Lecture 3:

Overview
Fall 2018

Kai-Wei Chang

CS @ UCLA

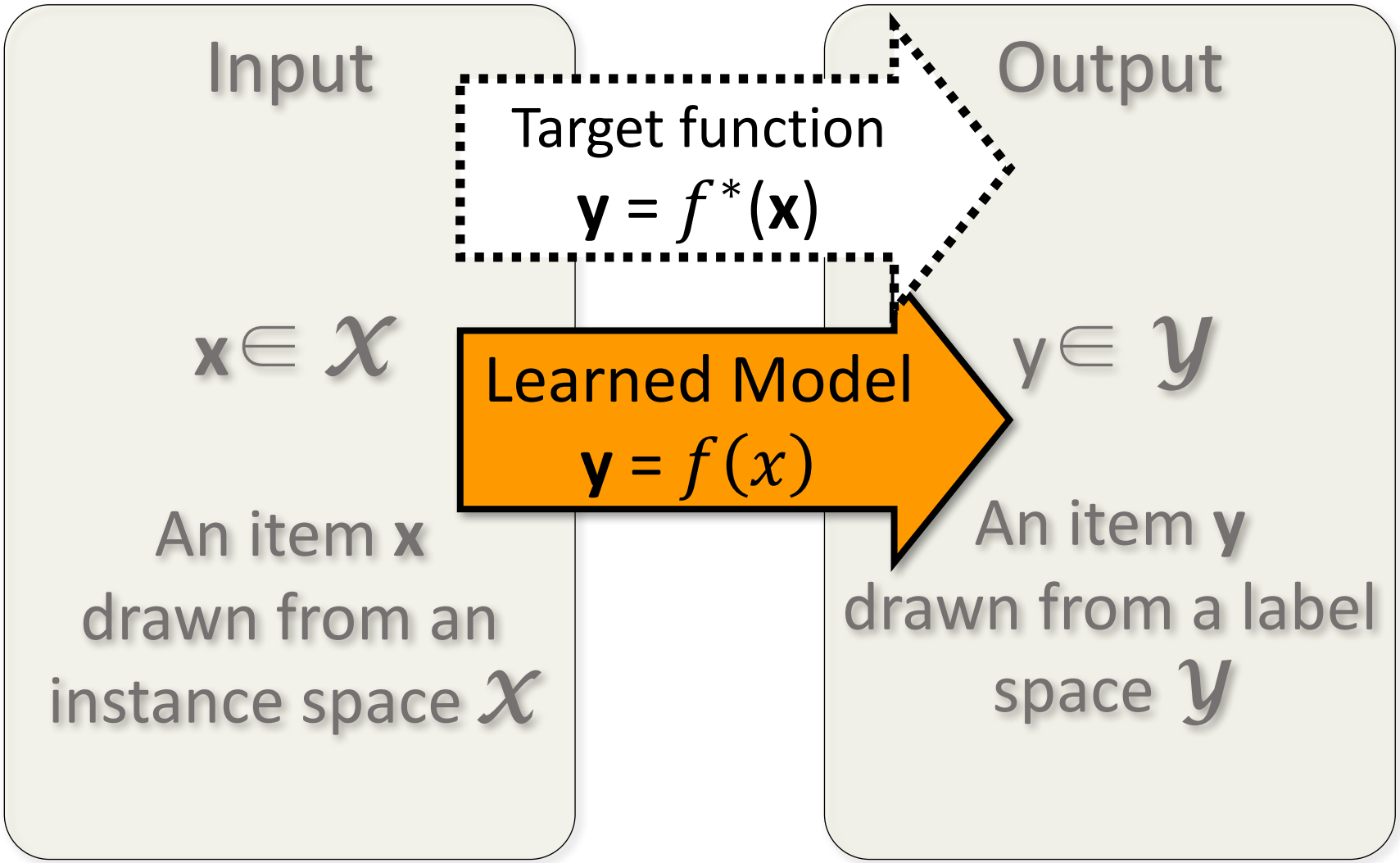
kw+cm146@kwchang.net

The instructor gratefully acknowledges Eric Eaton (UPenn), who assembled the original slides, Jessica Wu (Harvey Mudd), David Kauchak (Pomona), Dan Roth (Upenn), Sriram Sankararaman (UCLA), whose slides are also heavily used, and the many others who made their course materials freely available online.

What is Learning

- The Badges Game……
  - This is an example of the key learning protocol: supervised learning
- Issues:
  - Prediction or Modeling?
  - Representation
  - Problem setting
  - Background Knowledge
  - When did learning take place?
  - Algorithm

Learning the mapping



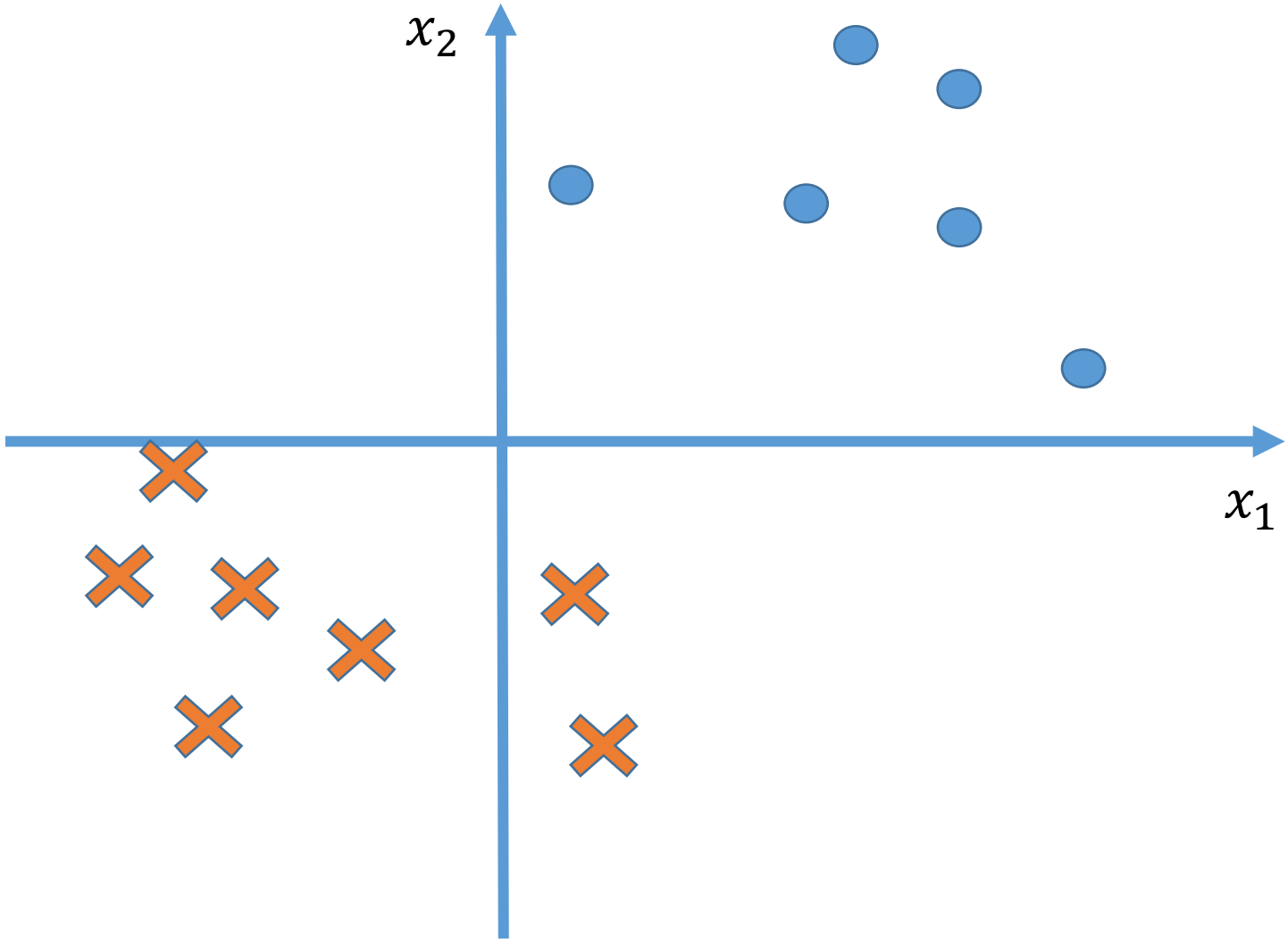
Using supervised learning

- 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?

