Deep Learning

Lecture 3: Machine Learning Basics II

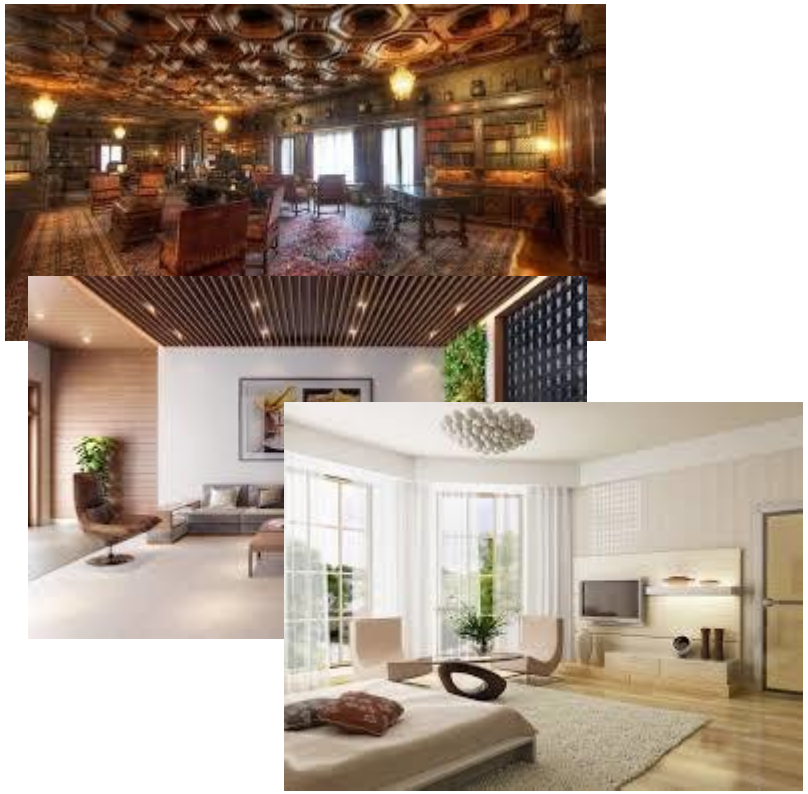
National University of Singapore

EE5934/6934

Instructor: Joey Tianyi Zhou

Linear classification

Example 1: image classification



Indoor

outdoor

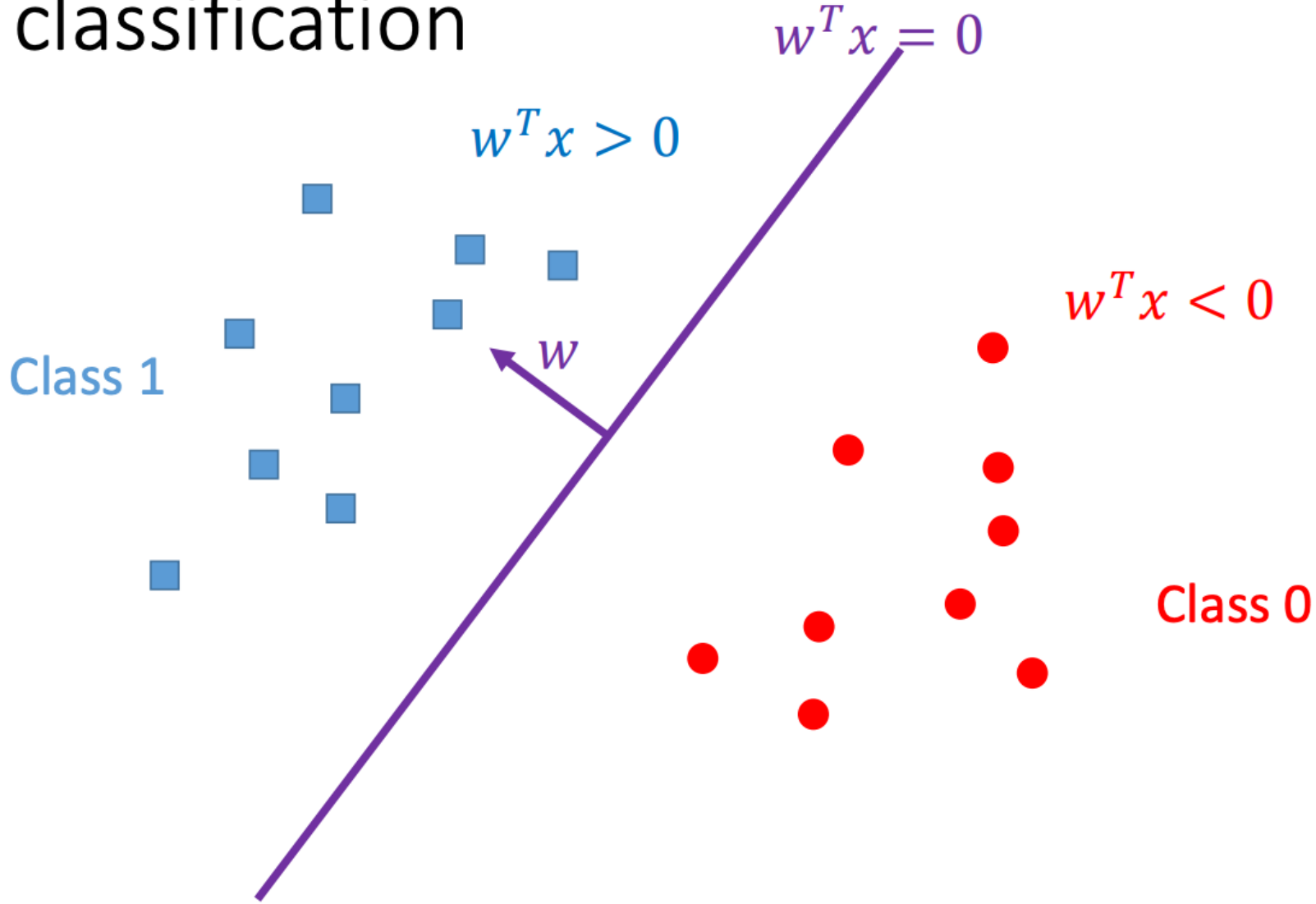
Example 2: Spam detection

	#"$"	#"Mr."	#"sale"	…	Spam?
Email 1	2	1	1		Yes
Email 2	0	1	0		No
Email 3	1	1	1		Yes
…					
Email n	0	0	0		No
New email	0	0	1		??

Why classification

- Classification: a kind of summary
- Easy to interpret
- Easy for making decisions

Linear classification



Linear classification: natural attempt

- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Hypothesis $f_w(x) = w^T x$
  - $y = 1$ if $w^T x > 0$
  - $y = 0$ if $w^T x < 0$
- Prediction: $y = \text{step}(f_w(x)) = \text{step}(w^T x)$

Linear model $\mathcal{H}$

Linear classification: natural attempt

- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Find $f_w(x) = w^T x$ to minimize $\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\text{step}(w^T x_i) \neq y_i]$

- Drawback: difficult to optimize
  - NP-hard in the worst case

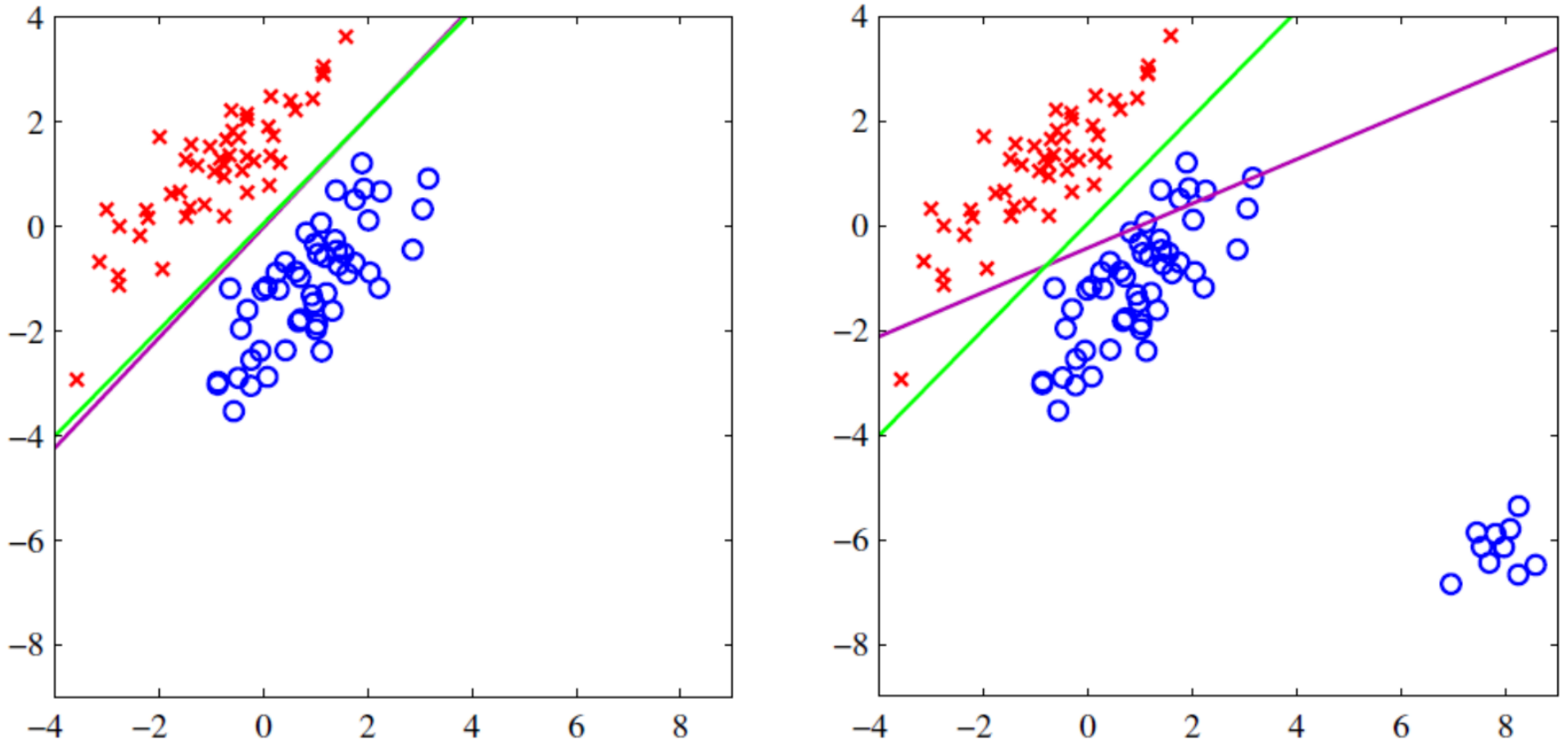
0-1 loss

Linear classification: simple approach

- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Find $f_w(x) = w^T x$ that minimizes $\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2$

Reduce to linear regression; ignore the fact $y \in \{0,1\}$

Linear classification: simple approach



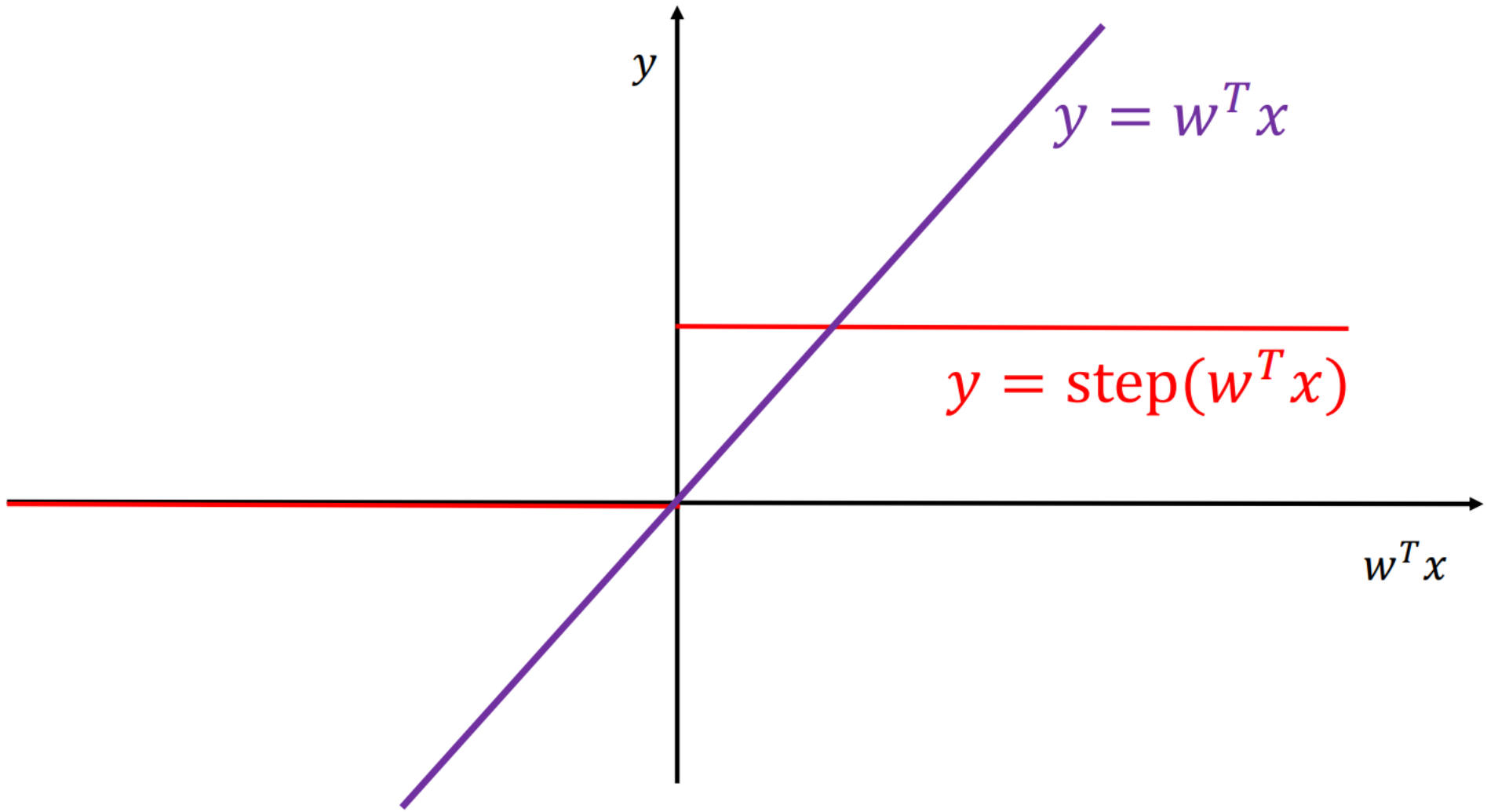
Drawback: not robust to "outliers"

Figure borrowed from *Pattern Recognition and Machine Learning*, Bishop

Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

Why better?

