(\mathbf{x}^i, \mathbf{x}^j)$ computes the inner product between the $\varphi(\mathbf{x}^i)$ and $\varphi(\mathbf{x}^j)$

$$K(\mathbf{x}^i, \mathbf{x}^j) = \varphi(\mathbf{x}^i)\varphi(\mathbf{x}^j)$$

- Dual representation: We don't need to learn $\mathbf{w}$ in this higher-dimensional space. It is sufficient to evaluate $K(\mathbf{x}^i, \mathbf{x}^j)$

The kernel matrix

The kernel matrix of a data set D = $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ defined by a kernel function $k(\mathbf{x}, \mathbf{z}) = \varphi(\mathbf{x})\varphi(\mathbf{z})$ is the $n \times n$ matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

You'll also find the term 'Gram matrix' used:

- The Gram matrix of a set of n vectors S = $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ is the $n \times n$ matrix $\mathbf{G}$ with $\mathbf{G}_{ij} = \mathbf{x}_i\mathbf{x}_j$
- The kernel matrix is the Gram matrix of $\{\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_n)\}$

Polynomial kernels

- Linear kernel: $k(\mathbf{x}, \mathbf{z}) = \mathbf{xz}$

- Polynomial kernel of degree d:
  (only dth-order interactions):
  $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz})^d$

- Polynomial kernel up to degree d:
  (all interactions of order d or lower:
  $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz} + c)^d$ with $c > 0$

Kernels over (finite) sets

X, Z: subsets of a finite set D with |D| elements

$k(X, Z) = |X \cap Z|$ (the number of elements in X and Z) is a valid kernel:

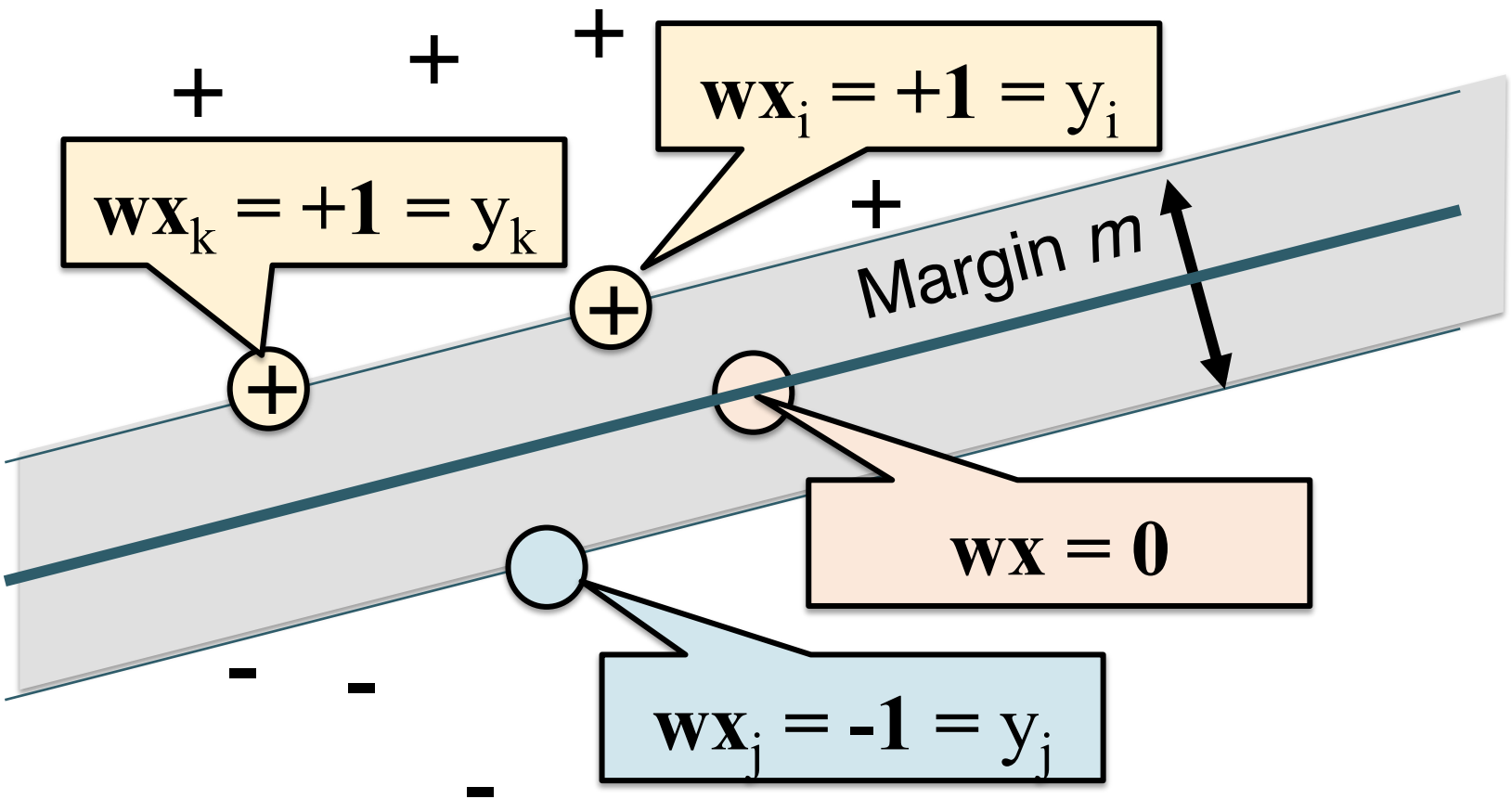
$k(X, Z) = \varphi(X)\varphi(Z)$ where $\varphi(X)$ maps X to a bit vector of length |D| (*i*th bit: does X contains the *i*-th element of D?).