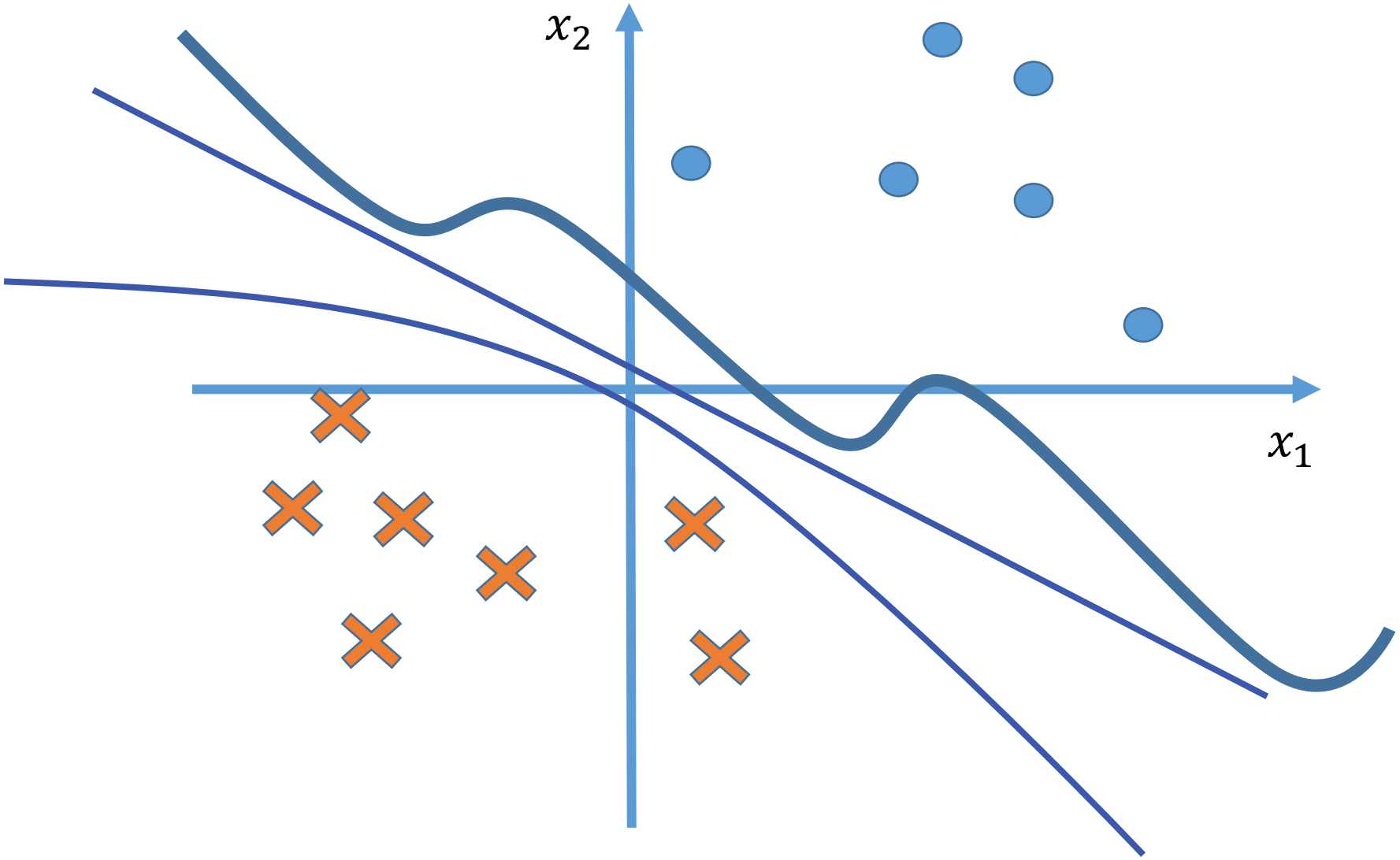
Views of Learning

- 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 guess of the hypothesis space could be wrong
    - y=x4 Λ one-of (x1, x3) is also consistent

Example problem

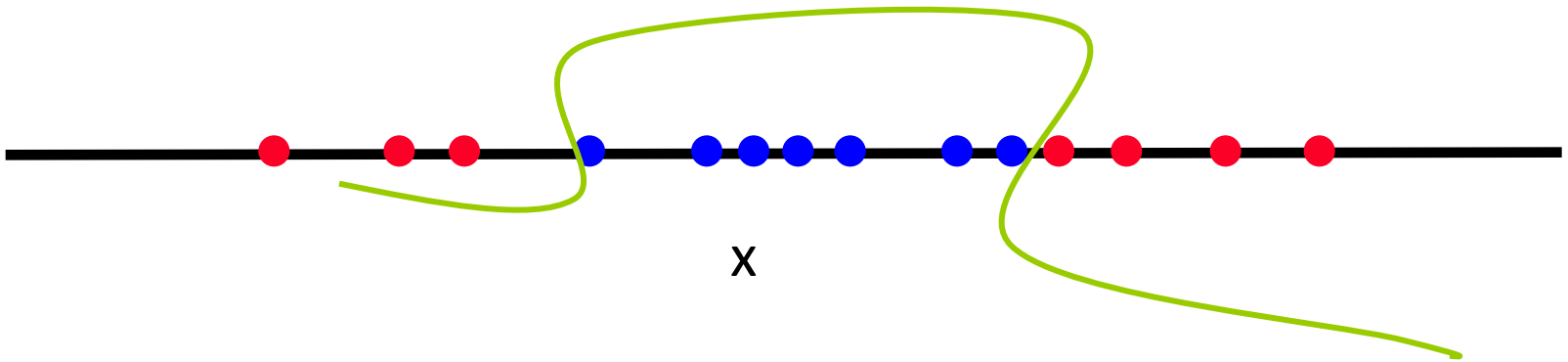


Hypothesis space:



Functions Can be Made Linear

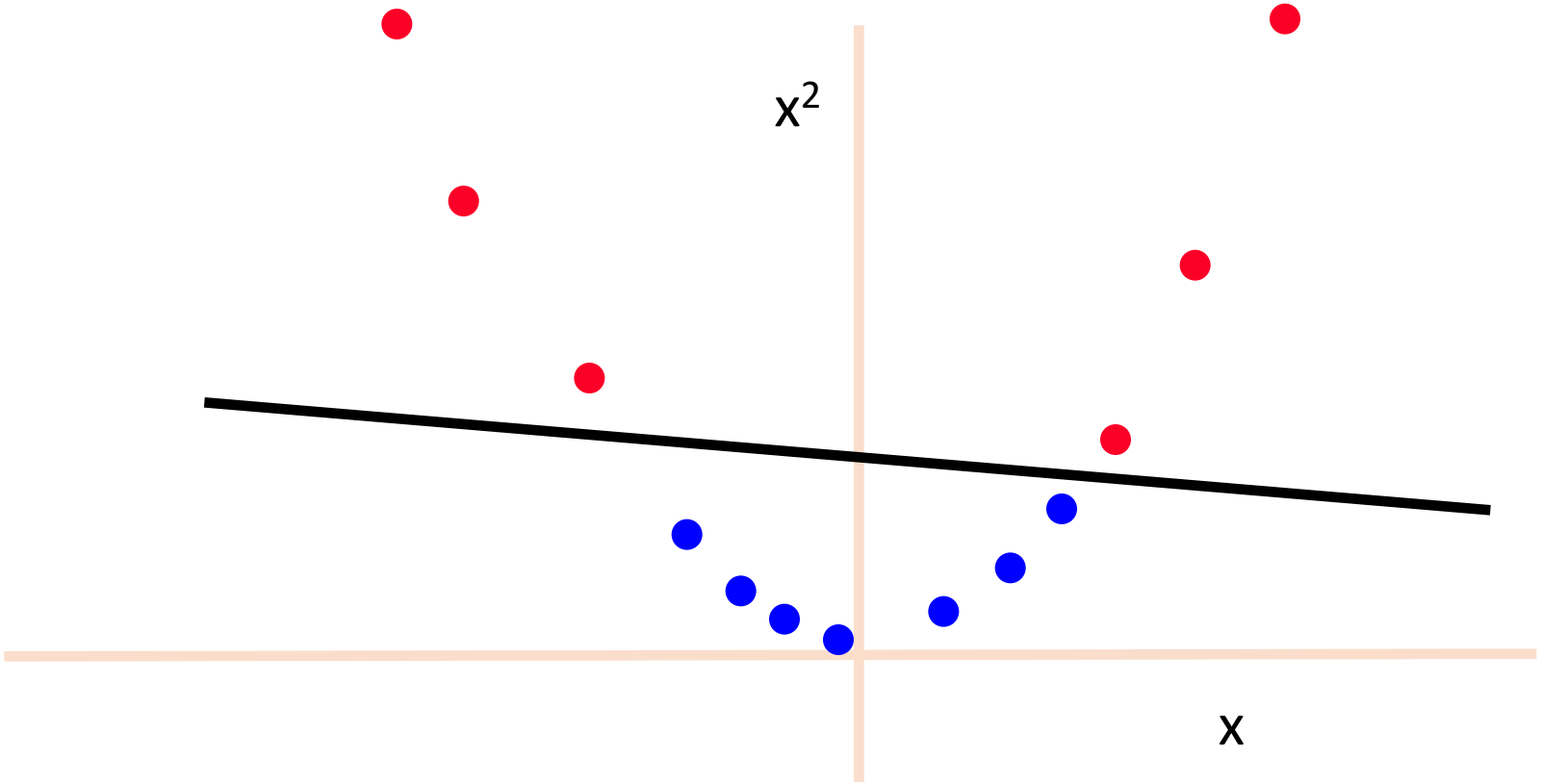
- Data are not linearly separable in one dimension
- Not separable if you insist on using a specific class of functions



Can we do some mapping to make it linear spreadable?