Compare the two



Between the two

- Prediction bounded in $[0,1]$
- Smooth

- Sigmoid: $\sigma(a) = \frac{1}{1+\exp(-a)}$

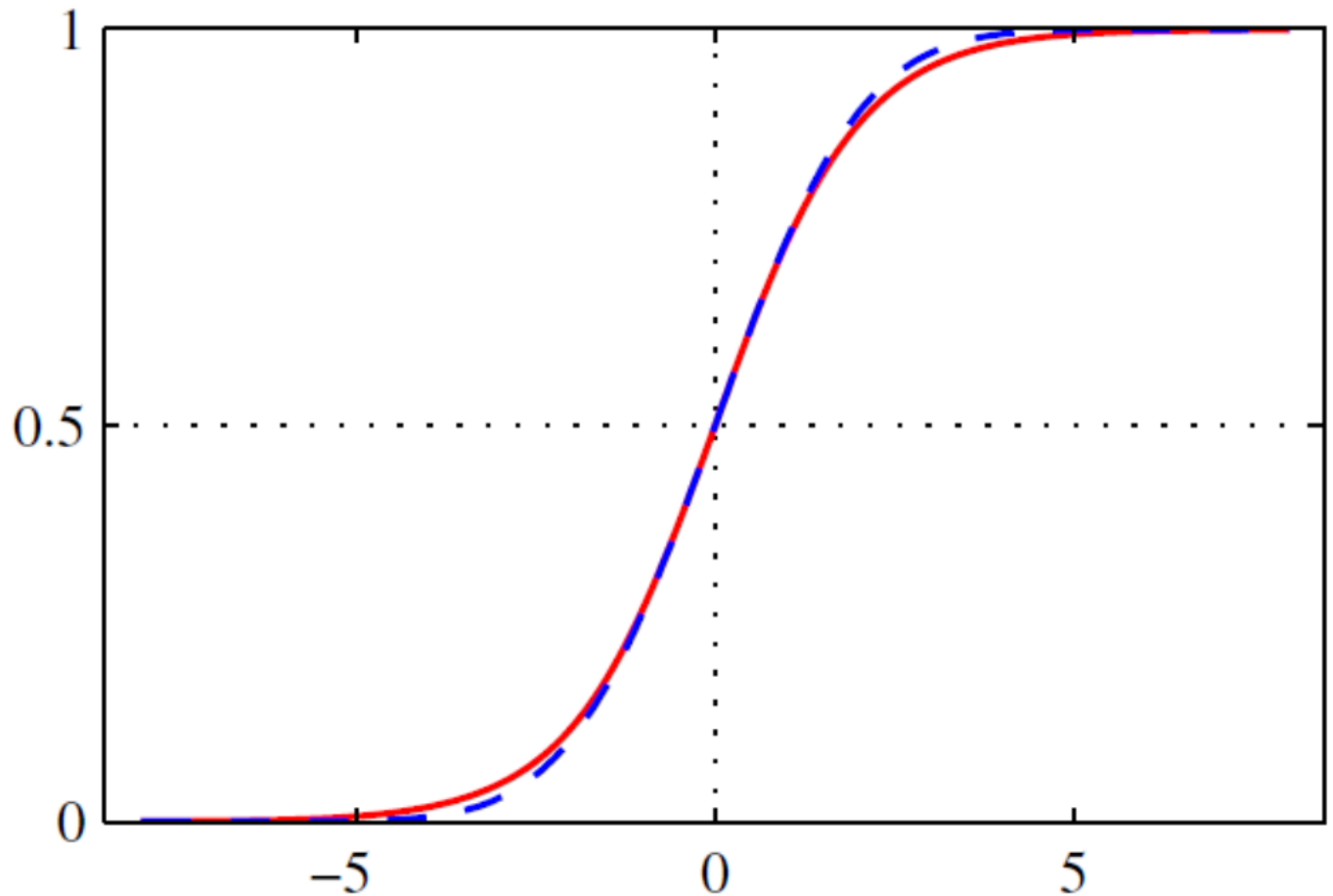


Figure borrowed from *Pattern Recognition and Machine Learning*, Bishop

Cost function for **Linear Regression**

- **Squash the output of the linear function**

$$\text{Sigmoid}(w^T x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- Find w that minimizes $\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(\sigma(w^T x_i) - y_i)^2$

Cost function for **Logistic Regression**

- Squash the output of the linear function

$$\text{Sigmoid}(w^T x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- A better approach: Interpret as a probability

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

$$P_w(y = 0|x) = 1 - P_w(y = 1|x) = 1 - \sigma(w^T x)$$

Cost function for **Logistic Regression**

- **Given training data** $\{(x_i, y_i): 1 \leq i \leq n\}$ **i.i.d. from distribution** D
- **Find** w **that minimizes**

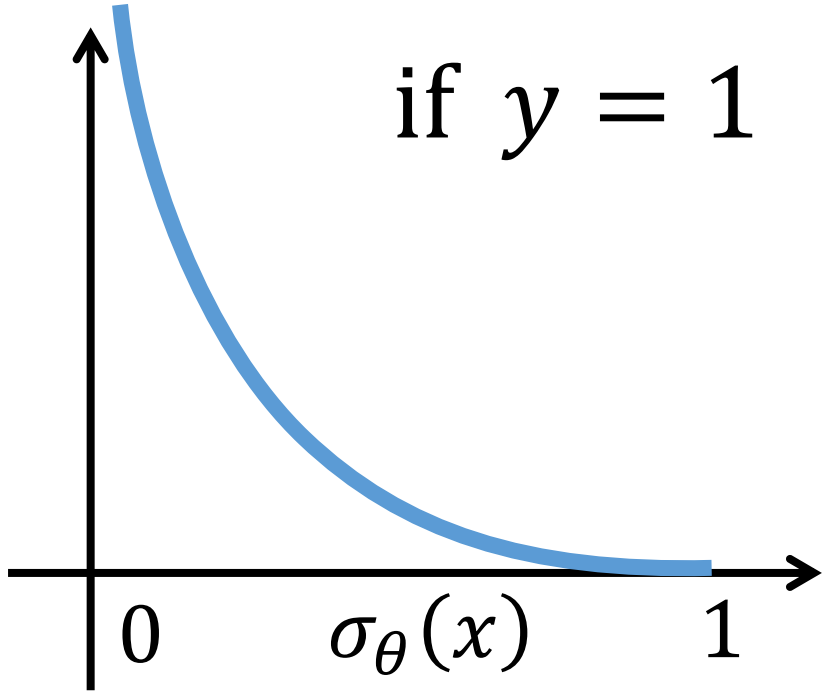
$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log P_w(y|x)$$

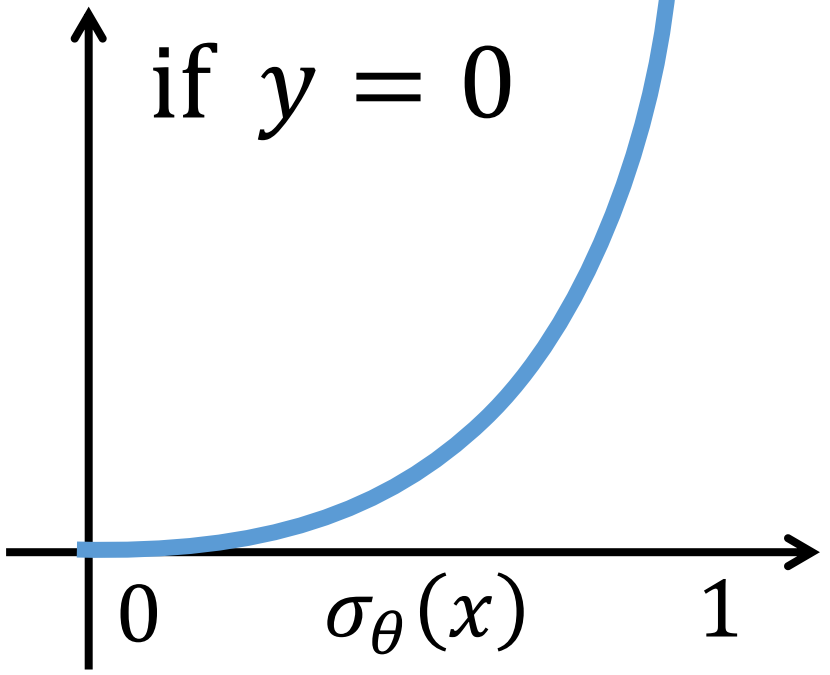
$$\hat{L}(w) = -\frac{1}{n}\sum_{y_i=1} \log\sigma(w^T x_i) - \frac{1}{n}\sum_{y_i=0} \log[1 - \sigma(w^T x_i)]$$

Logistic regression:
MLE with sigmoid

Cost function for **Logistic Regression**

$$\text{Cost}(\sigma_\theta(x), y) = \begin{cases} -\log\left(\sigma_\theta(x)\right) & \text{if } y = 1 \\ -\log\left(1 - \sigma_\theta(x)\right) & \text{if } y = 0 \end{cases}$$





Cost function for **Logistic Regression**

- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Find w that minimizes

$$\hat{L}(w) = -\frac{1}{n}\sum_{y_i=1} \log\sigma(w^T x_i) - \frac{1}{n}\sum_{y_i=0} \log[1 - \sigma(w^T x_i)]$$

No close form solution;
Need to use gradient descent

Properties of sigmoid function

- Bounded

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \in (0,1)$$

- Symmetric

$$1 - \sigma(a) = \frac{\exp(-a)}{1 + \exp(-a)} = \frac{1}{\exp(a) + 1} = \sigma(-a)$$

- Gradient

$$\sigma'(a) = \frac{\exp(-a)}{(1 + \exp(-a))^2} = \sigma(a)(1 - \sigma(a))$$

Questions: Why not Logistic Regression + Square Error?

For example:

Let's say

•Actual label for a given sample in a dataset is "1"

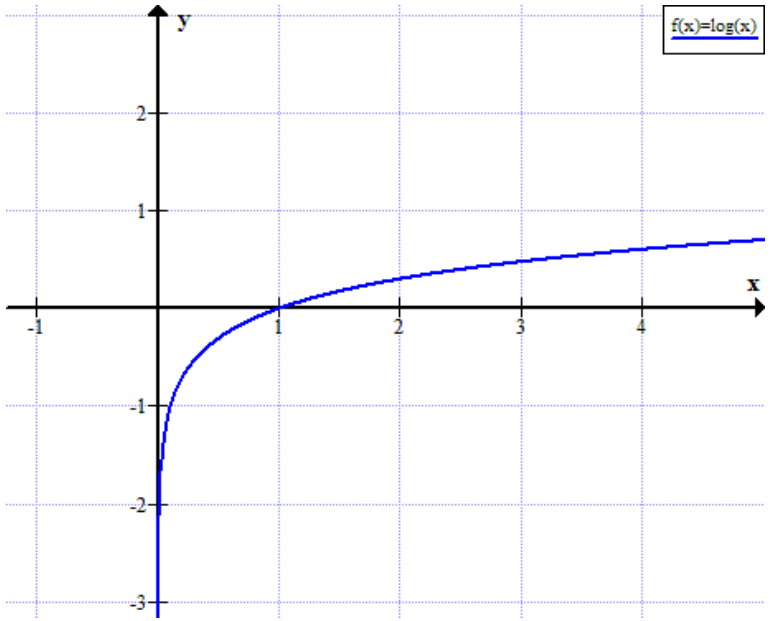
•Prediction from the model after applying sigmoid function = 0

Loss value when using MSE:

$(1- 0)^2 = 1$

Loss value when using log loss:

Before plugging in the values for loss equation, we can have a look at how the graph of ***log(x)*** looks like.



As seen from the above graph as **x tends to 0, *log(x) tends to -infinity.***

Therefore, loss value would be:

-(1 * log(0) + 0 * log(1)) = tends to infinity !!

As seen above, loss value using MSE was much much less compared to the loss value computed using the log loss function. Hence it is very clear to us that MSE doesn't strongly penalize misclassifications even for the perfect mismatch!

Logistic Regression + Square Error

Step 1: $f_{w,b}(x) = \sigma\left(\sum_i w_i x_i + b\right)$

Step 2: Training data: $(x^n, \hat{y}^n)$, $\hat{y}^n$: 1 for class 1, 0 for class 2

$$L(f) = \frac{1}{2}\sum_n \left(f_{w,b}(x^n) - \hat{y}^n\right)^2$$

Step 3:

$$\frac{\partial\, (f_{w,b}(x)-\hat{y})^2}{\partial w_i} = 2\left(f_{w,b}(x) - \hat{y}\right) \frac{\partial f_{w,b}(x)}{\partial z} \frac{\partial z}{\partial w_i}$$

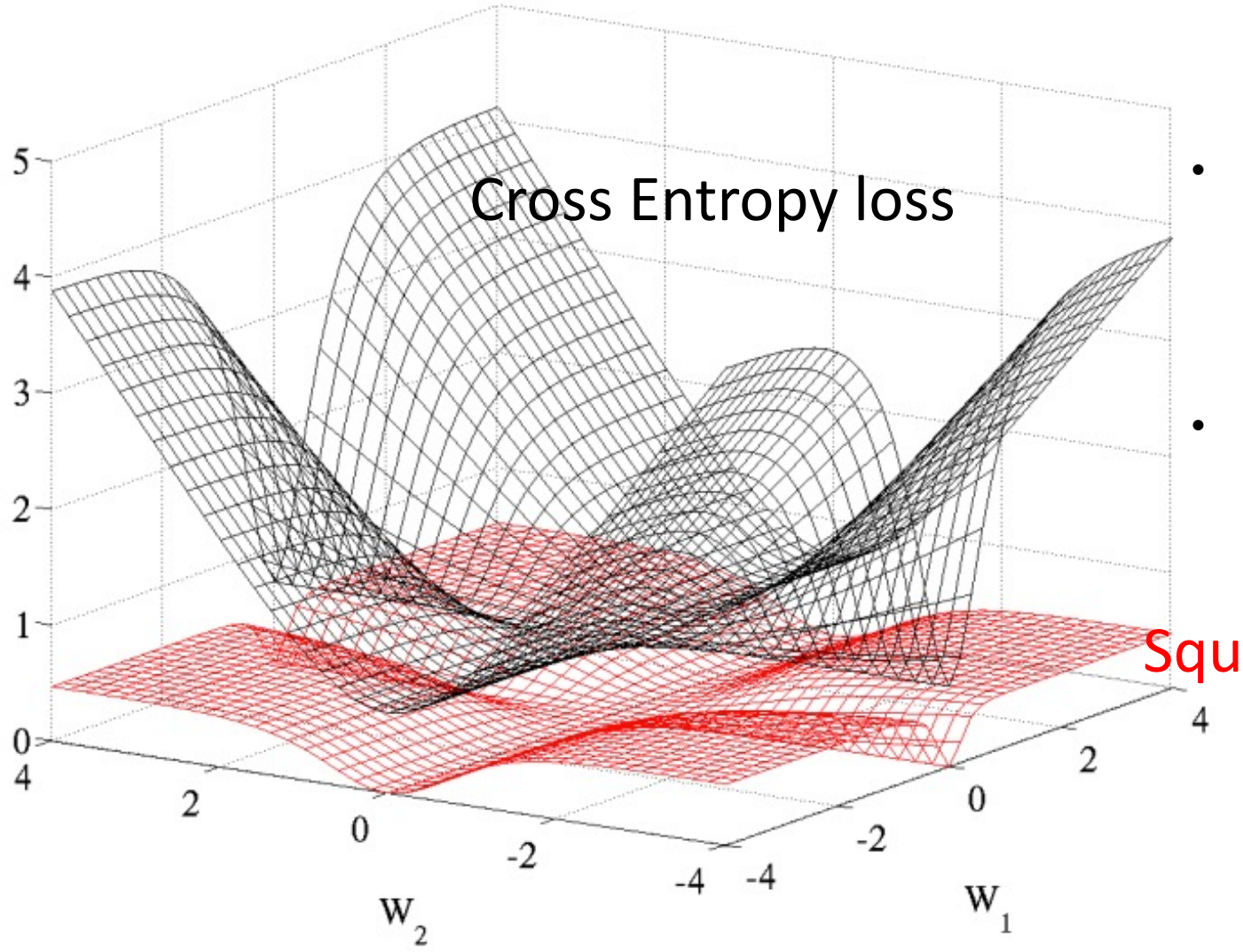
$$= 2\left(f_{w,b}(x) - \hat{y}\right) f_{w,b}(x)\left(1 - f_{w,b}(x)\right) x_i$$

$\hat{y}^n = 1$ If $f_{w,b}(x^n) = 1$ (close to target) ➡ $\partial L/\partial w_i = 0$

If $f_{w,b}(x^n) = 0$ (far from target) ➡ $\partial L/\partial w_i = 0$

Understanding the difficulty of training deep feedforward neural networks
https://proceedings.mlr.press/v9/glorot10a.html

Total loss



- We found that the plateaus in the training criterion (as a function of the parameters) are less present with the log-likelihood cost function.
- There are clearly more severe plateaus with the quadratic cost.