$k(X, Z) = 2^{|X \cap Z|}$ (the number of subsets shared by X and Z) is a valid kernel:

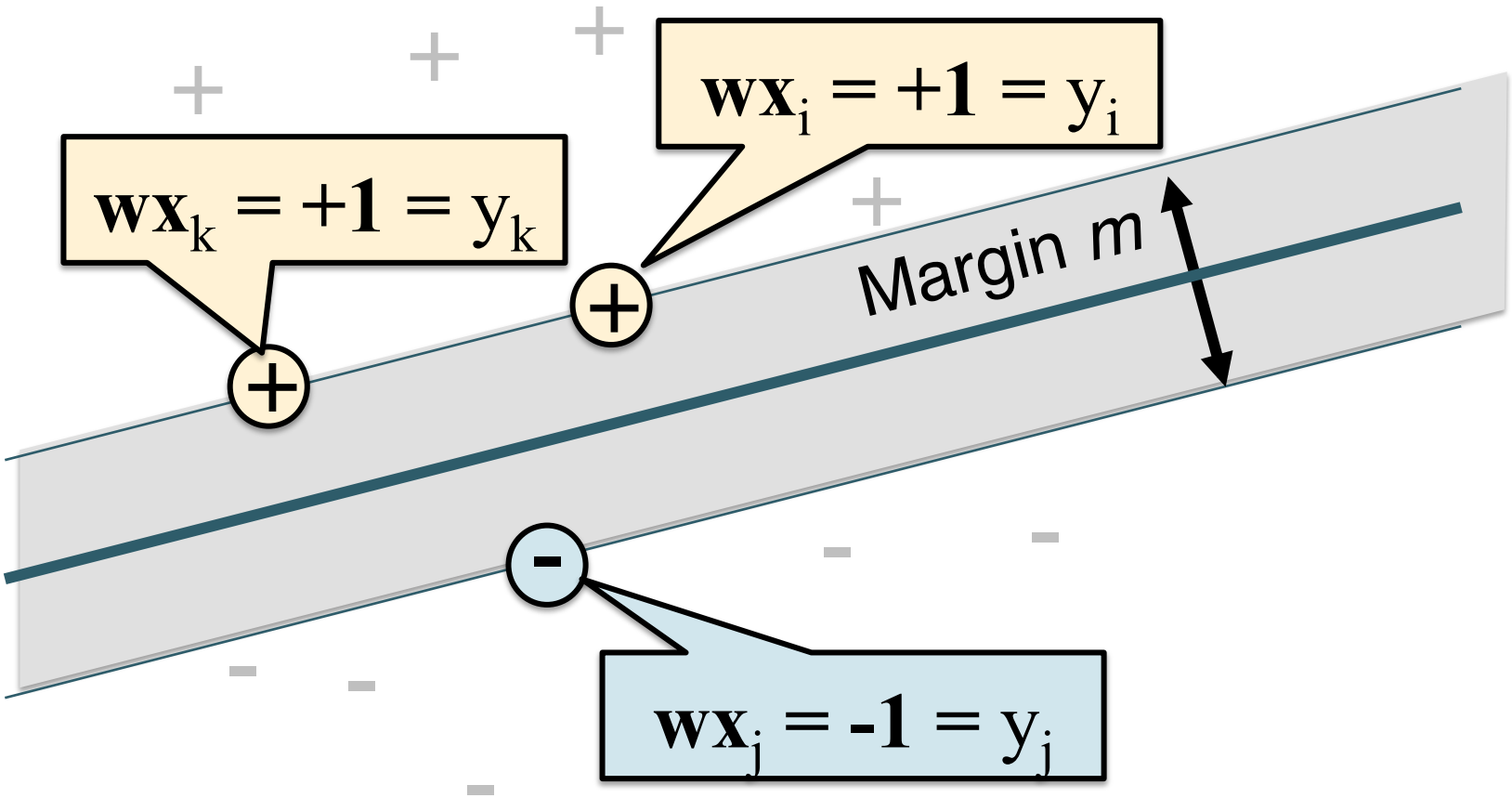
$\varphi(X)$ maps X to a bit vector of length $2^{|D|}$