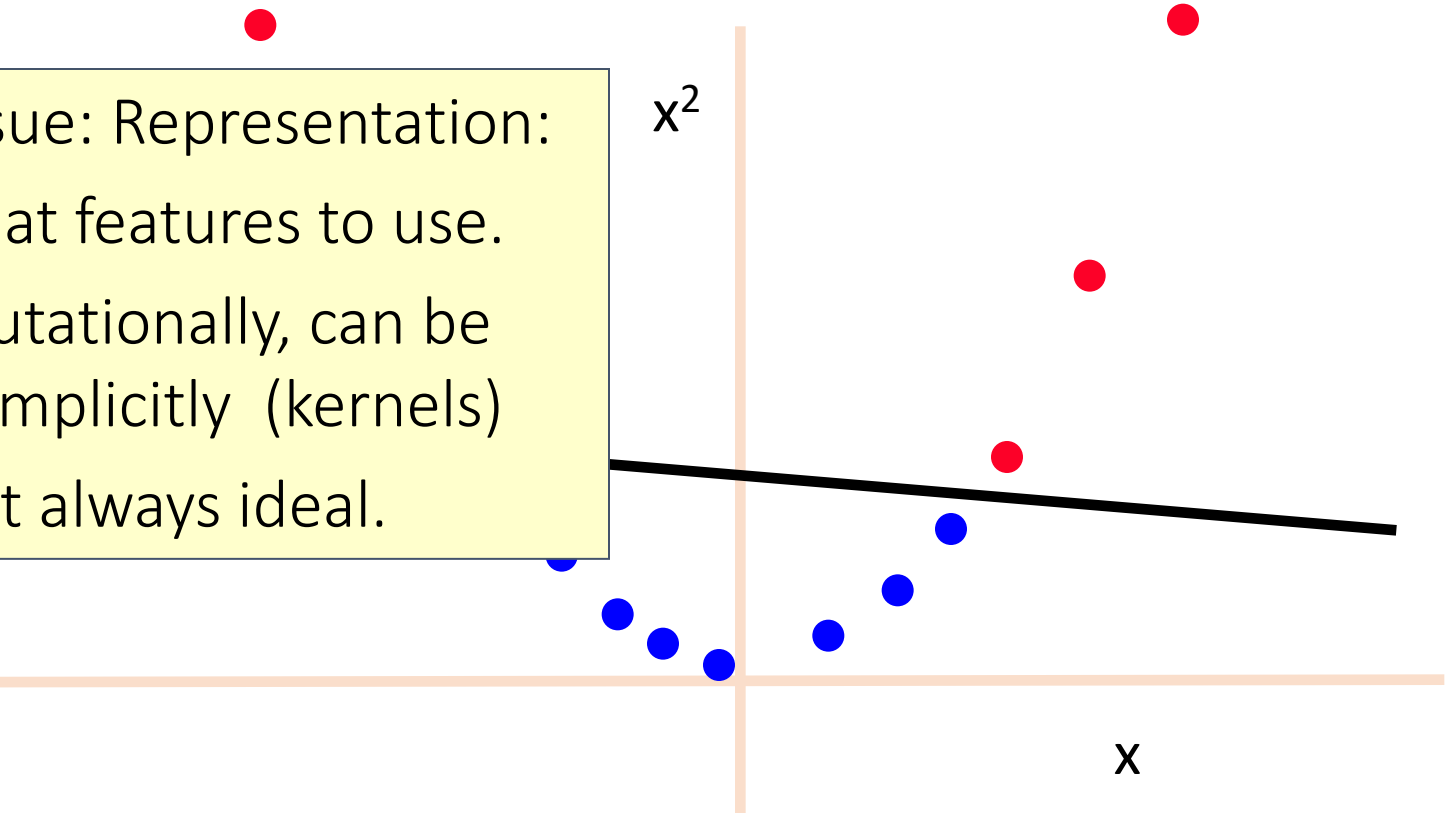
Blown Up Feature Space

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



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.

How to learn?

How can we find a good model from the hypothesis space?

Recap: rule-out

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

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"

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

variables	1-of	2-of	3-of	4-of
{X1}				
{X2}				
{X3}				
{X4}				
{X1,X2}				
{X1, X3}				
{X1, X4}				
{X2,X3}	2	3	-	-

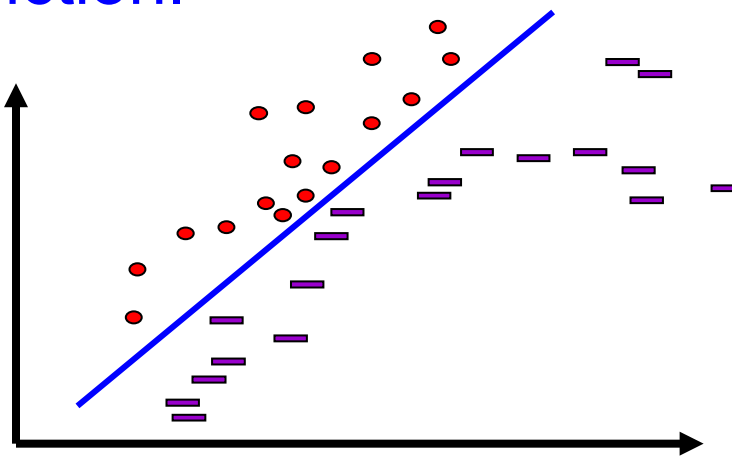
variables	1-of	2-of	3-of	4-of
{X1,X2,X3,X4}				3

Learning is the removal of our remaining uncertainty

Found a consistent hypothesis.

How about linear function?

- Challenges
  - The hypothesis space contains infinite # functions
  - Several functions are consistent with the data
- A possibility: Local search
  - Start with a linear threshold function.
  - See how well you are doing.
  - Correct
  - Repeat until you converge.



Search can be done in a smarter way

- There are other ways that do not search directly in the hypotheses space
- Directly compute the hypothesis by optimizing an objective

- Key question: what is a good classifier?

General Framework for Learning

Problem Setting

- Set of possible instances X
- Set of possible labels Y
- Unknown target function f : X ➔ Y
- Set of function hypotheses $H = \{h \mid h : X \rightarrow Y\}$

Input: Training instances drawn from data generating distribution p

$$\{(x_i, y_i)\}_{i=1}^{n} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$$

Output: Hypothesis h in H that best approximates f

Learning Problem

Output: Hypothesis *h* in *H* that best approximates *f*

h should do well (as measured by the loss) on future instances

Formally, *h* should have low expected (test) loss/Risk

$$\mathbb{E}_{(x,y)\sim p}\left[L(y,h(x))\right] = \sum_{x,y} p(x,y)L(y,h(x))$$

Problem?

We don't know what *p* is

But we are given samples drawn from *p*

Slide credit: Eric Eaton
Based on slide by Tom Mitchell

Learning Problem

We instead approximate the risk by the training error/ empirical risk

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))$$

When is this reasonable ?

Both the training data **and** the test set are generated based on this distribution

$D = \{x_i, y_i\}_{i=1}^{n}$ is called training data

Challenges

1. Minimizing. $\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))$ is in general a NP-hard problem

2. Can make the training error zero by memorizing if the hypothesis space is expressive.

Challenges

1. Minimizing. $\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))$ is in general a NP-hard problem
   - Think of the
   - To alleviate this computational problem, minimize a new function – a convex upper bound of the classification error function

1. Can make the training error zero by memorizing if the hypothesis space is expressive.

Algorithmic View of Learning: an Optimization Problem

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

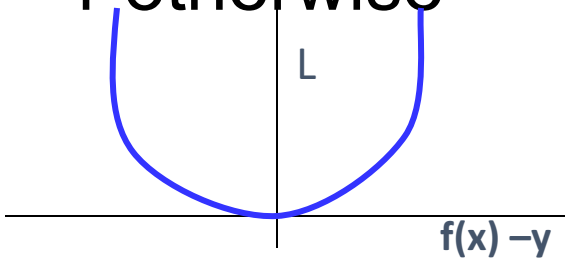
    $L(h(x),y) = 0$ if $h(x) = y$; 1 otherwise

  - Squared Loss:

    $L(h(x),y) = (h(x) - y)^2$

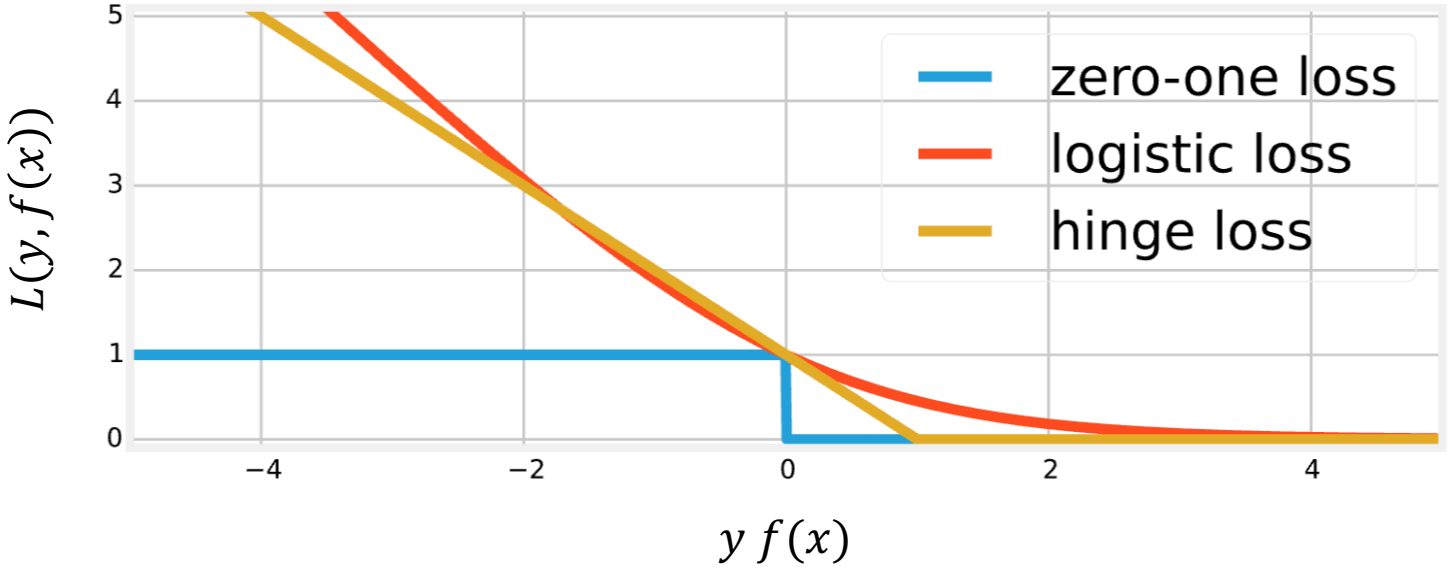
  - Input dependent loss:

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



How about the loss function?

- Usually, we cannot minimize 0-1 loss
  - It is a combinatorial optimization problem: NP-hard
- Idea: minimizing its upper-bound

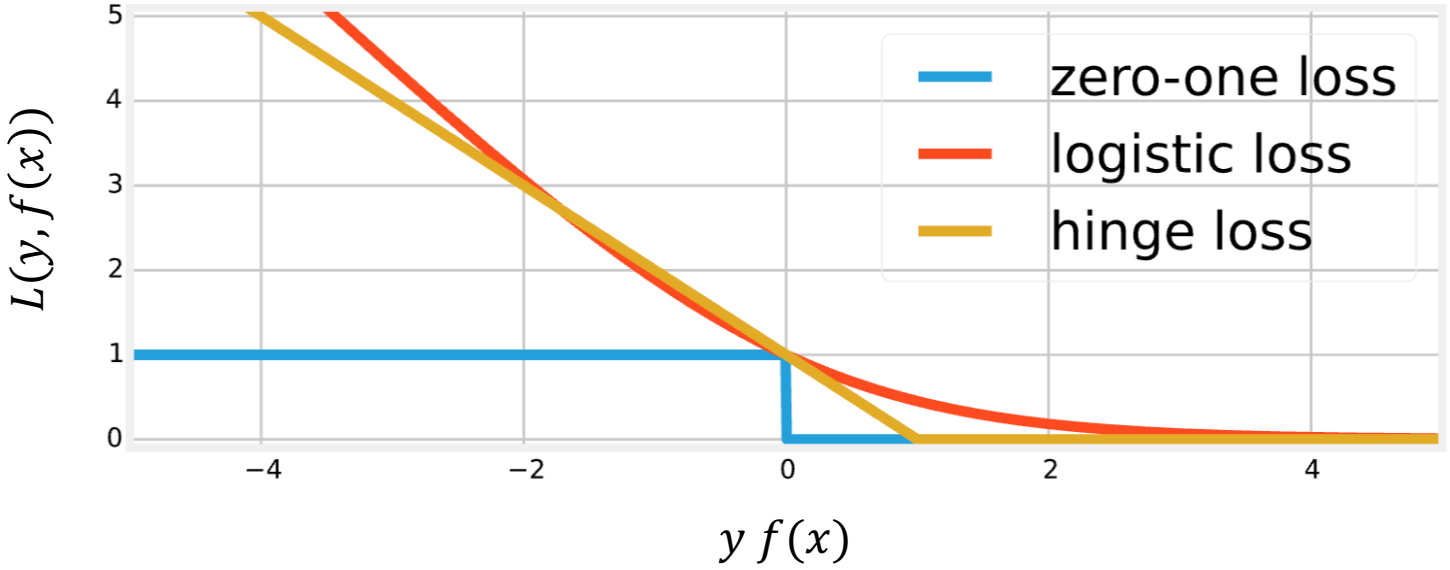


How about the loss function?

rd

How about the loss function?

- Usually, we cannot minimize 0-1 loss
  - It is a combinatorial optimization problem: NP-hard
- Idea: minimizing its upper-bound