Understanding the difficulty of training deep feedforward neural networks
https://proceedings.mlr.press/v9/glorot10a.html

Linear classifiers: which hyperplane?

Many possible solutions for *a, b, c.*

Some methods find a separating hyperplane, but not the optimal one according to some criterion of expected goodness.. e.g., **Perceptron**

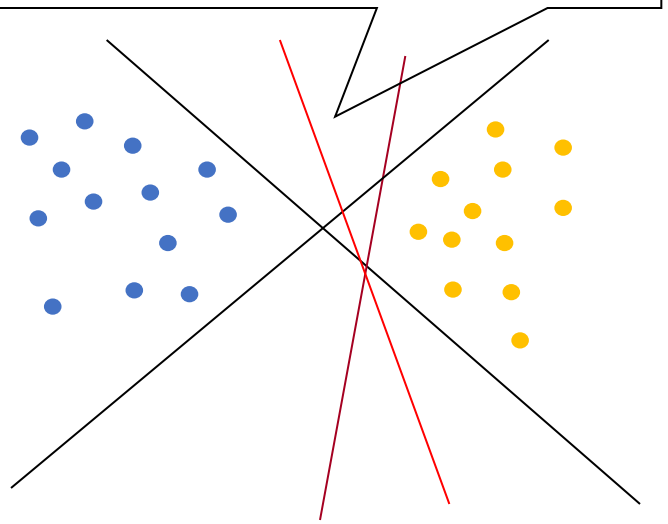
Support Vector Machine (SVM) finds an optimal solution:

Maximizes distance between hyperplane and "difficult points" close to decision boundary.

Intuition: if there are NO points near the decision surface, then there are no very uncertain classification decisions

line represents the decision boundary:

$ax + by - c = 0$



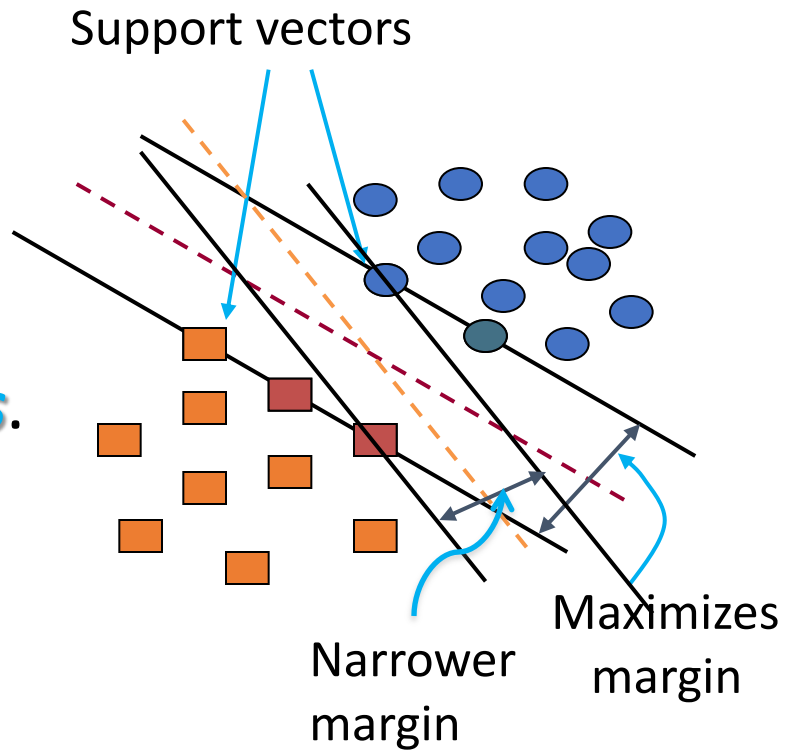
Support Vector Machine (SVM)

SVMs **maximize the *margin*** around the separating hyperplane .. a.k.a. **large margin classifiers**.

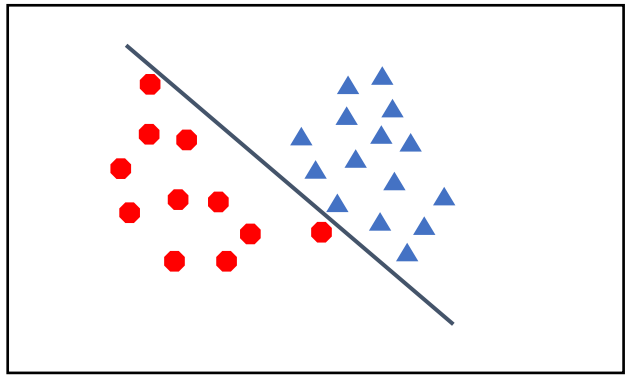
The decision function is fully specified by a subset of training samples, ***the support vectors***.

Solving SVMs is a ***quadratic programming*** problem.

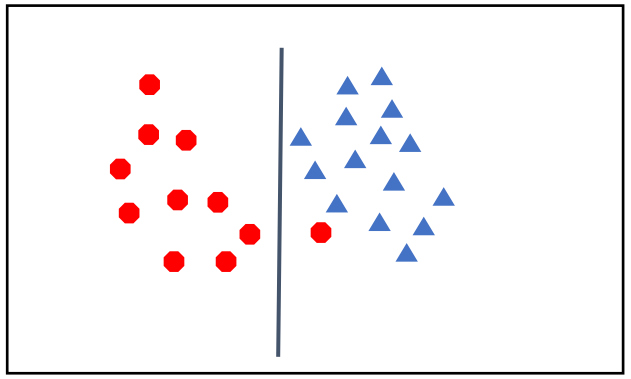
Successfully used for text classification etc.



SVM: Linear Separability



Points can be **linearly separated** but there is a very narrow margin.



However; **a large margin solution** should be better, although there would be one violation here.

In general, there is a trade off between the margin & number of mistakes on training data.

Soft Margin Classification

If training data is **NOT linearly separable**, *slack variables* ξ_i can be added to allow misclassification of difficult/noisy examples.

Resulting margin called ***'Soft'***.

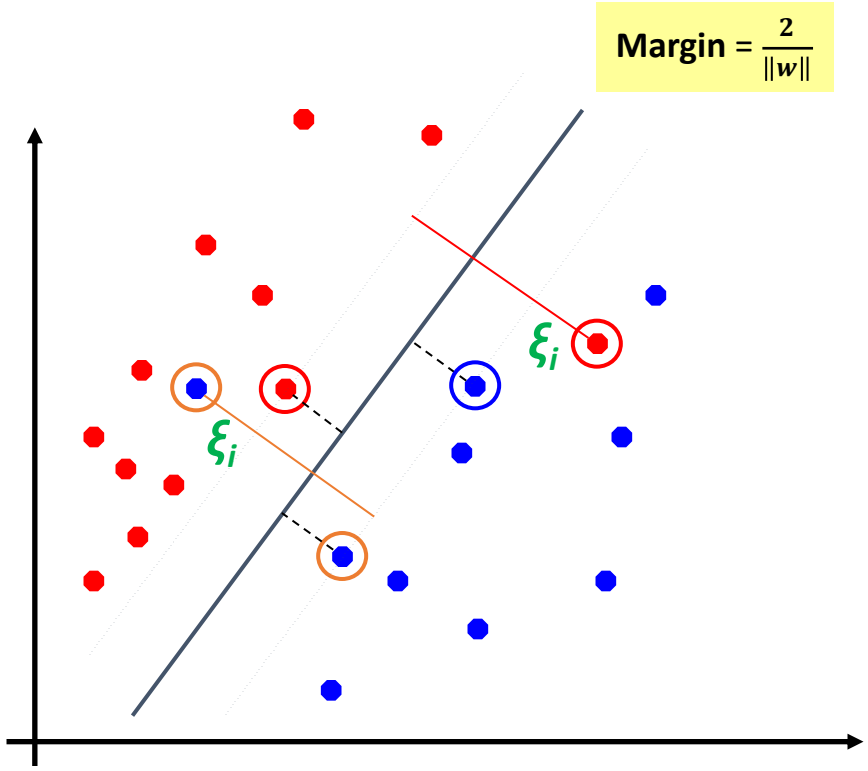
move some points to where they belong; at a cost / penalty:

$$\min_{\mathbf{w}, b, \xi_i > 0} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

$$\text{s.t. } \forall i: y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1 - \xi_i$$

parameter *C* can be viewed as a way to control overfitting i.e., a **regularization** term.

small C allows constraints to be easily ignored → **large margin**
large C makes constraints hard to ignore → **narrow margin**
C = ∞ enforces all constraints → **hard margin**.



Linear SVM: Summary

SVM classifier is a **separating hyperplane**.

Key training points are the **support vectors** which define the **hyperplane**.

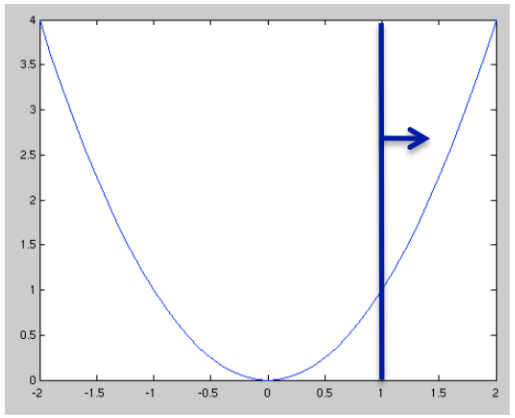
Quadratic optimization algorithms can identify which training points $\mathbf{x}_i$ are **support vectors** with non-zero Lagrangian multipliers α_i.

In dual formulation & solution: training points appear only inside **inner products**:

Find $\alpha_1 \ldots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \le \alpha_i \le C$ for all α_i

$$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^T\mathbf{x} + b$$

Lagrange multipliers – Dual variables



$$\min_x \; x^2 \quad \text{s.t.} \quad x \geq b$$

Add Lagrange multiplier

Rewrite Constraint

Introduce Lagrangian (objective):

$$L(x, \alpha) = x^2 - \alpha(x - b)$$

Why is this equivalent?

- min is fighting max!

$x<b \rightarrow (x-b)<0 \rightarrow \max_\alpha -\alpha(x-b) = \infty$

- min won't let this happen!

We will solve:

$$\min_x \max_\alpha \; L(x, \alpha) \quad \text{s.t.} \quad \alpha \geq 0$$

Add new constraint

$x>b,\ \alpha \geq 0 \rightarrow (x-b)>0 \rightarrow \max_\alpha -\alpha(x-b) = 0,\ \alpha^*=0$

- min is cool with 0, and $L(x, \alpha)=x^2$ (original objective)

$x=b \rightarrow \alpha$ can be anything, and $L(x, \alpha)=x^2$ (original objective)

The *min* on the outside forces *max* to behave, so constraints will be satisfied.

Dual SVM derivation (1) – the linearly separable case

Original optimization problem:

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w}$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1, \ \forall j$$

Rewrite constraints

One Lagrange multiplier per example

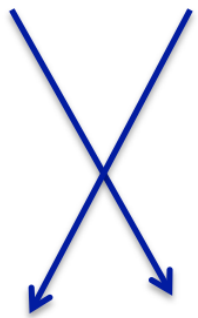
Lagrangian:

$$L(\mathbf{w}, \alpha) = \frac{1}{2}\mathbf{w}.\mathbf{w} - \sum_j \alpha_j \left[\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j - 1\right]$$
$$\alpha_j \geq 0, \ \forall j$$

Our goal now is to solve: $\min_{\vec{w},b} \ \max_{\vec{\alpha}\geq 0} \ L(\vec{w}, \vec{\alpha})$

Dual SVM derivation (2) – the linearly separable case

(Primal) $$\min_{\vec{w},b} \max_{\vec{\alpha}\geq 0} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$



Swap min and max

(Dual) $$\max_{\vec{\alpha}\geq 0} \min_{\vec{w},b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w}\cdot\vec{x}_j + b)\, y_j - 1\right]$$

Slater's condition from convex optimization guarantees that these two optimization problems are equivalent!

Dual SVM derivation (3) – the linearly separable case

(Dual) $$\max_{\vec{\alpha} \geq 0} \min_{\vec{w}, b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w} \cdot \vec{x}_j + b) y_j - 1 \right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

(Dual) $$\max_{\vec{\alpha} \geq 0,\ \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\vec{x}_i \cdot \vec{x}_j)$$

Sums over all training examples — scalars — dot product

Dual SVM derivation (3) – the linearly separable case

(Dual) $$\max_{\vec{\alpha} \geq 0} \min_{\vec{w}, b} \frac{1}{2}||\vec{w}||^2 - \sum_j \alpha_j \left[(\vec{w} \cdot \vec{x}_j + b) y_j - 1 \right]$$

Can solve for optimal **w**, b as function of α:

$$\frac{\partial L}{\partial w} = w - \sum_j \alpha_j y_j x_j \quad \rightarrow \quad \mathbf{w} = \sum_j \alpha_j y_j \mathbf{x}_j$$

$$\frac{\partial L}{\partial b} = -\sum_j \alpha_j y_j \quad \rightarrow \quad \sum_j \alpha_j y_j = 0$$

Substituting these values back in (and simplifying), we obtain:

(Dual) $$\max_{\vec{\alpha} \geq 0,\ \sum_j \alpha_j y_j = 0} \sum_j \alpha_j - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \left(\vec{x}_i \cdot \vec{x}_j \right)$$

So, in dual formulation we will solve for α directly!