(*i*-th bit: does X contains the *i*-th subset of D?)

The maximum margin decision boundary



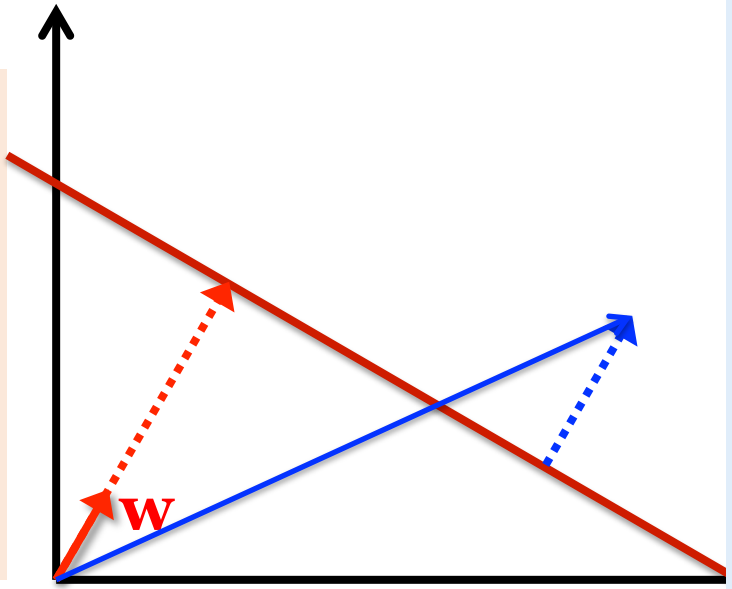
Support vectors



Margins

Distance of hyperplane $\mathbf{wx} + b = 0$ to origin:

$$\frac{-b}{\|\mathbf{w}\|}$$



Absolute distance of point $\mathbf{x}$ to hyperplane $\mathbf{wx} + b = 0$:

$$\frac{|\mathbf{wx} + b|}{\|\mathbf{w}\|}$$

Decision boundary:

Hyperplane with $f(\mathbf{x}) = 0$

i.e. $\mathbf{wx} + b = 0$

Rescaling $\mathbf{w}$ and b

- Rescaling $\mathbf{w}$ and b by a factor k changes the functional margin γ by a factor k:

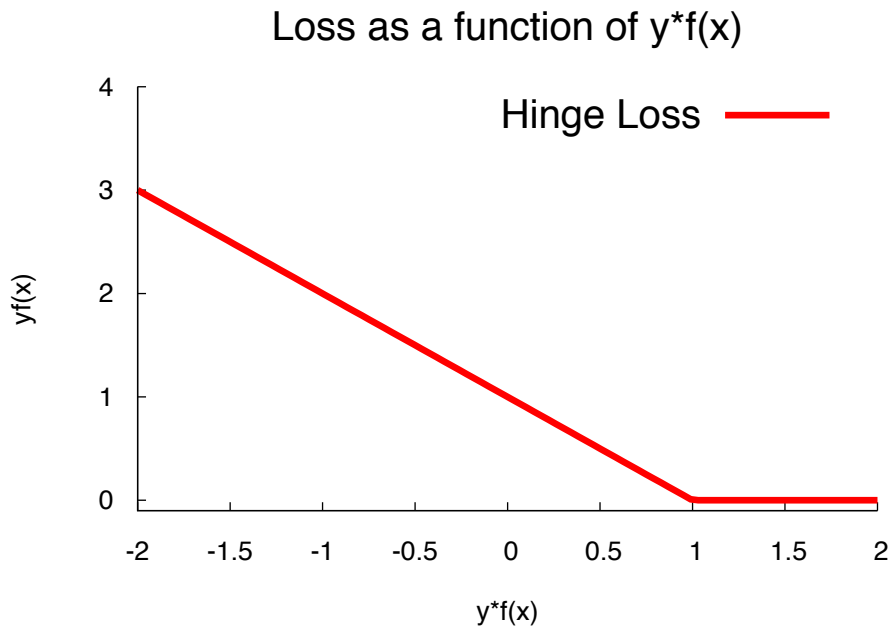
$$\gamma = y^{(i)} (\mathbf{w}\mathbf{x}^{(i)} + b)$$