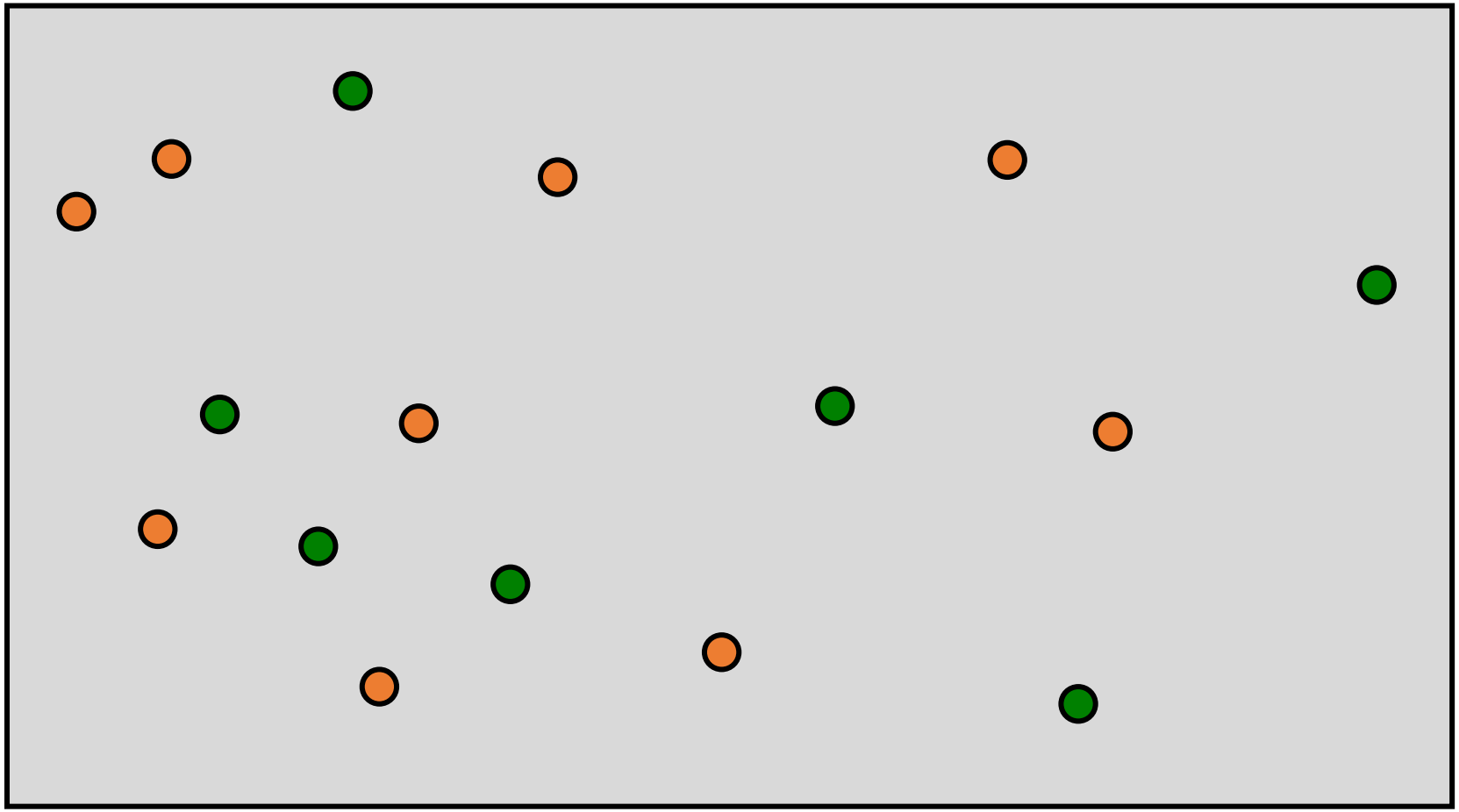


Challenges

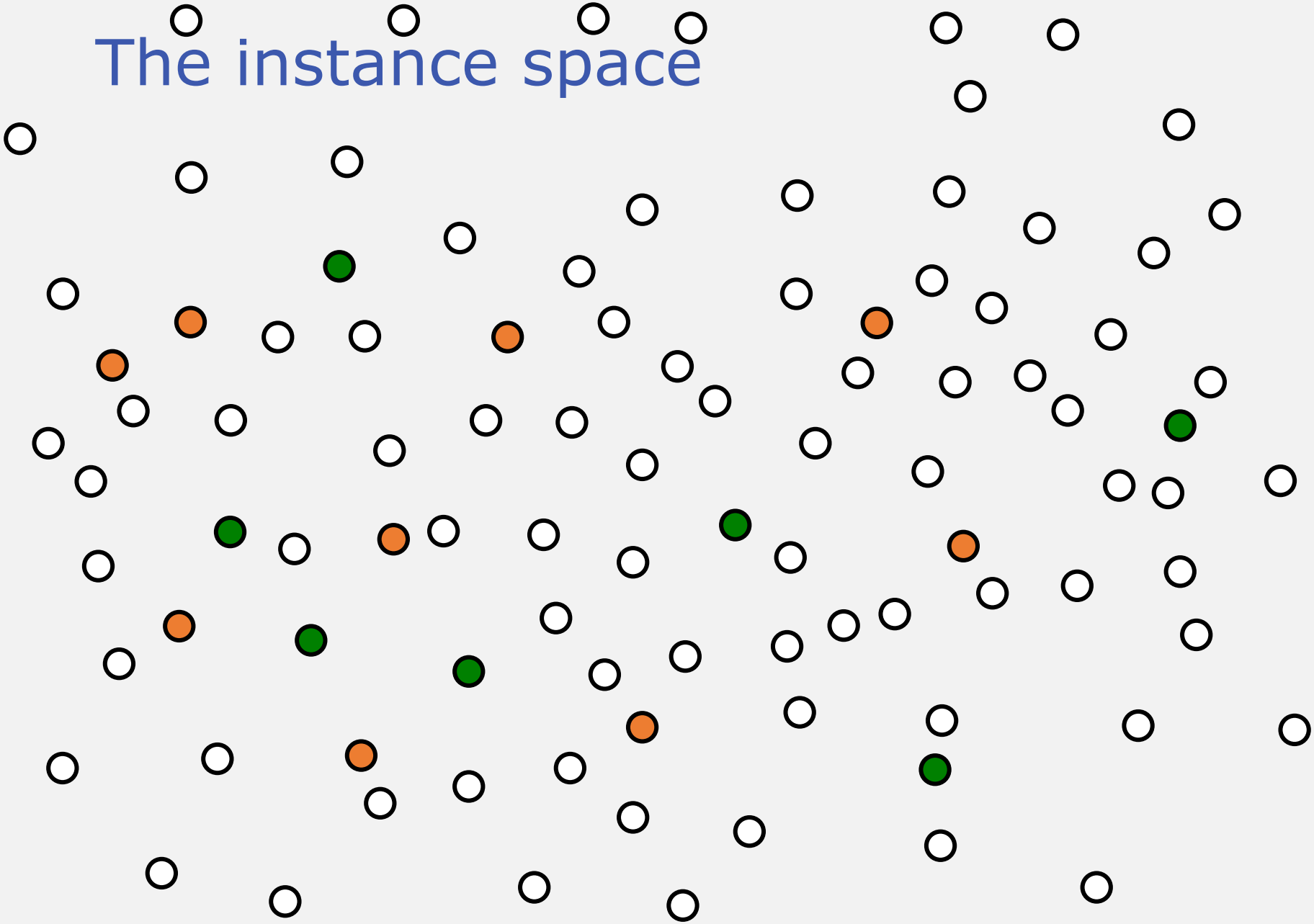
1. Minimizing. $\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i))$ is in general a NP-hard problem
   - **To alleviate this computational problem, minimize a new function – a convex upper bound of the classification error function**

2. Can make the training error zero by memorizing if the hypothesis space is expressive.

Our training data

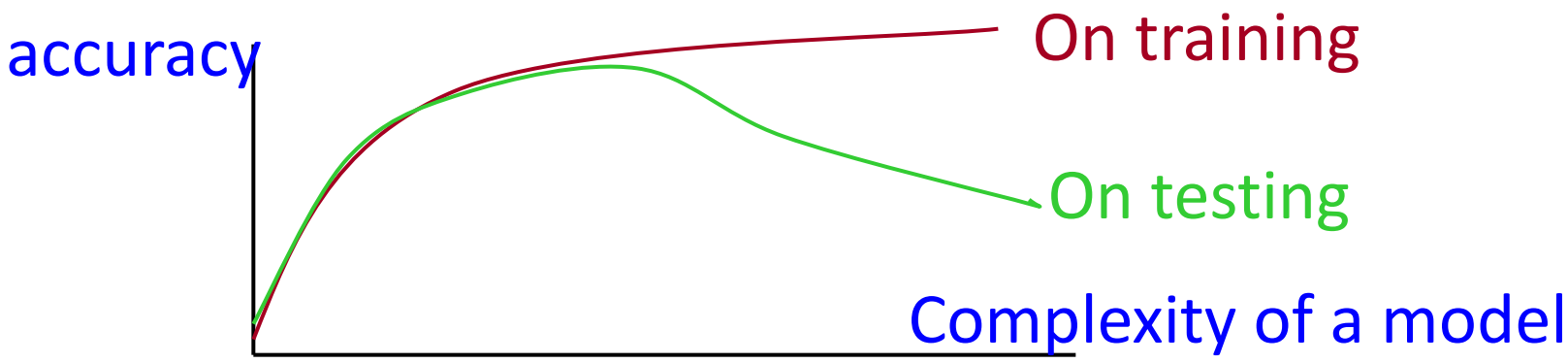


The instance space



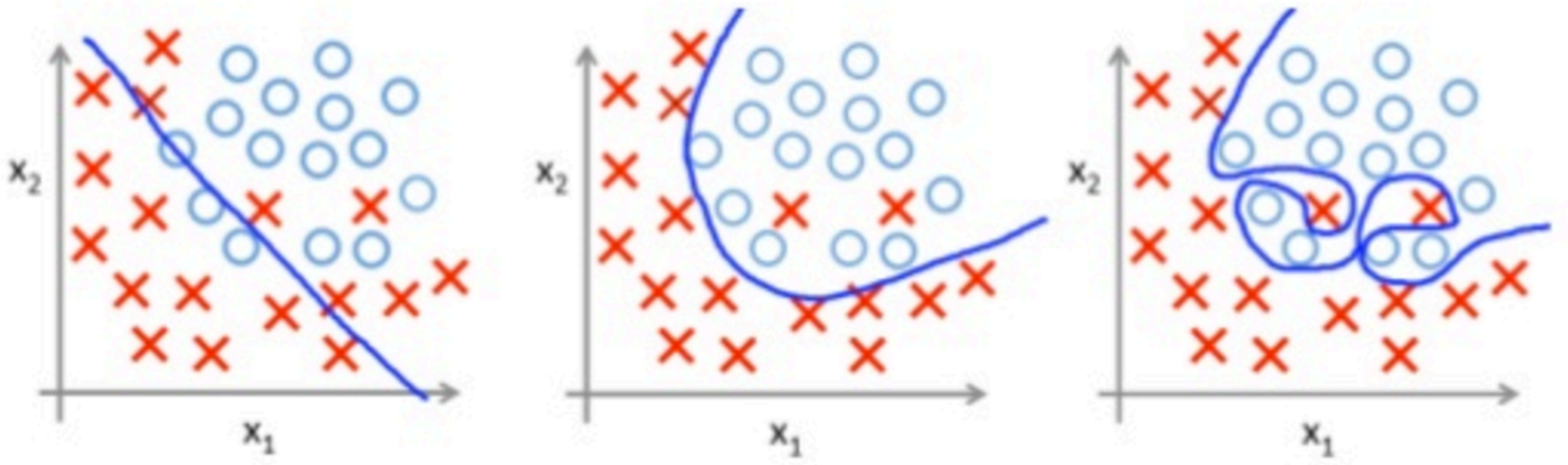
Overfitting the Data

- Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization performance.
  - There may be noise in the training data
  - The algorithm might be making decisions based on very little data