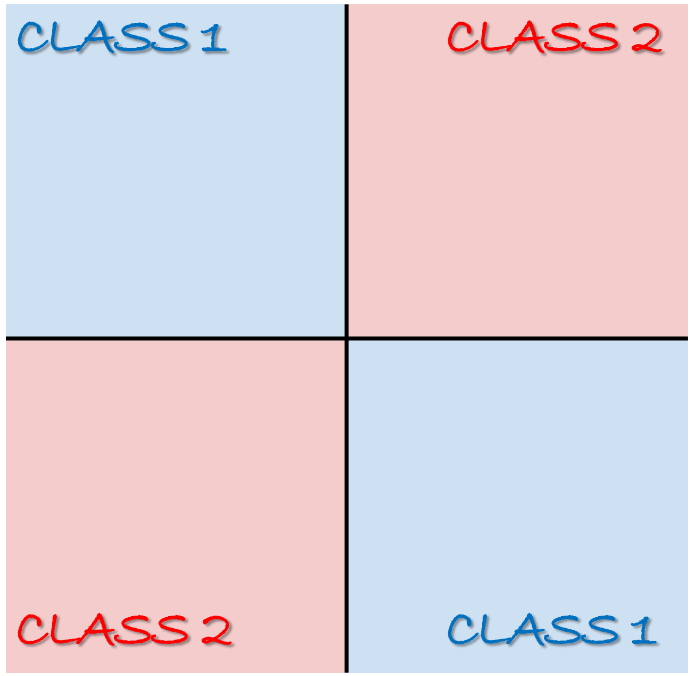
- **w** and b are computed from α (if needed)

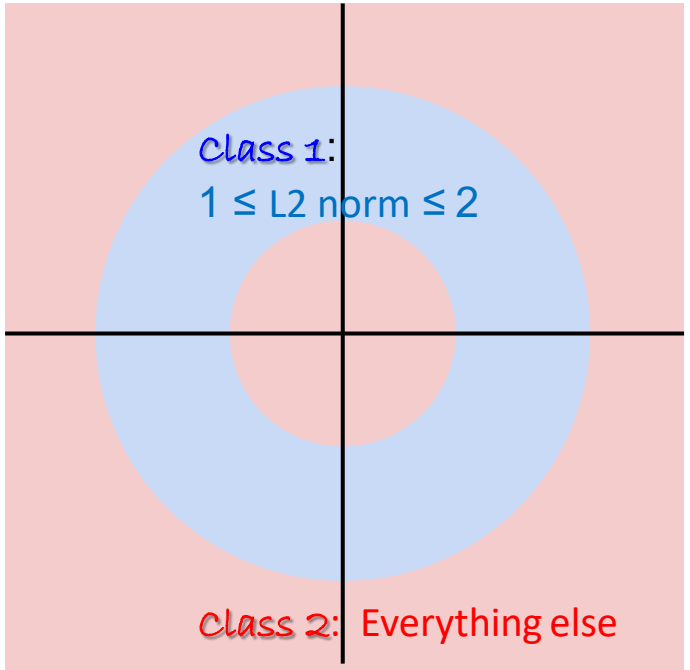
Break time-15 mins

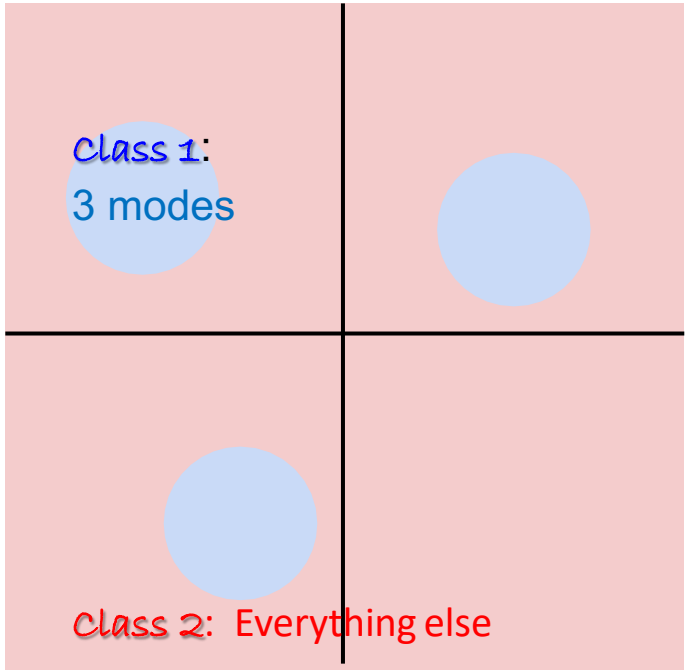
Nonlinear classification

Linear Classifier: hard cases

Two Category examples: a single linear line cannot be used to separate the classes.







Nearest Neighbour Classifier

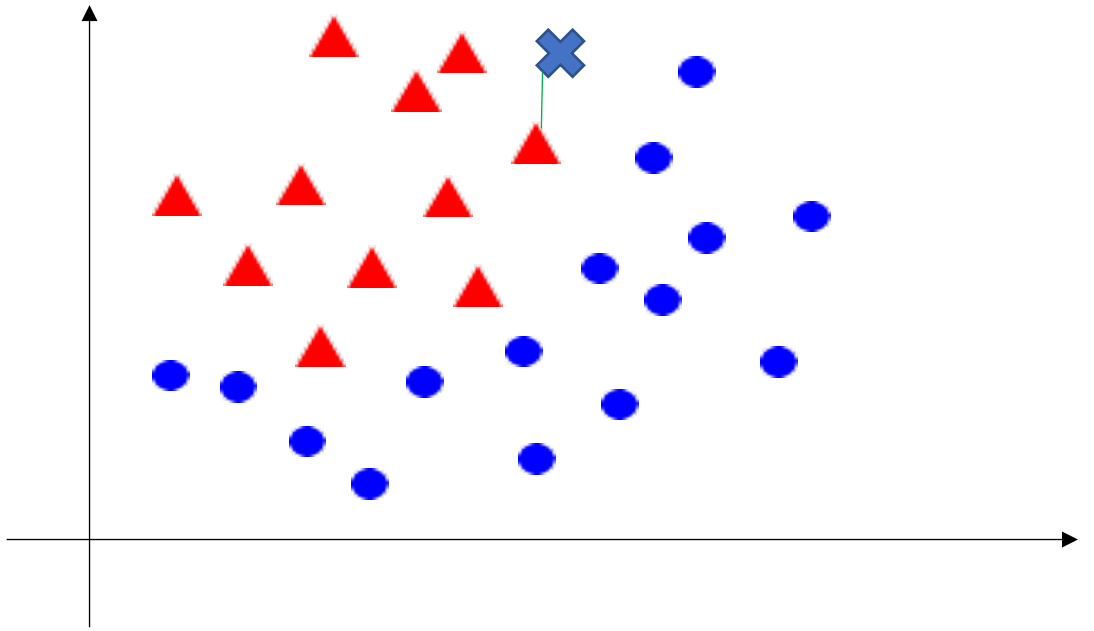
For each test point x to be classified, find the nearest sample in the training data

What **distance measure** to use?

Euclidean, Manhattan ?

How fast is training & prediction?

training: $\mathcal{O}(1)$; prediction $\mathcal{O}(N)$ with N examples



K Nearest Neighbour Classifier

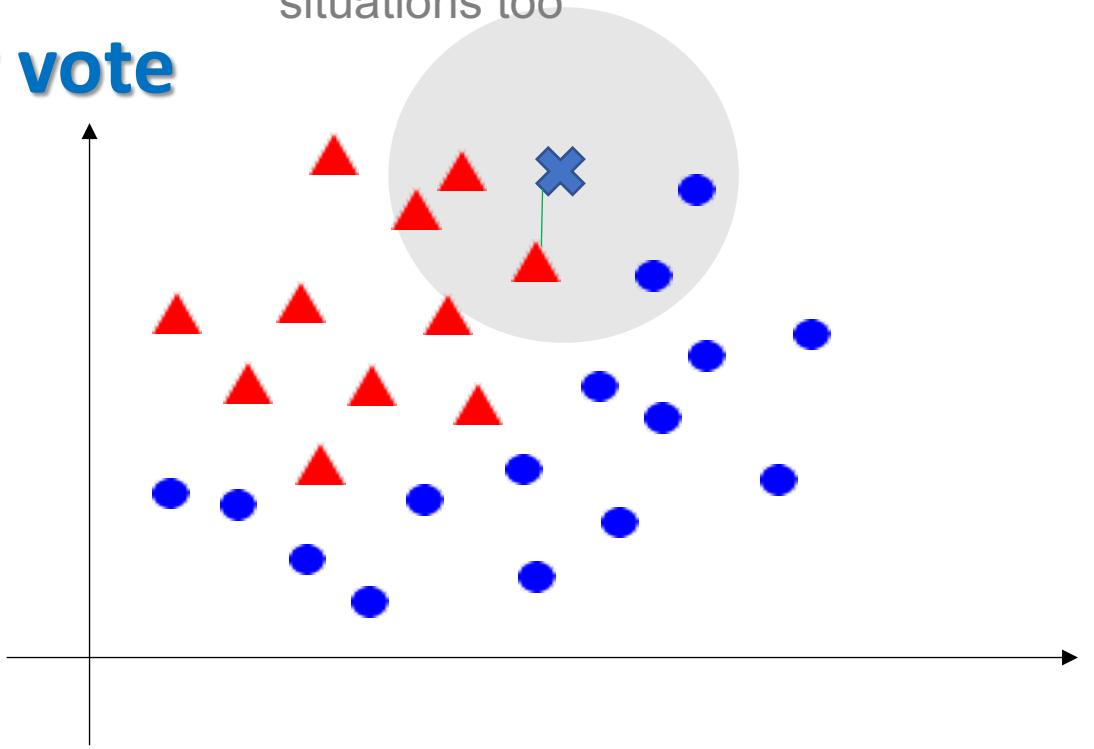
For each test point x to be classified, find the **K** nearest samples in the training data

Classify x according to the **majority vote** of their class labels

Aim of supervised learning:

to do well on test data which is **not known** during learning..

e.g. K = 5
applicable to multi-class situations too



Nearest Neighbour Classifier

Distance measures for the **1**-NN / **K**-NN:

L1 distance (Manhattan):

$$d_1(I_1, I_2) = \sum_m |I_1^m - I_2^m|$$

L2 distance (Euclidean):

$$d_2(I_1, I_2) = \sqrt{\sum_m (I_1^m - I_2^m)^2}$$

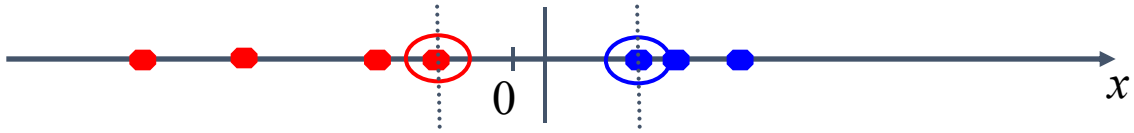
What's the ***best value*** *of* **K** to use? What is the ***best distance measure*** to use?

Very problem-dependent. *Try & see what works best.*

NN Classification Error : $\frac{1}{N}\sum_{i=1}^{N}[y_i \neq f(x_i)]$

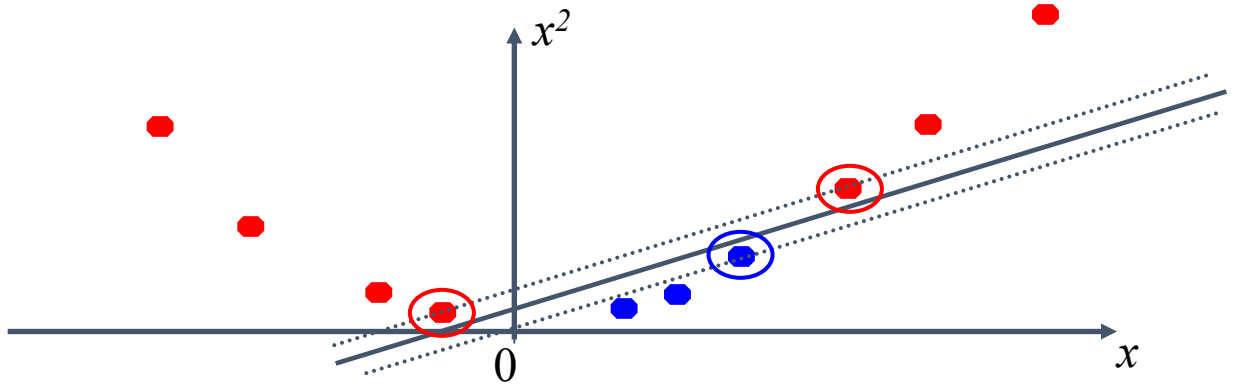
Non-linear SVMs

Datasets that are **linearly separable** (even with some noise) .. *easy to solve*..



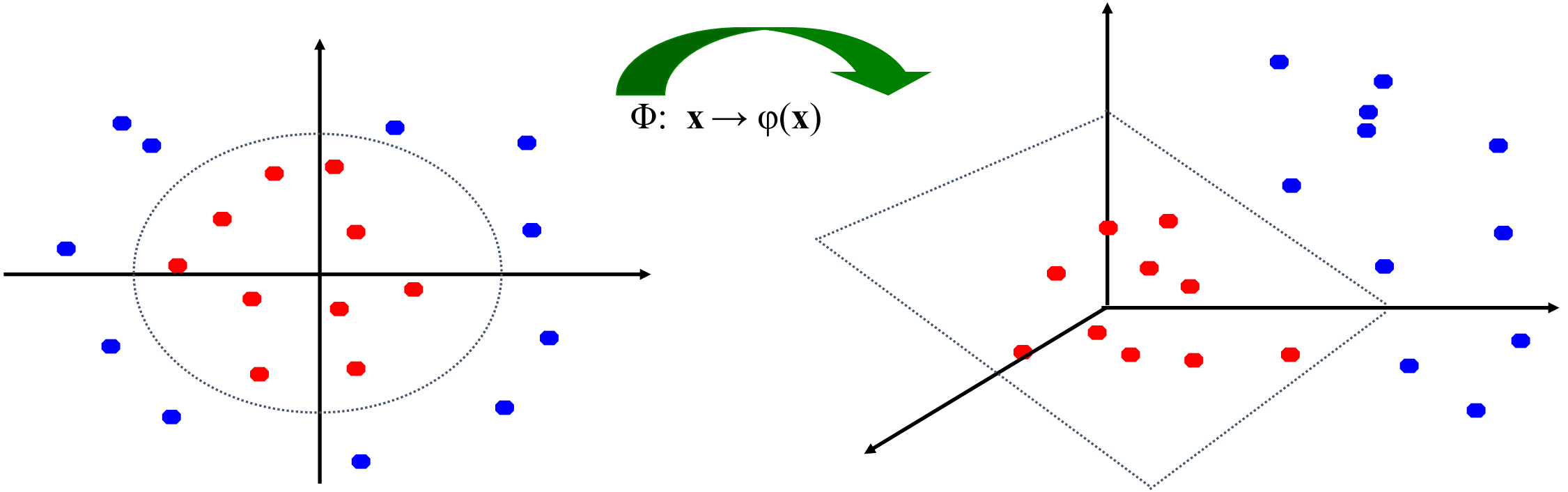
If the dataset is **NOT** linearly separable ??

How about … mapping data to a higher-dimensional space:



Non-linear SVMs: Feature spaces

General idea: original feature space can be **mapped to some higher-dimensional feature space** where the training set is **separable**..