$$k\gamma = y^{(i)} (k\mathbf{w}\mathbf{x}^{(i)} + kb)$$

- The point that is closest to the decision boundary has functional margin γ_{min}
- $\mathbf{w}$ and b can be rescaled so that $\gamma_{min} = 1$
- When learning $\mathbf{w}$ and b, we can set $\gamma_{min} = 1$ (and still get the same decision boundary)

Hinge loss

$$L(y, f(\mathbf{x})) = \max(0, 1 - yf(\mathbf{x}))$$



Support Vector Machines

Learn **w** in an SVM = maximize the margin:

$$\underset{\mathbf{w},\, b}{\operatorname{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \min_{n} \left[y^{(n)} (\mathbf{w}\mathbf{x} + b) \right] \right\}$$

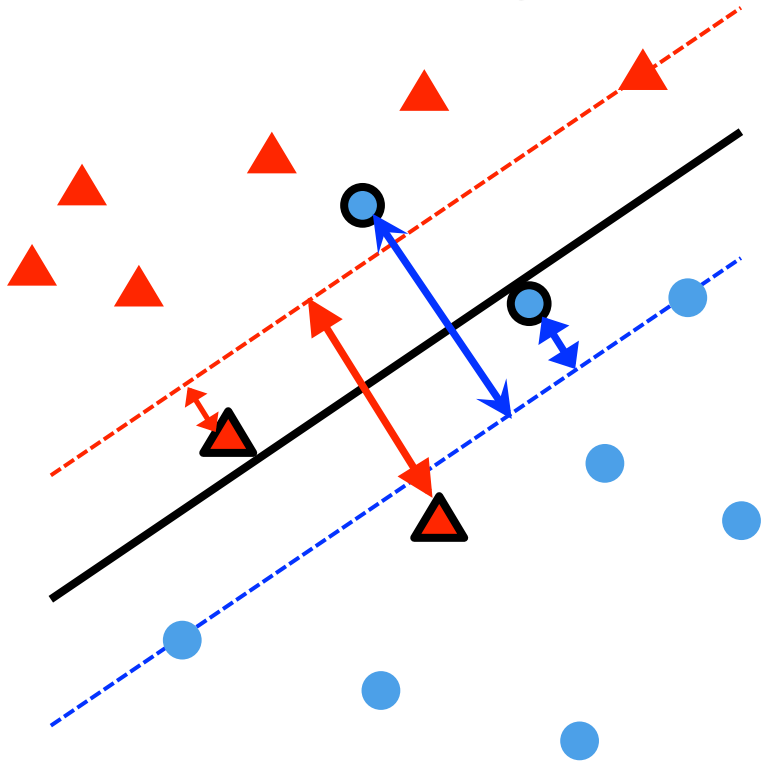
Easier equivalent problem: a quadratic program

- Setting $\min_n(y^{(n)}(\mathbf{w}\mathbf{x}^{(n)} + b) = 1$ implies $(y^{(n)}(\mathbf{w}\mathbf{x}^{(n)} + b) \geq 1$ for all n
- argmax(1/**ww**)= argmin(**ww**) = argmin(1/2·**ww**)

$$\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2} \mathbf{w} \cdot \mathbf{w}$$
$$subject\ \ to$$
$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \ \ \forall i$$

Dealing with outliers: Slack variables ξ_i

ξ_i measures by how much example $(\mathbf{x}_i, y_i)$ fails to achieve margin δ



Soft margins

$$\underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$

$$subject\ to$$

$$\xi_i \geq 0 \ \ \forall i$$

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq (1-\xi_i)\forall i$$

ξ_i (slack): how far off is $\mathbf{x}_i$ from the margin?

C (cost): how much do we have to pay for misclassifying $\mathbf{x}_i$

We want to minimize $C\sum_i \xi_i$ and maximize the margin

C controls the tradeoff between margin and training error