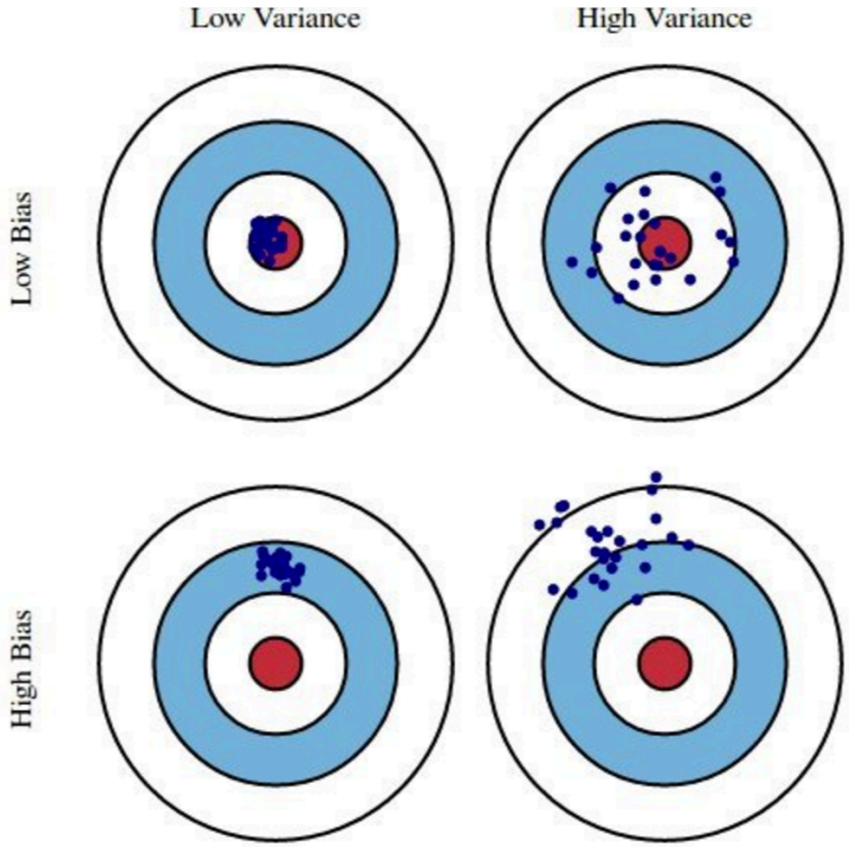
Under-fitting and over-fitting

- Which classifier (blue line) is the best one?



Bias V.S. Variance

- Remember, training data are subsamples drawn from the true distribution
- Exam strategy:
  - Study every chapter well
    - A+: Low var & bias
  - Study only a few chapters
    - A+? B? C? Low bias; High var
  - Study every chapter roughly
    - B+: Low var; high bias
  - Go to sleep
    - B ~D: High var, high bias



Prevent overfitting

- Using a less-expressive model
  - E.g., linear model
- Adding regularization
  - Promote simper models
- Data perturbation
  - Make the model more robust
  - Can be done algorithmically (e.g., dropout)
- Stop the optimization process earlier
  - Sounds bad in theory; but works in practice.

Machine Learning Practice

Putting it all together:

Develop ML algorithms

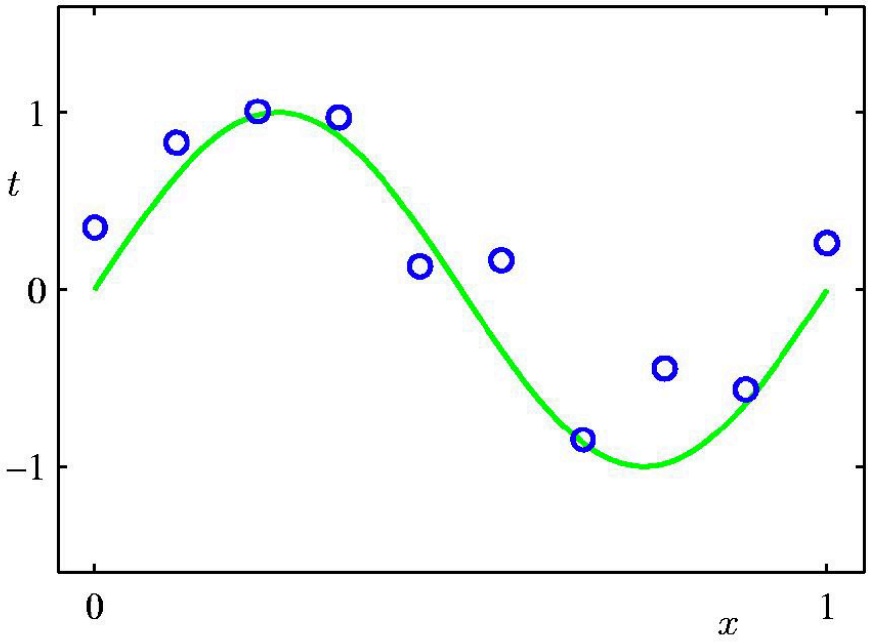
- Step 1: Collect data
- Step 2: Design your algorithm
- Step 3: Verify your algorithm; tune hyper-parameters
  - Option 1: Split data to Train/Dev
  - Option 2: use Cross-Validation
- Step 4: Retrain the model using the best parameter on the whole dataset
- Step 5: Deploy your model on real test data

Example: Regression Problem

- Consider simple regression dataset
  - $f : X \rightarrow Y$
  - $x \in R$
  - $y \in R$

- Question 1: How should we pick the hypothesis space H ?
- Question 2: How do we find the best h in this space?

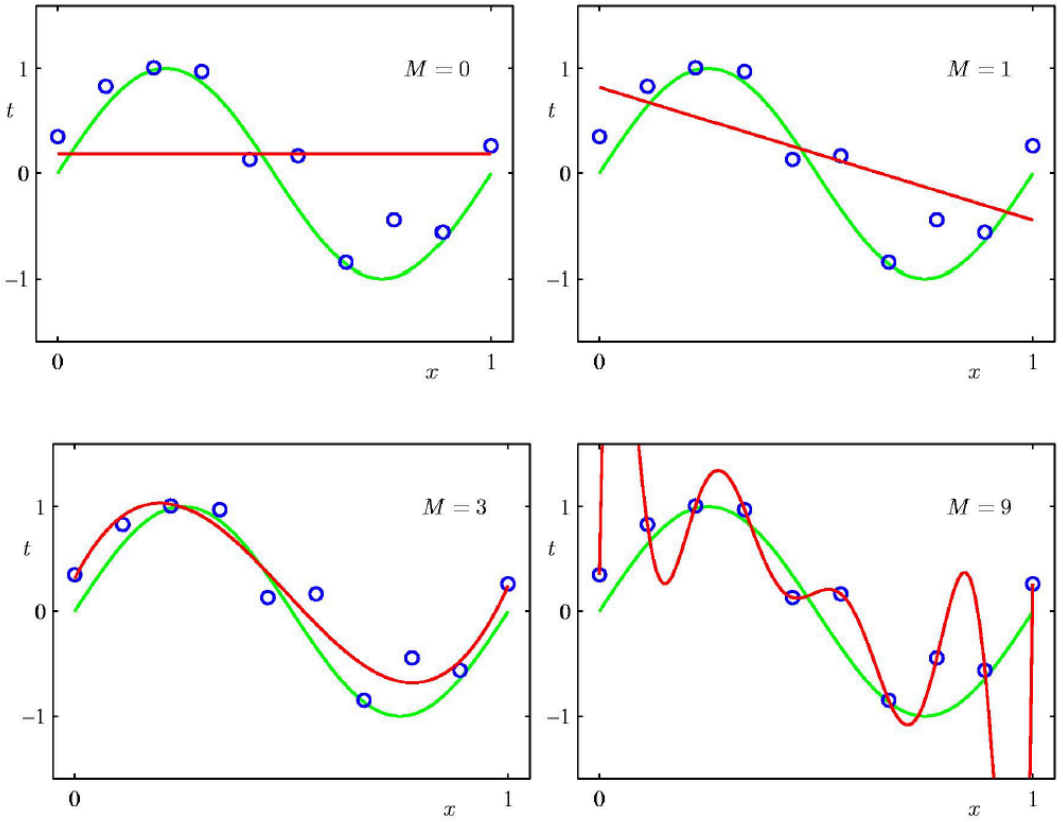
Dataset: 10 points generated from sin function with noise



Based on slide by David Sontag
Images from Bishop [PRML]

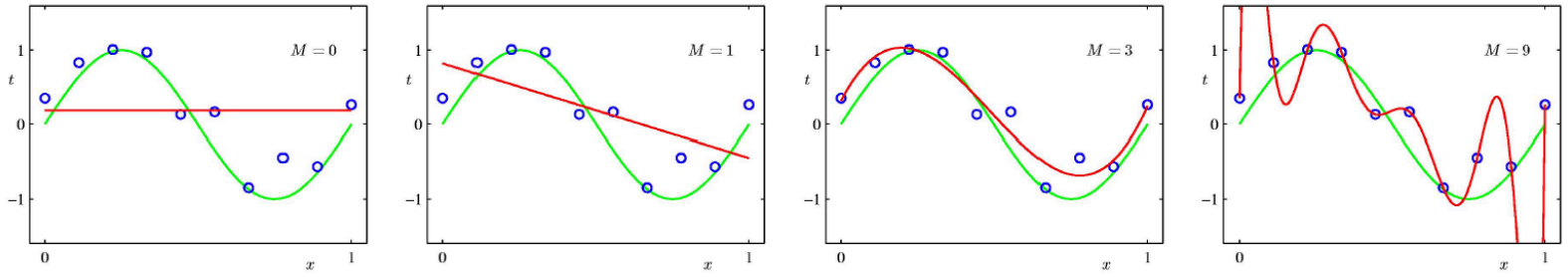
Hypothesis Space: Degree-M Polynomials

- Infinitely many hypotheses
- Which one is **best**?