Dual for the non-separable case – same basic story (we will skip details)

Primal:

$$\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_j \xi_j$$
$$\left(\mathbf{w}.\mathbf{x}_j + b\right) y_j \geq 1 - \xi_j, \ \forall j$$
$$\xi_j \geq 0, \ \forall j$$

Solve for w,b, α :

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$
$$b = y_k - \mathbf{w}.\mathbf{x}_k$$

for any k where $C > \alpha_k > 0$

Dual:

$$\text{maximize}_{\alpha} \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

$$\hat{y} = \text{sign}\left(\sum_i^n \alpha_i (\mathbf{x}_i^T \mathbf{x})\right)$$

What changed?

- Added upper bound of C on α_i!
- Intuitive explanation:
  - Without slack, $\alpha_i \rightarrow \infty$ when constraints are violated (points misclassified)
  - Upper bound of C limits the α_i, so misclassifications are allowed

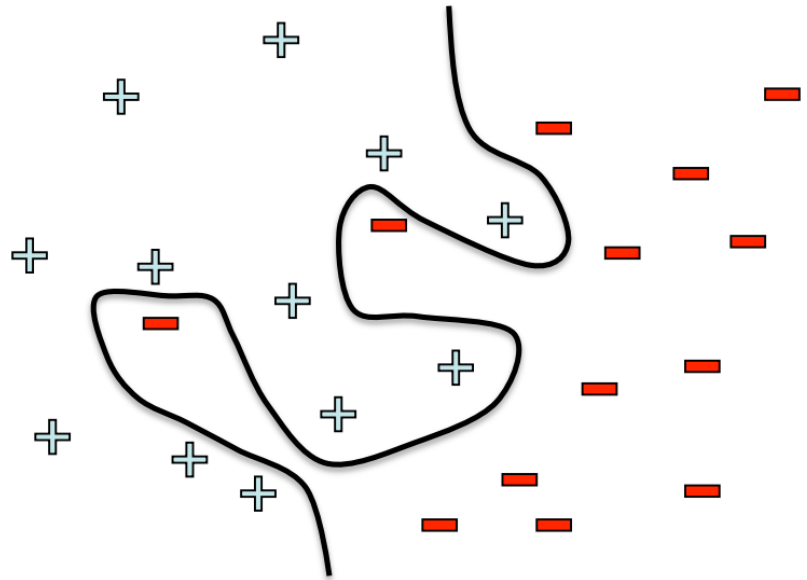
Wait a minute: why did we learn about the dual SVM?

- There are some quadratic programming algorithms that can solve the dual faster than the primal
  - At least for small datasets

- But, more importantly, the "**kernel trick**"!!!

Reminder: What if the data is not linearly separable?

Use features of features of features of features….

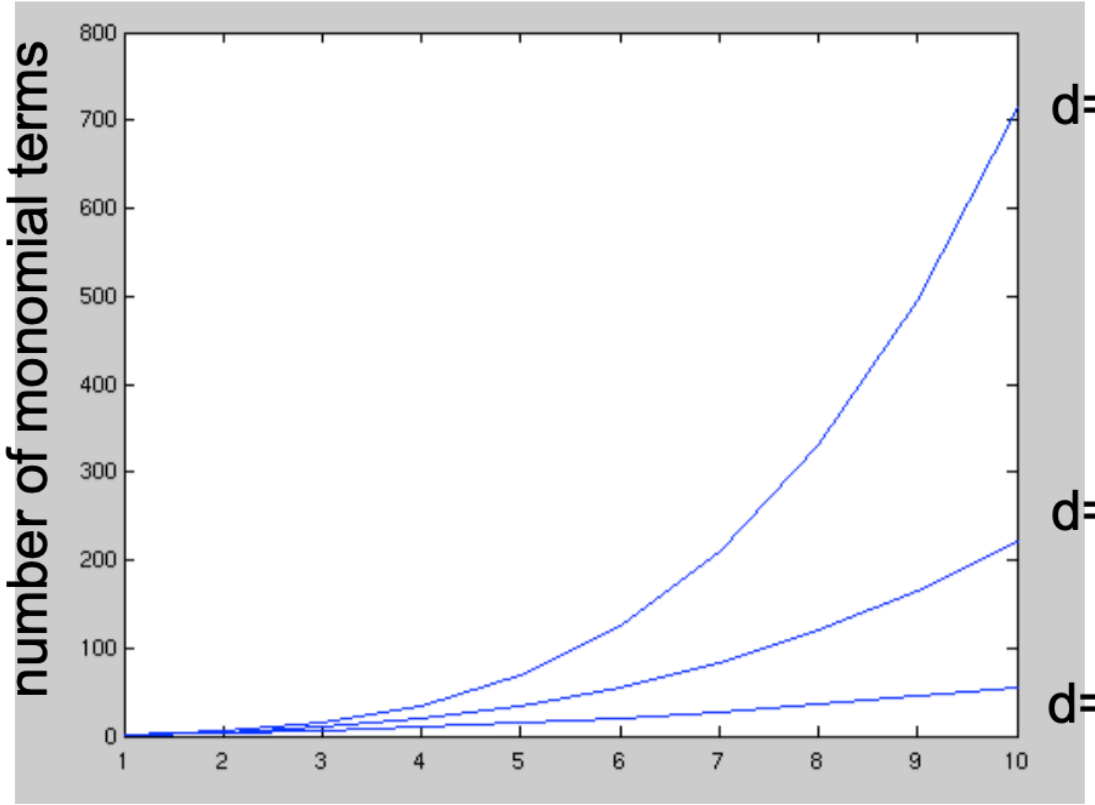


$$\phi(x) = \begin{pmatrix} x^{(1)} \\ \dots \\ x^{(n)} \\ x^{(1)}x^{(2)} \\ x^{(1)}x^{(3)} \\ \dots \\ e^{x^{(1)}} \\ \dots \end{pmatrix}$$

Feature space can get really large really quickly!

Higher order polynomials

$$\text{num. terms} = \binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!}$$



m – input features
d – degree of polynomial

grows fast!
d = 6, m = 100
about 1.6 billion terms

Dual formulation only depends on dot-products, not on **w**!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j$$

$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Remember the examples x only appear in one dot product

First, we introduce features:

$$\mathbf{x}_i \mathbf{x}_j \quad \rightarrow \quad \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Next, replace the dot product with a Kernel:

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$
$$\sum_i \alpha_i y_i = 0$$
$$C \geq \alpha_i \geq 0$$

Why is this useful???

Efficient dot-product of polynomials

Polynomials of degree exactly *d*

d=1

$$\phi(u).\phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} . \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u.v$$

d=2

$$\phi(u).\phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} . \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2 u_1 v_1 u_2 v_2 + u_2^2 v_2^2$$
$$= (u_1 v_1 + u_2 v_2)^2$$
$$= (u.v)^2$$

For any *d (we will skip proof):*

$$\phi(u).\phi(v) = (u.v)^d$$

- Cool! Taking a dot product and exponentiating gives same results as mapping into high dimensional space and then taking dot produce

Finally: the "kernel trick"!

$$\text{maximize}_\alpha \quad \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

$$\sum_i \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0$$

- Never compute features explicitly!!!
  - Compute dot products in closed form
- Constant-time high-dimensional dot-products for many classes of features
- But, O(n^2) time in size of dataset to compute objective
  - Naïve implements slow
  - much work on speeding up

$$\mathbf{w} = \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)$$

$$b = y_k - \mathbf{w}.\Phi(\mathbf{x}_k)$$

for any k where $C > \alpha_k > 0$

Common kernels

- Polynomials of degree exactly *d*

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

- Polynomials of degree up to *d*

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian kernels

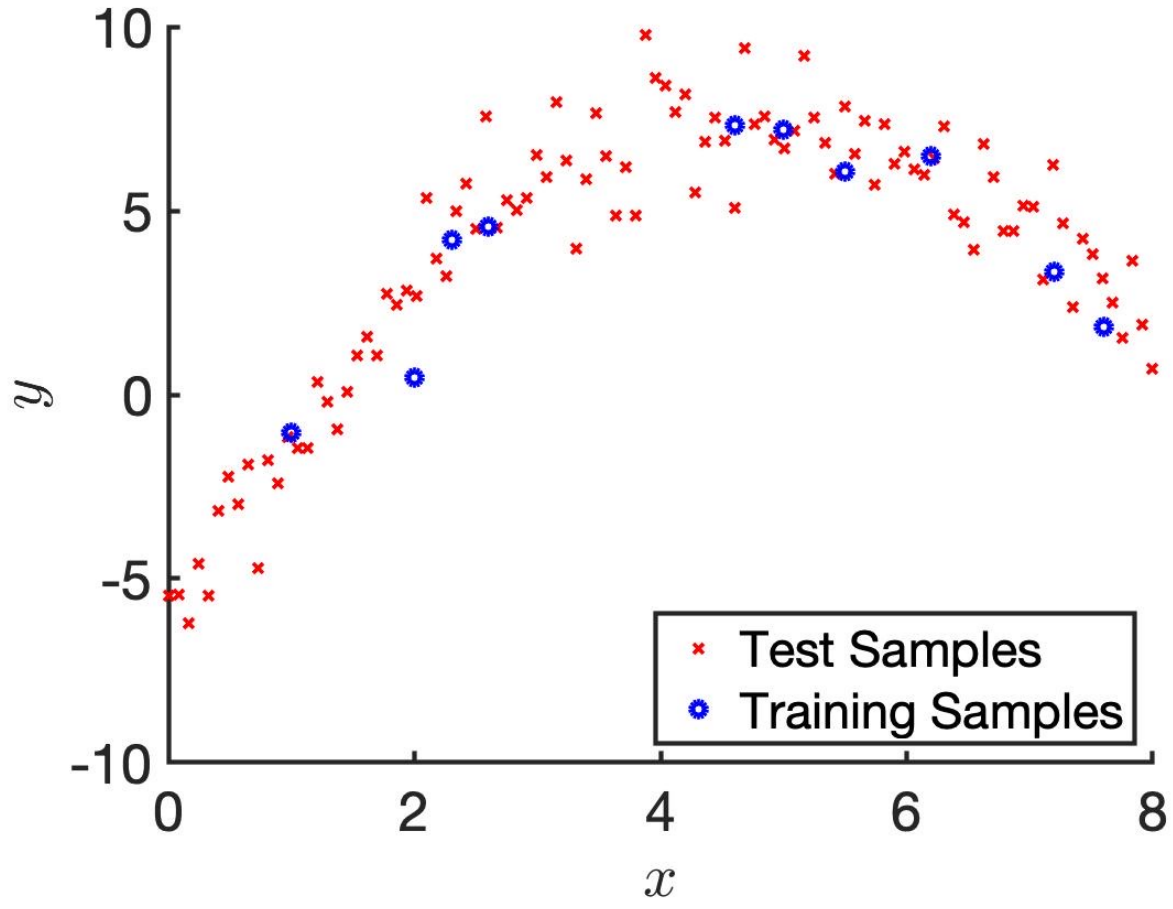
$$K(\vec{u}, \vec{v}) = \exp\left(-\frac{||\vec{u} - \vec{v}||_2^2}{2\sigma^2}\right)$$

- Sigmoid

$$K(\mathbf{u}, \mathbf{v}) = \tanh(\eta \mathbf{u} \cdot \mathbf{v} + \nu)$$

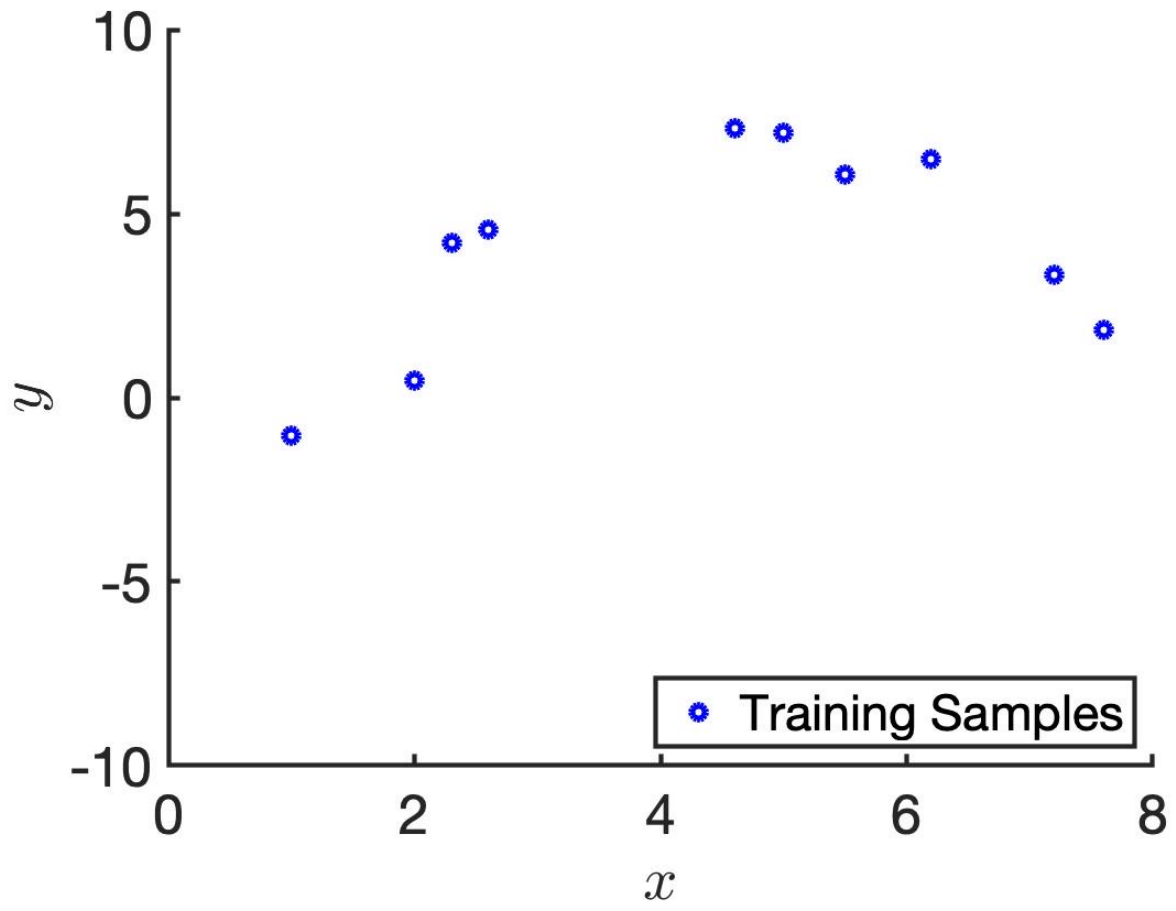
- And many others: very active area of research!

Overfitting Example

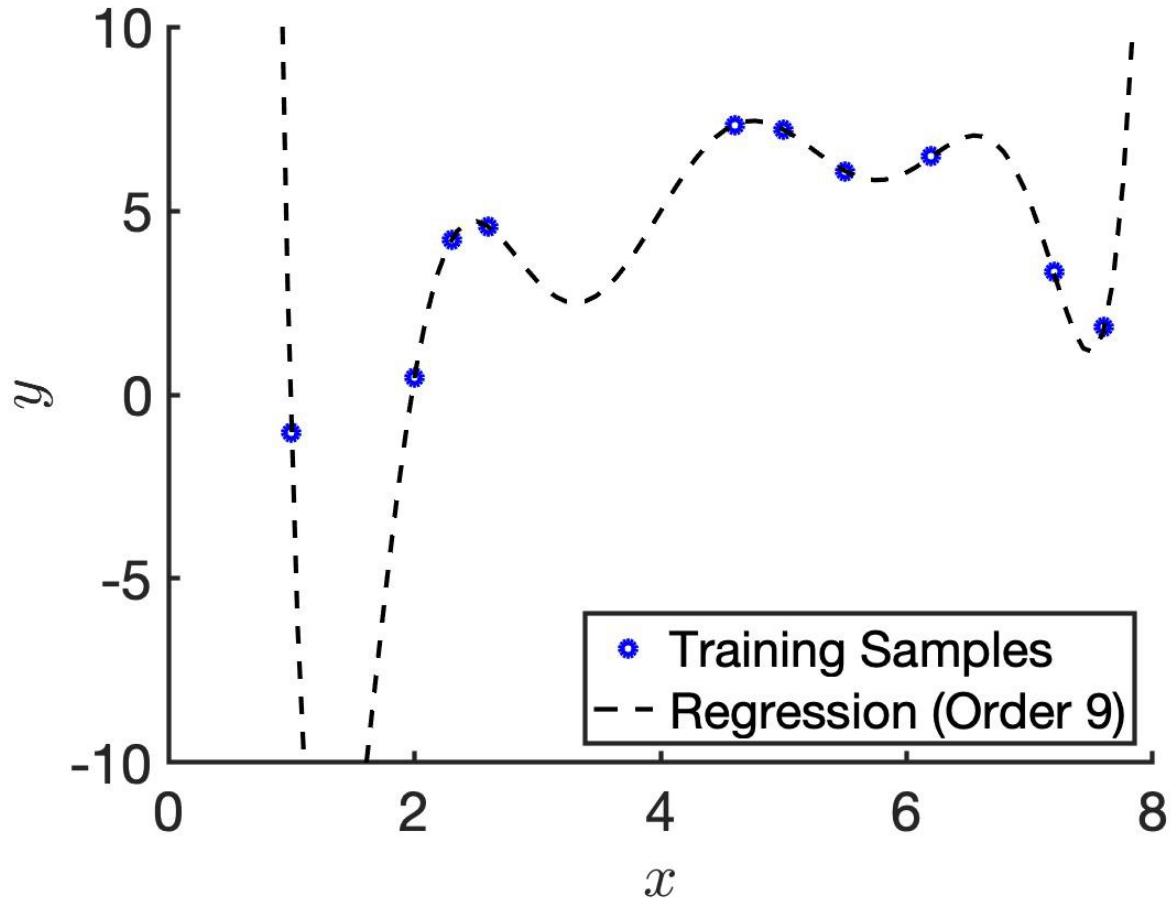


Overfitting

Overfitting Example

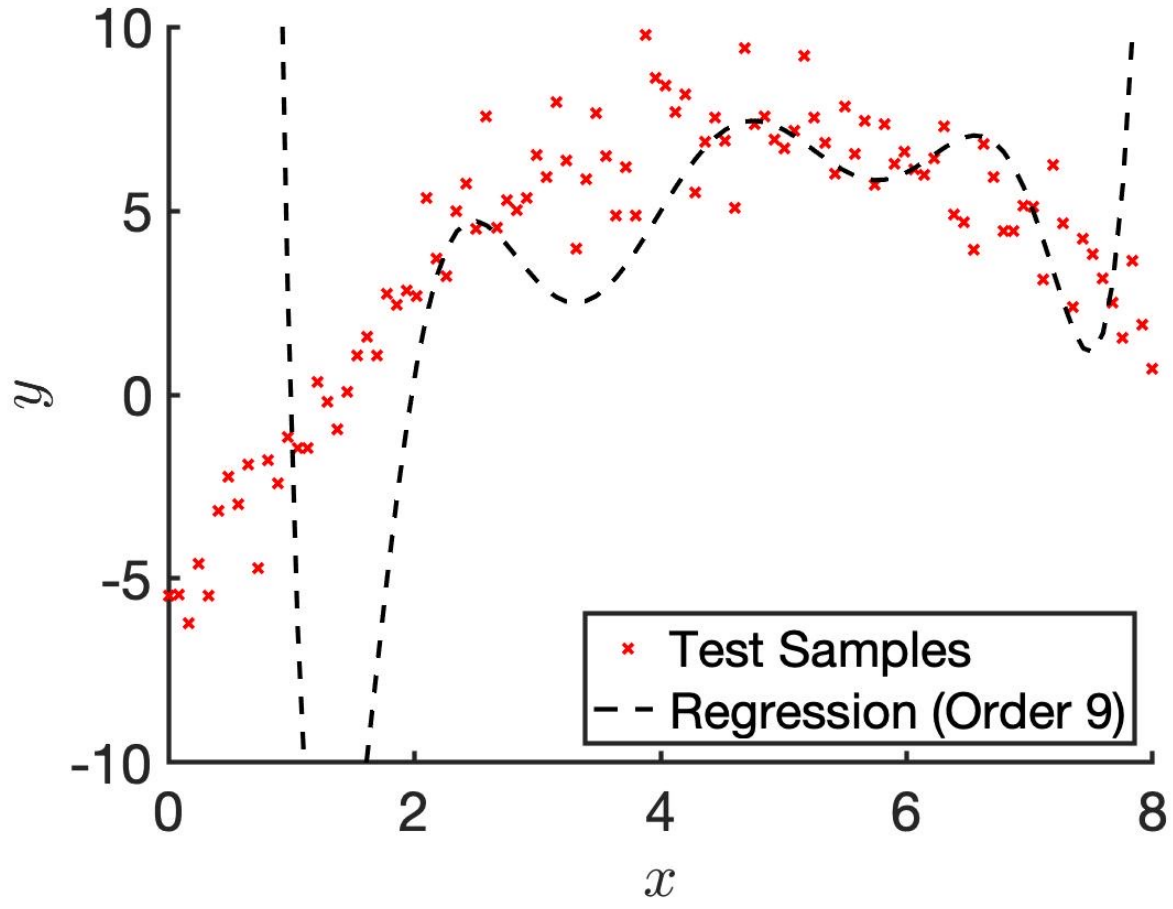


Overfitting Example



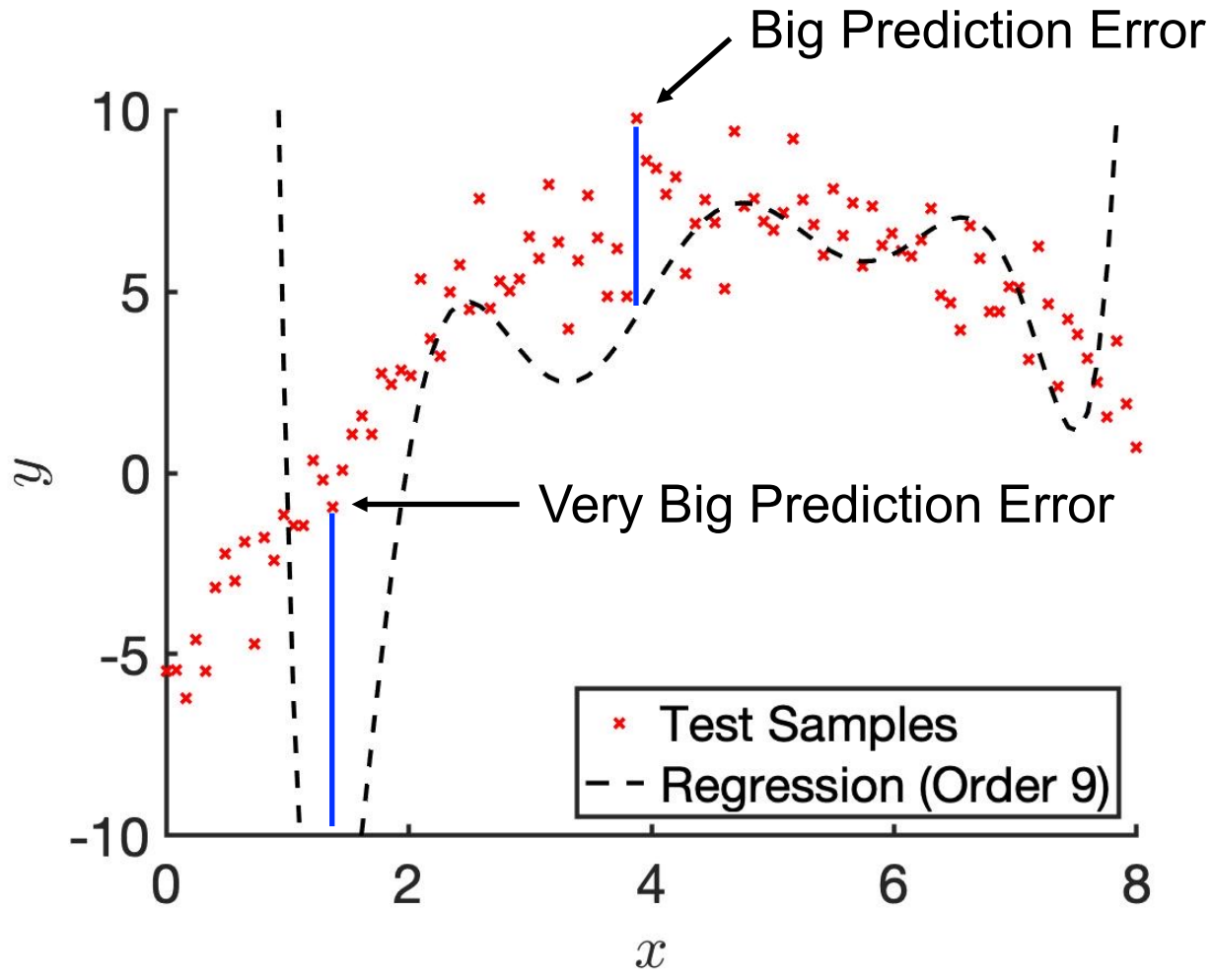
	Training Set Fit	
Order 9	Good	

Overfitting Example



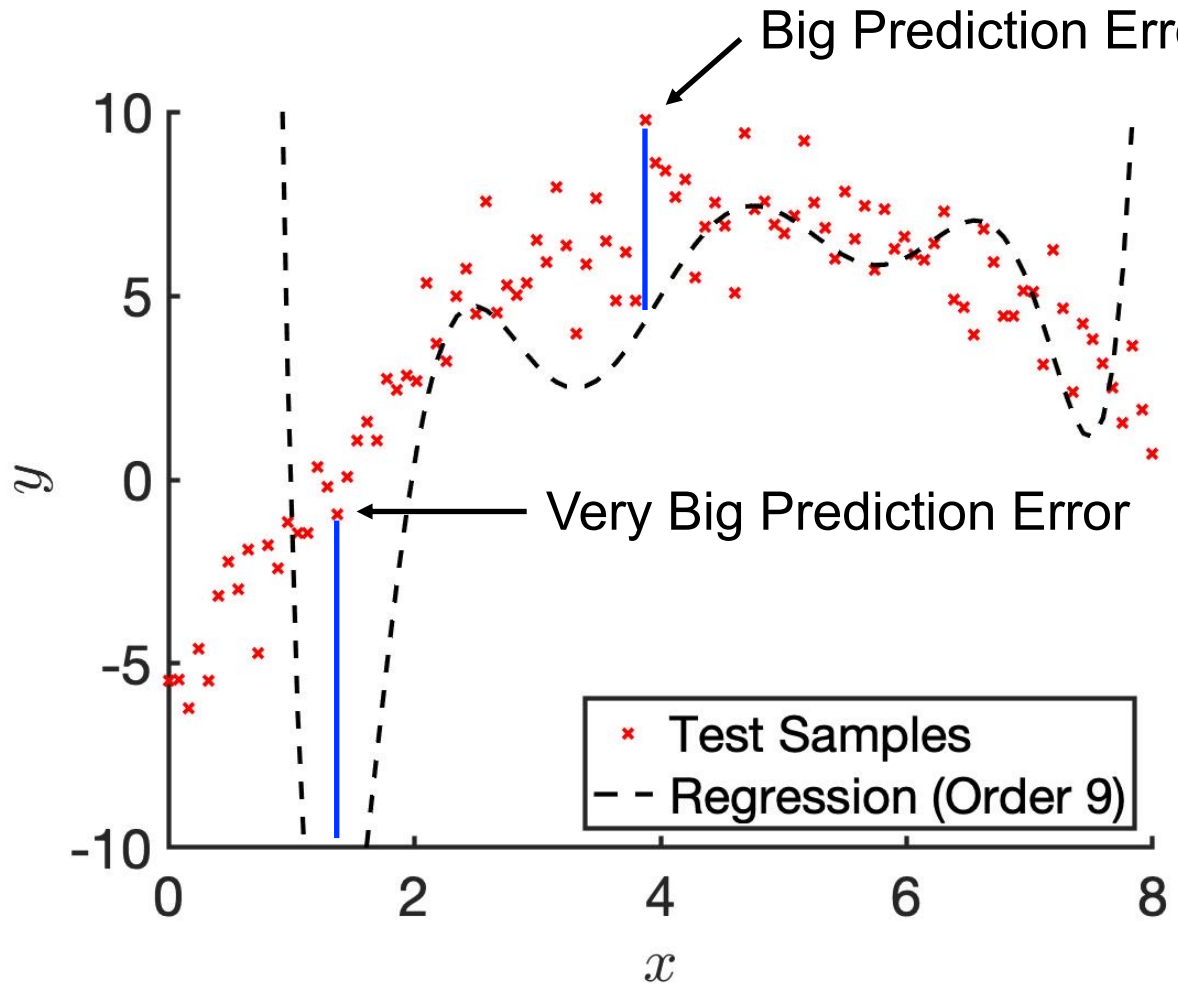
	Training Set Fit	Test Set Fit
Order 9	Good	