Based on slide by David Sontag
Images from Bishop [PRML]

Hypothesis Space: Degree-M Polynomials

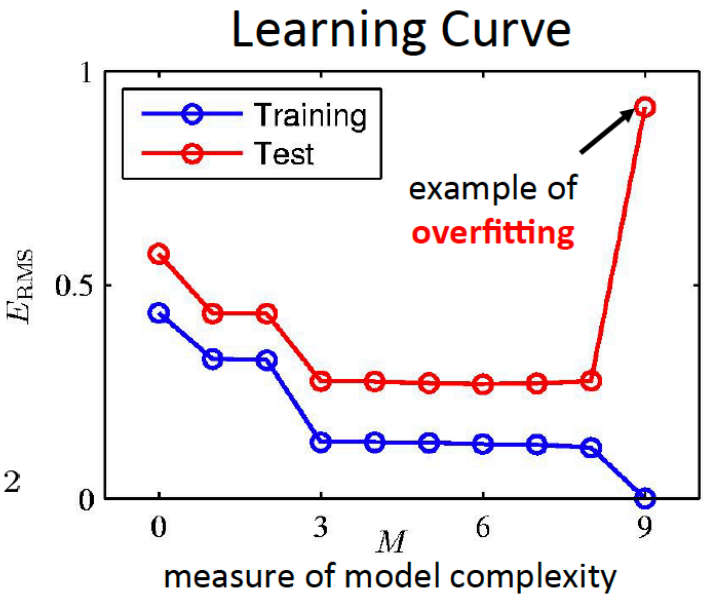


- For regression, common choice is squared loss

$$L(y_i, h(x_i)) = (y_i - h(x_i))^2$$

- *Empirical loss* of function *h* applied to training data is then

$$\frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i)) = \frac{1}{n}\sum_{i=1}^{n} (y_i - h(x_i))^2$$

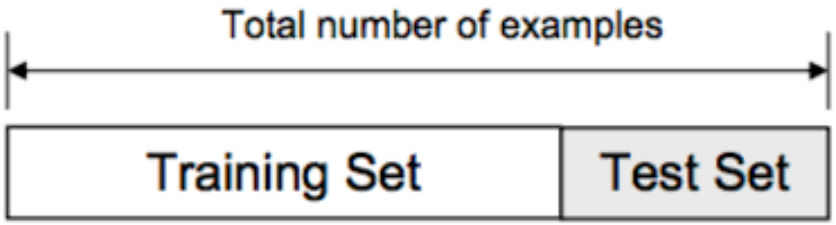


Based on slide by David Sontag
Images from Bishop [PRML]

How to find the best M?

- There are two types of parameters
  - Hyper-parameters (e.g., M) need to be tuned
  - Model parameters that can be learned from the training set
- M is a hyper-parameter for the regression model
- We can try out different M and see which one work
- How to?

Train/Dev splits



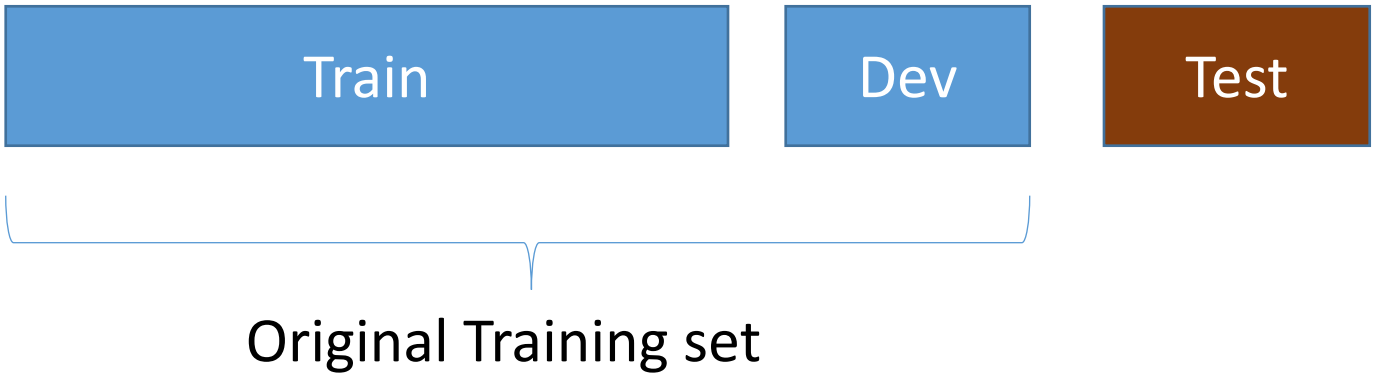
- Usually a data set consists of a training set (for learning) and a test set (for evaluation)

- You need to report performance on test data, but you are not allowed to look at it.
  - Why?
  - You are allowed to look at the development data (and use it to tweak parameters)

Train/Dev splits

Dev is like mock exam

- Further split your training data into two sets:
  - Training data (often 70-90%)
  - Development data (10-20%)



Trade off in Train/Dev splits

- Large Train, small Dev
  (e.g., #train = 48, #dev = 2)

Result on dev is not representative

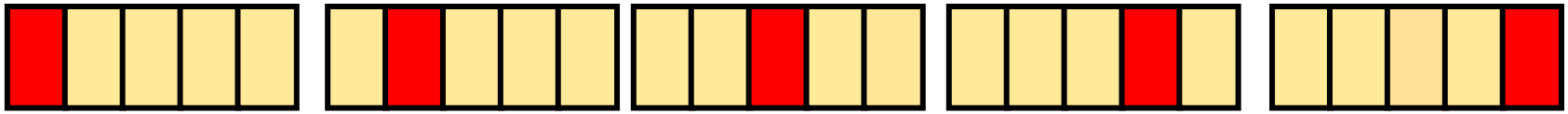
- Small Train, Large Dev
  (e.g., #train = 30, #dev = 20)

May not have enough data to train a model

N-fold cross validation (useful esp. when training set is small)

- Instead of a single test-training split:

- Split data into N equal-sized parts

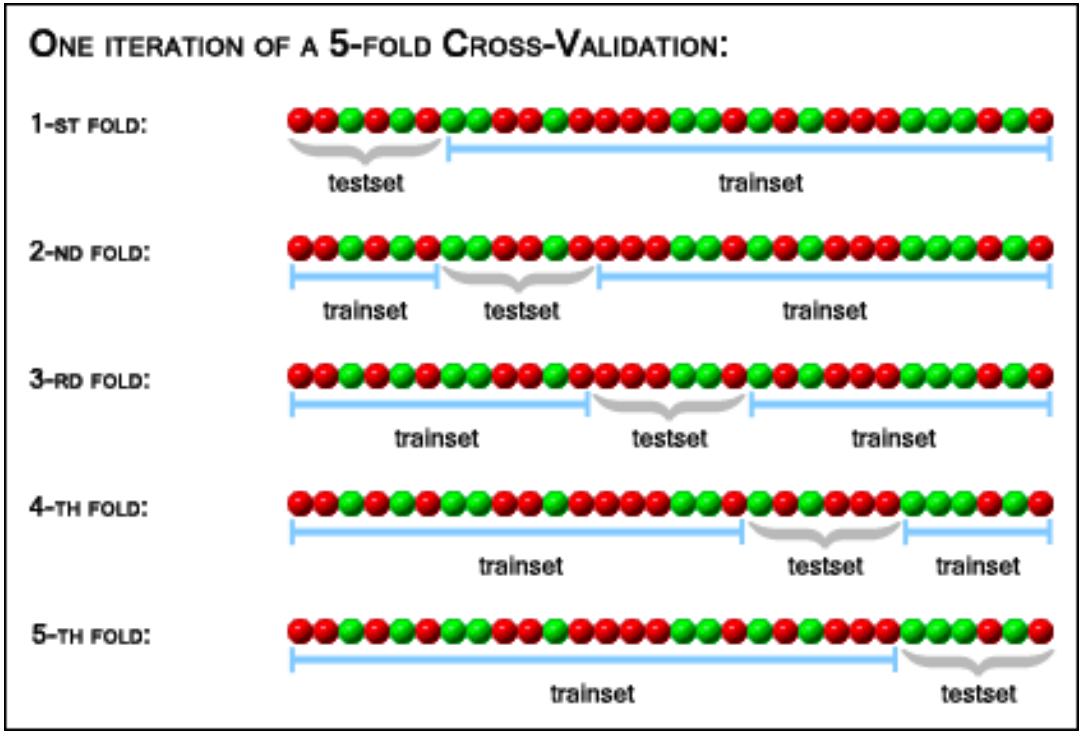


- Train and test N different classifiers
- Report average accuracy and standard deviation of the accuracy to choose parameters