Overfitting Example



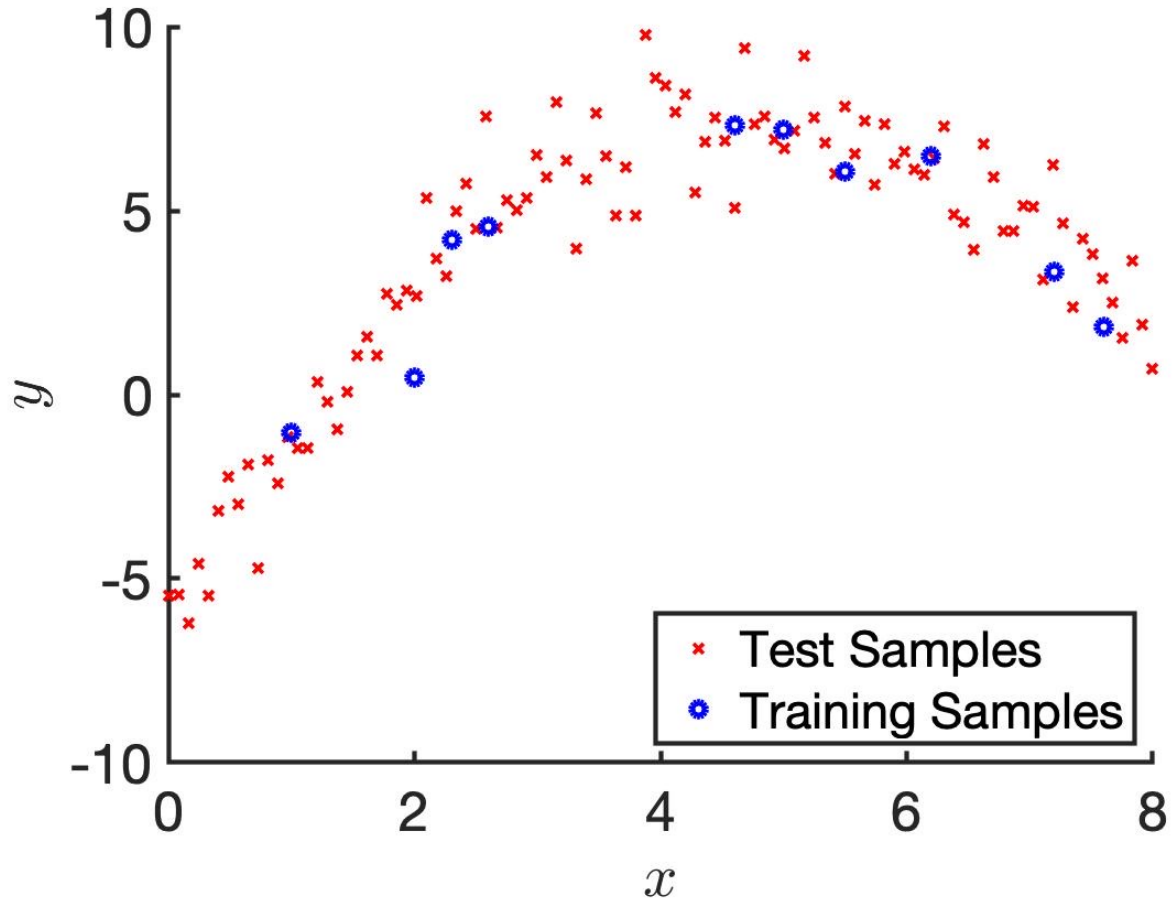
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Overfitting Example



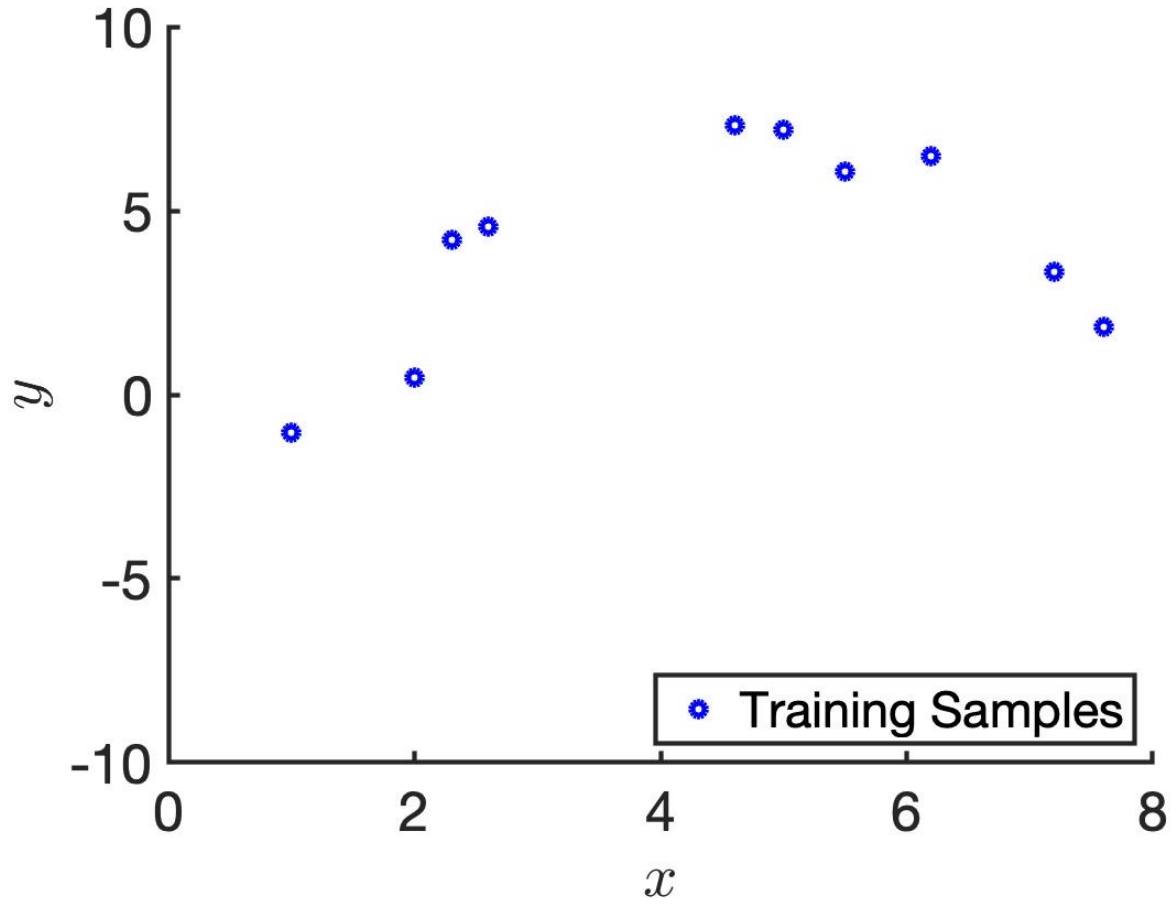
- If we take one of the blue lines and compute the square of its length, this is called “squared error” for that particular data point
- If we average squared errors across all the red crosses, it’s called mean squared error (MSE) in the test set

Underfitting Example



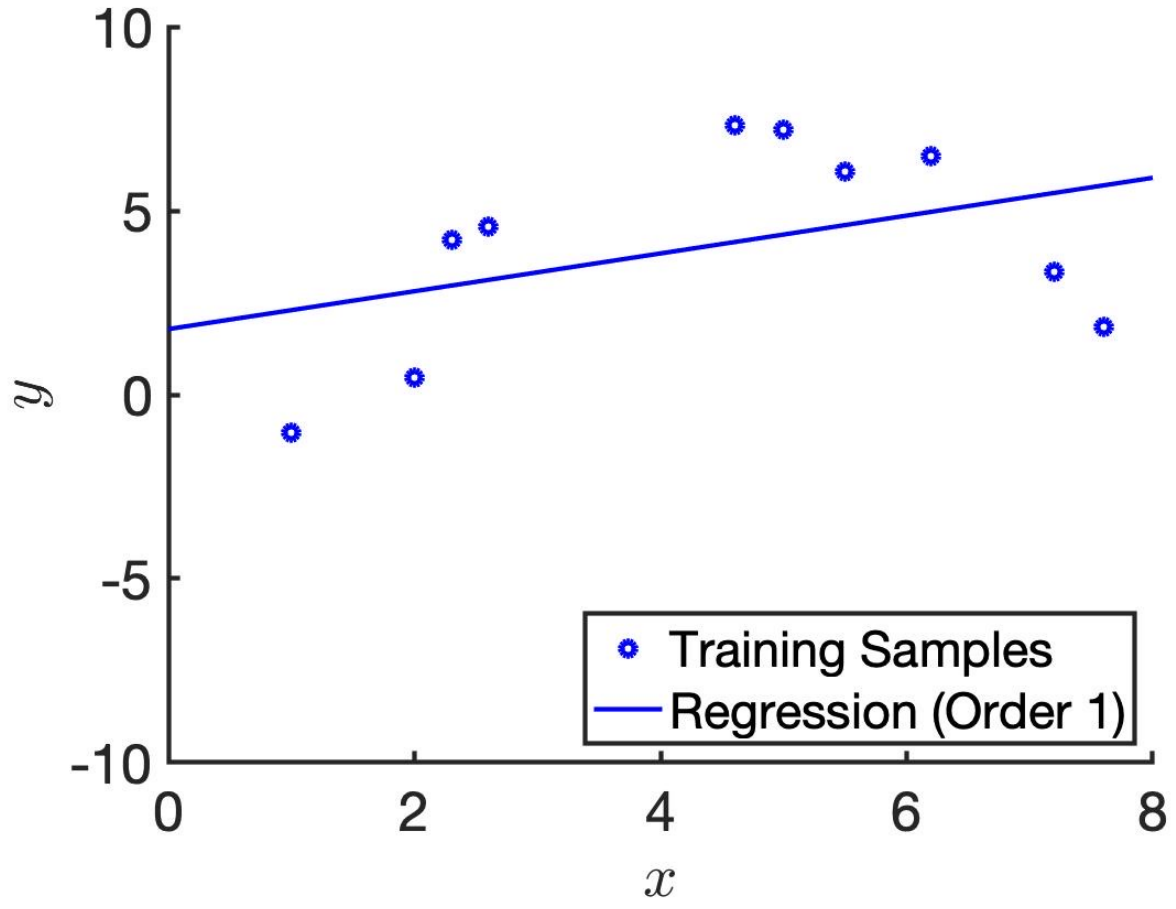
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Underfitting Example



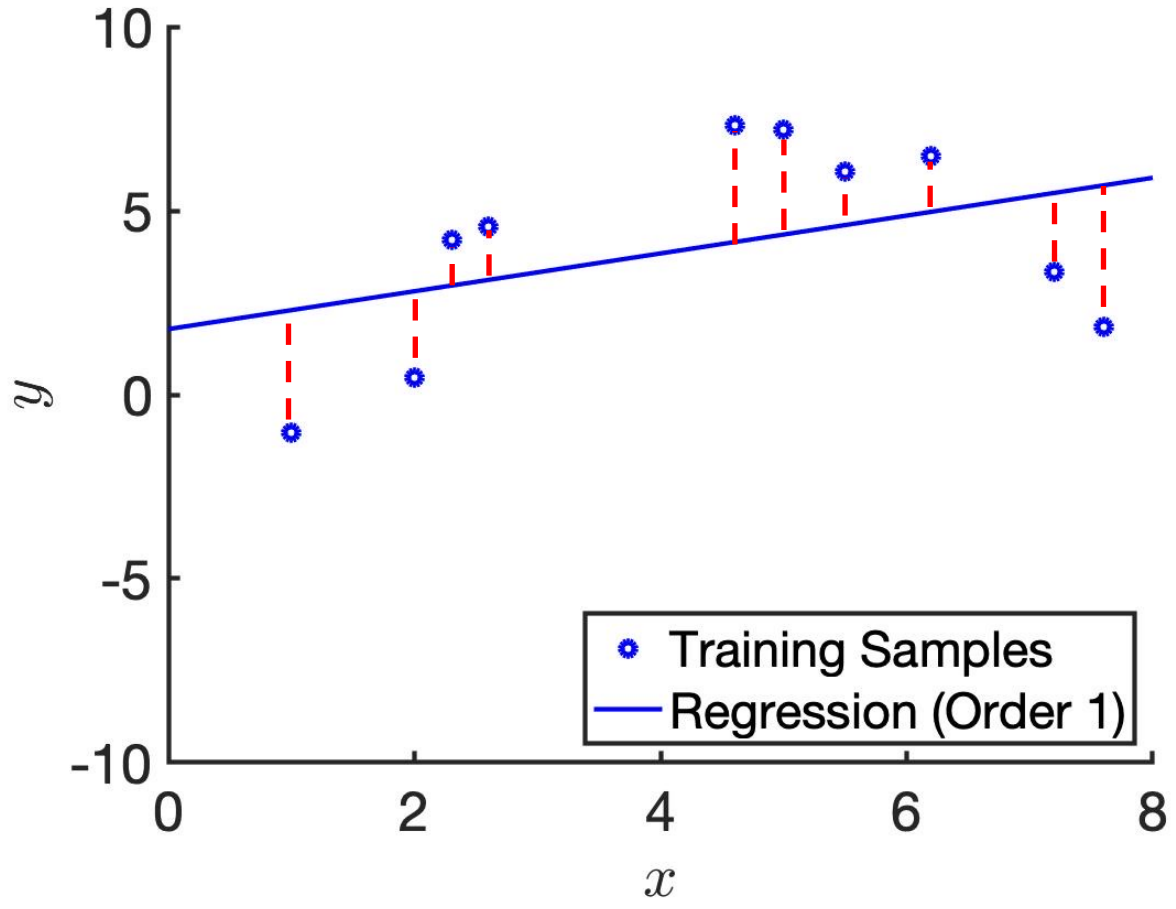
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Underfitting Example



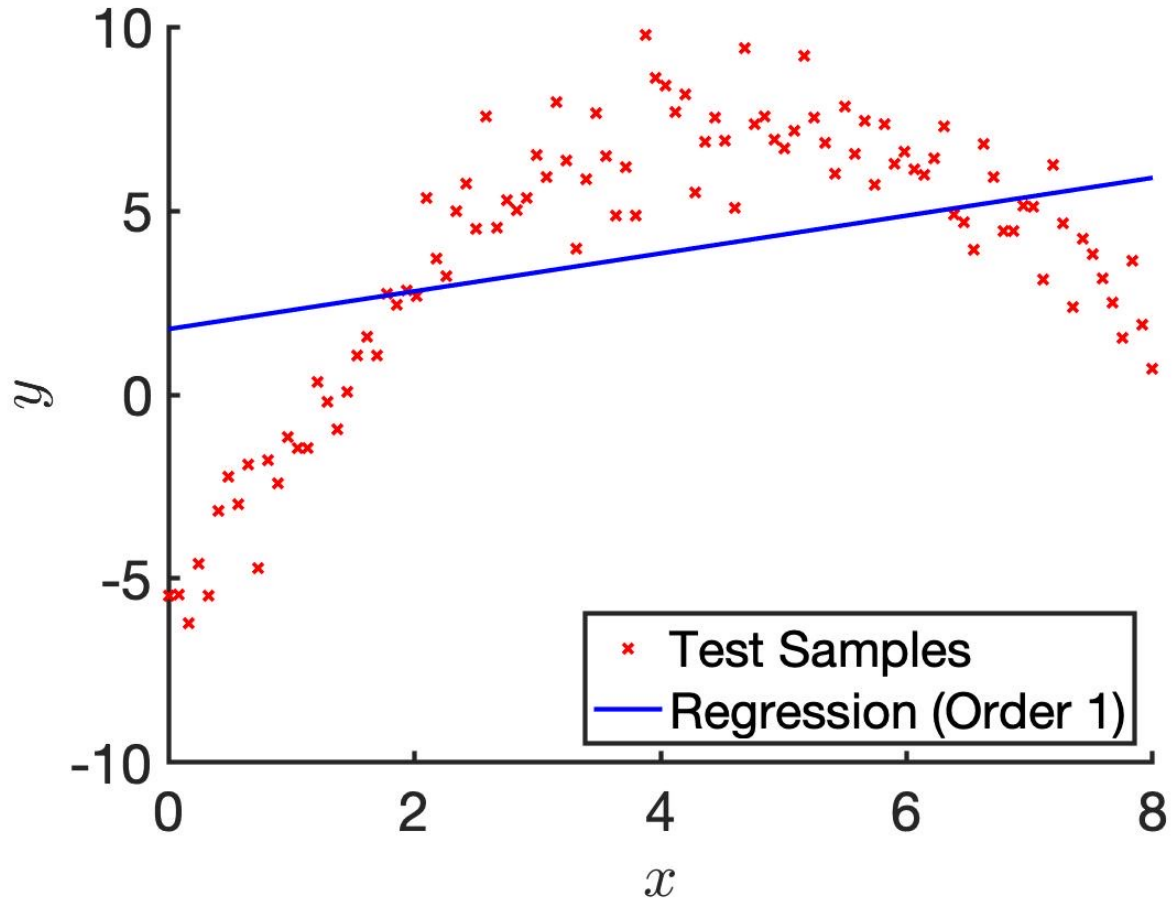
	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1		