Finding parameters based on cross validation

- Given D^{TRAIN} and D^{TEST}, for each possible value of the hyper-parameter (e.g., M = 1,2, 3,…,10)
  - Conduct cross validation on D^{TRAIN} with parameter K
- Choose the model parameter with the best cross validation performance
- Re-train the model on D^{TRAIN} with the best parameter set
- Evaluate the model on D^{TEST}

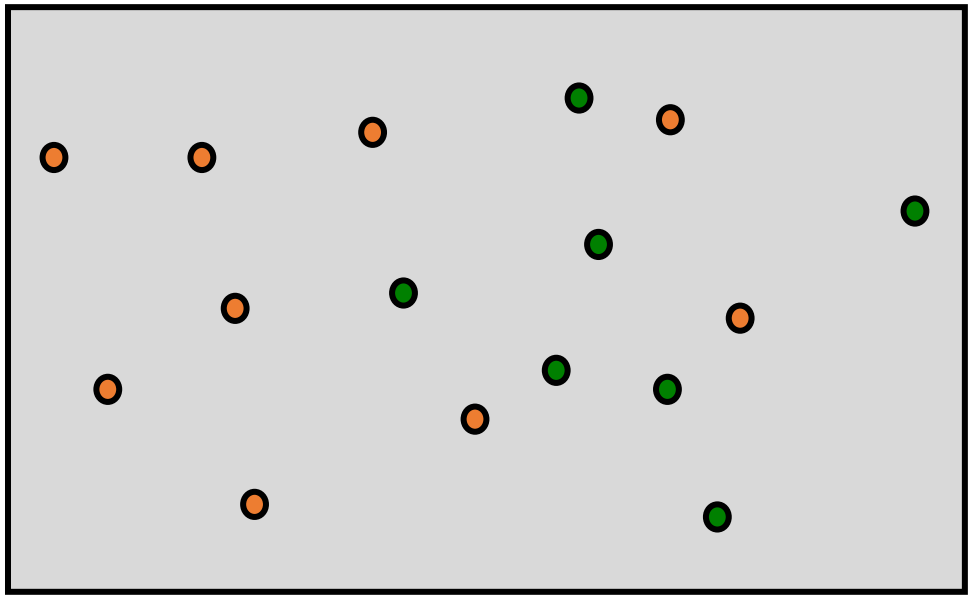
Example



	Parameter 1	Parameter 2
	Accuracy: 100%	Accuracy: 100%
	Accuracy: 50%	Accuracy: 100%
	Accuracy: 50%	Accuracy: 100%
	Accuracy: 100%	Accuracy: 100%
	Accuracy: 100%	Accuracy: 50%
	Avg Accuracy: 80%	Avg Accuracy: 90%

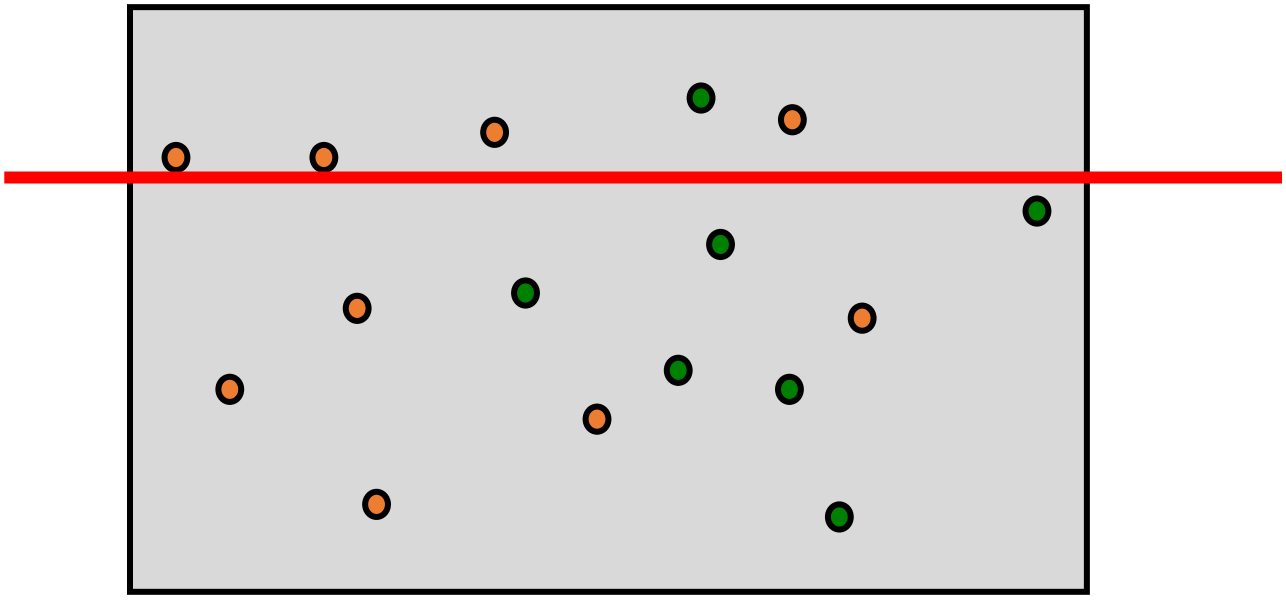
In class-practice

- Consider two class classifiers:
  - Vertical line; Horizontal line
  - Which one is better on the following dataset?



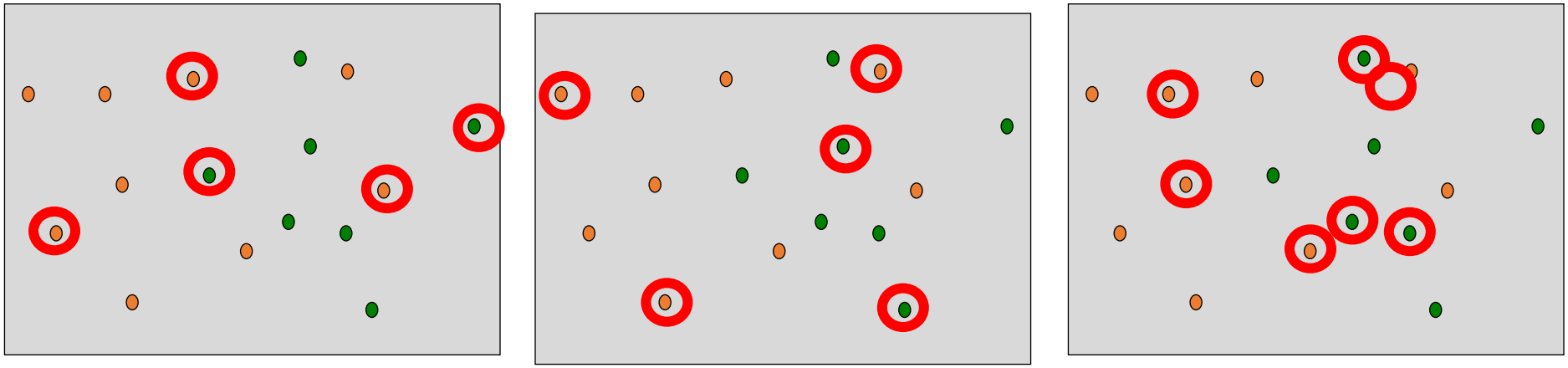
In class-practice

- Consider two class classifiers:
  - Vertical line; Horizontal line
  - Which one is better on the following dataset?



In class-practice

- Consider two class classifiers:
  - Vertical line; Horizontal line
  - Which one is better on the following dataset?
  - Three-fold cross validation



Machine learning libraries

- Machine learning toolbox:
  - Scikit learn (python), Weka (java), many others.
- Deep learning models
  - Pytorch, DyNet, Tensorflow
- Big data
  - Apache spark
- Many others …

Playground & Resources

- Tutorial:
  - https://pytorch.org/tutorials/
  - http://scikit-learn.org/stable/tutorial/index.html
- ML competitions:
  - https://www.kaggle.com/ (several others)
- Conferences/Papers:
  - ICML / NIPS / AAAI/ IJCAI/ ICLR/ …

Practical tricks

- Make sure you can reproduce your experiments
  - Set random seed; Keep exp log; Use code repository (e.g., Github)
  - Use configure file to manager parameters
  - Script for finding parameters
- Start with small data
  - Split a tiny dataset for debugging
  - Split a small dataset to check which parameters are important; Some hyper-parameters do not depend on size of data (in theory)