Underfitting Example



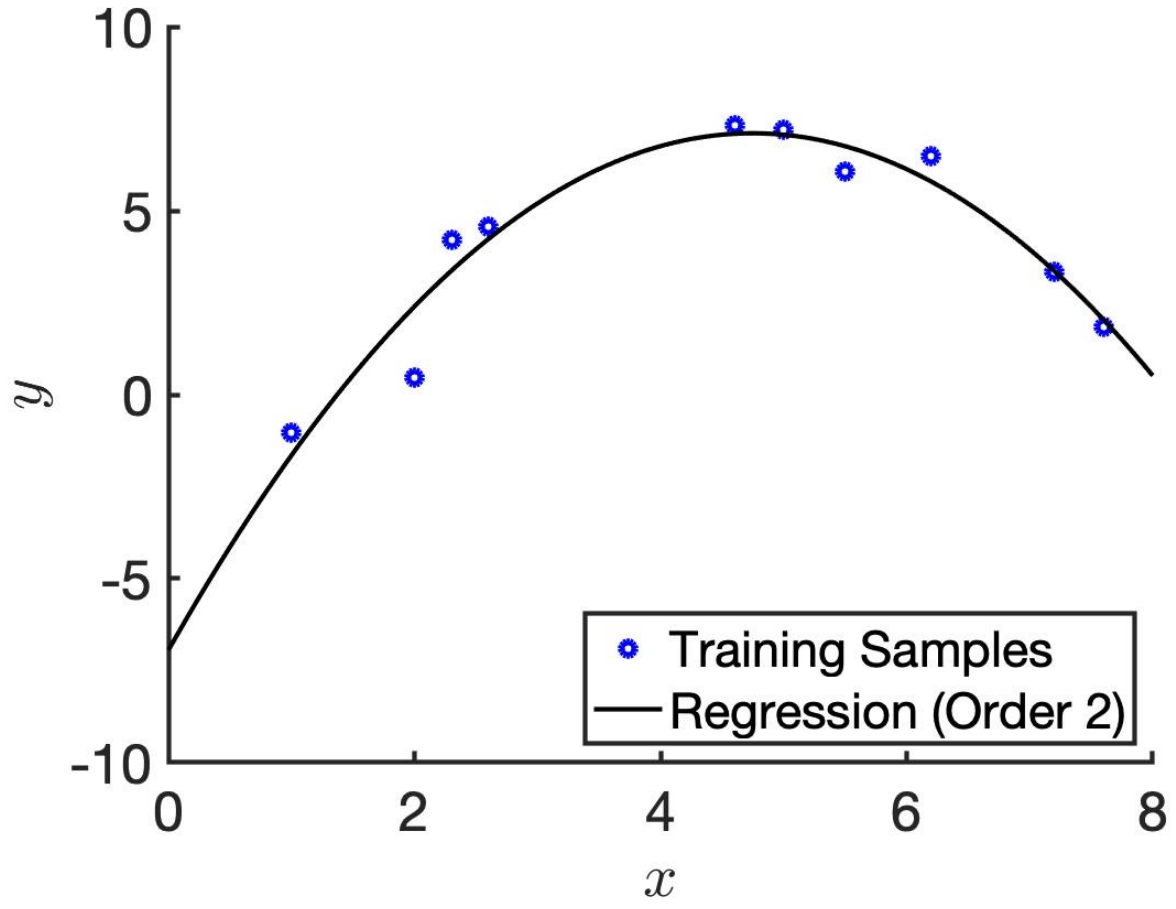
	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	

Underfitting Example



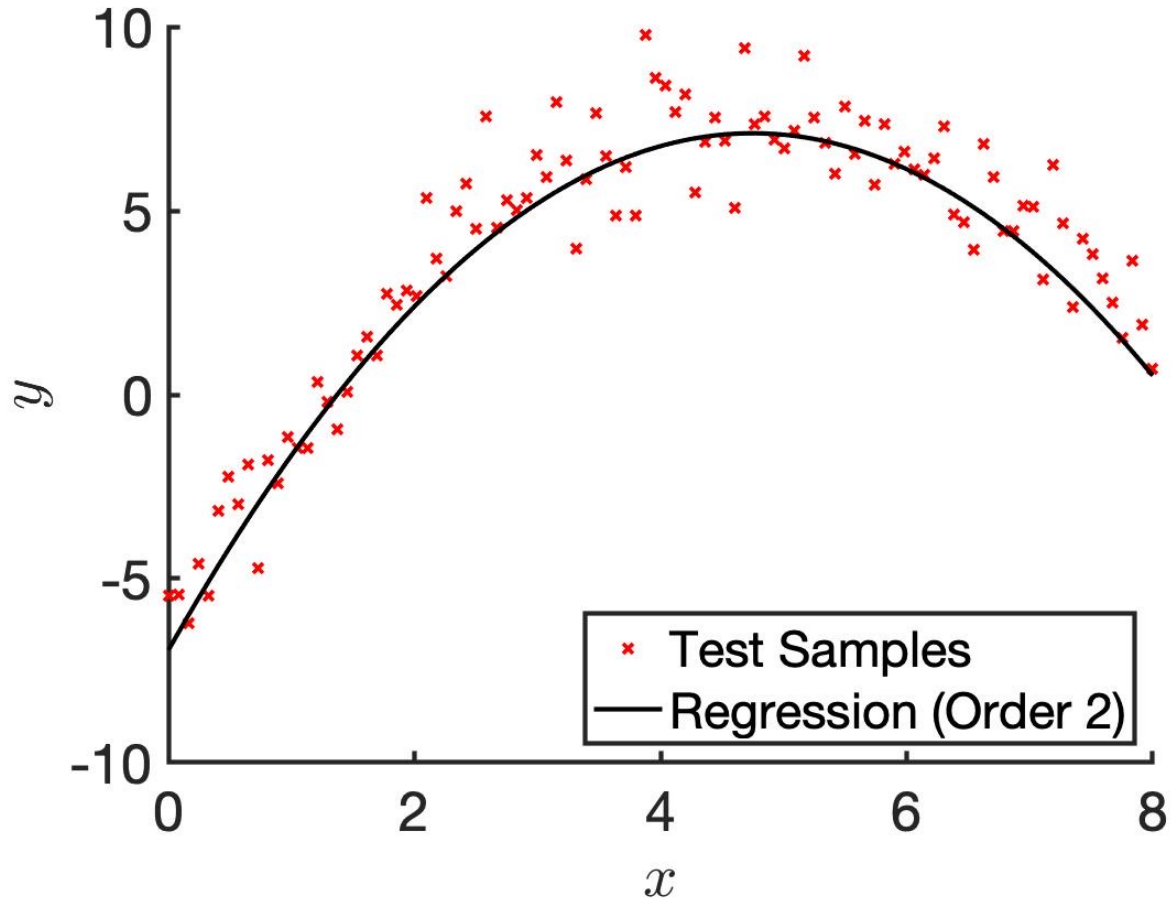
	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad

“Just Nice”



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	

“Just Nice”



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good

Overfitting & Underfitting

Overfitting →

Underfitting →

	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 1	Bad	Bad
Order 2	Good	Good

Overfitting & Underfitting

- Overfitting occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly

Overfitting & Underfitting

- Overfitting occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- Reason 1
  - Model is too complex for the data
  - Previous example: Fit order 9 polynomial to 10 data points

Overfitting & Underfitting

- Overfitting occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- Reason 1
  - Model is too complex for the data
  - Previous example: Fit order 9 polynomial to 10 data points
- Reason 2
  - Too many features but number of training samples too small
  - Even linear model can overfit, e.g., linear model with 9 input features (i.e., w is 10-D) and 10 data points in training set => data might not be enough to estimate 10 unknowns well

Overfitting & Underfitting

- Overfitting occurs when model predicts the training data well, but predicts new data (e.g., from test set) poorly
- Reason 1
  - Model is too complex for the data
  - Previous example: Fit order 9 polynomial to 10 data points
- Reason 2
  - Too many features but number of training samples too small
  - Even linear model can overfit, e.g., linear model with 9 input features (i.e., w is 10-D) and 10 data points in training set => data might not be enough to estimate 10 unknowns well
- Solutions
  - Use simpler models (e.g., lower order polynomial)
  - Use regularization (see next part of lecture)

Overfitting & Underfitting

- Underfitting is the inability of trained model to predict the targets in the training set

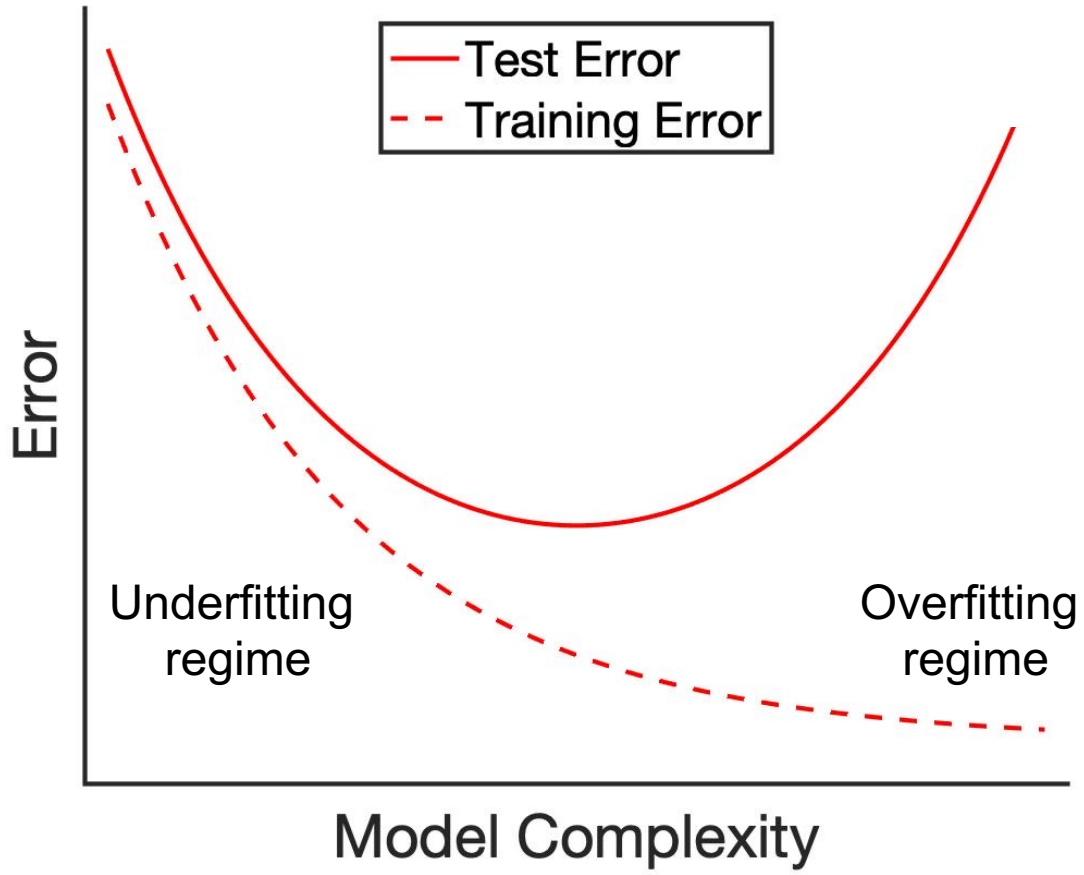
Overfitting & Underfitting

- Underfitting is the inability of trained model to predict the targets in the training set
- Reason 1
  - Model is too simple for the data
  - Previous example: Fit order 1 polynomial to 10 data points that came from an order 2 polynomial
  - Solution: Try more complex model

Overfitting & Underfitting

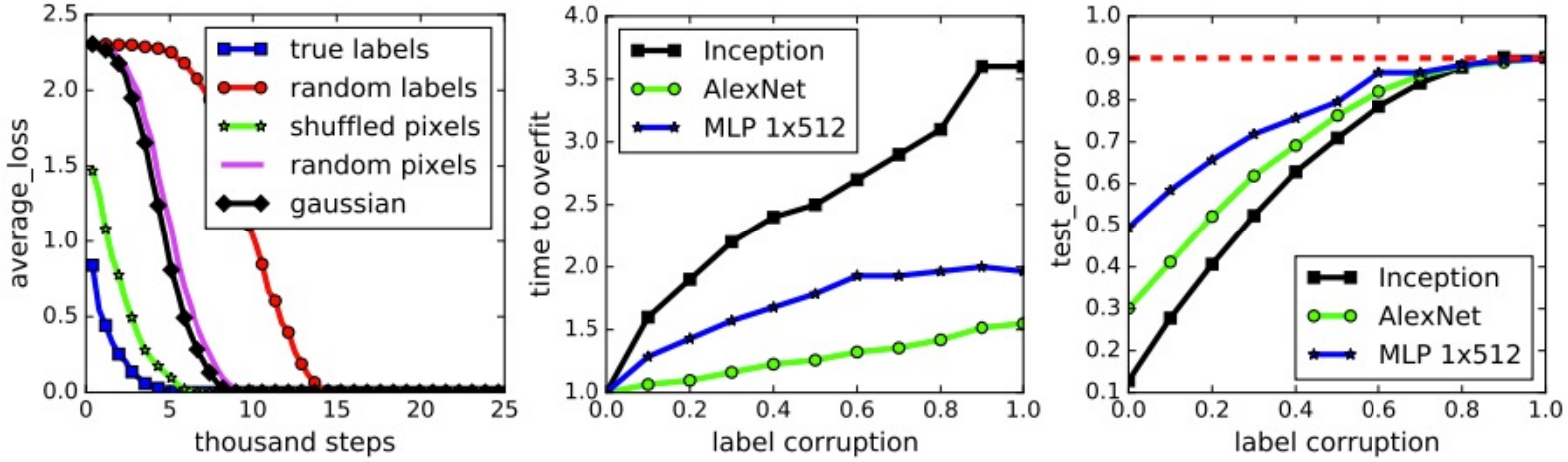
- Underfitting is the inability of trained model to predict the targets in the training set
- Reason 1
  - Model is too simple for the data
  - Previous example: Fit order 1 polynomial to 10 data points that came from an order 2 polynomial
  - Solution: Try more complex model
- Reason 2
  - Features are not informative enough
  - Solution: Try to develop more informative features

Overfitting / Underfitting Schematic



Deep neural networks easily fit random labels.

In a first set of experiments, we train several standard architectures on a copy of the data where the true labels were replaced by random labels. More precisely, when trained on a completely random labeling of the true data, neural networks achieve 0 training error. The test error, of course, is no better than random chance as there is no correlation between the training labels and the test labels. In other words, by randomizing labels alone we can force the generalization error of a model to jump up considerably without changing the model, its size, hyperparameters, or the optimizer. We establish this fact for several different standard architectures trained on the CIFAR10 and ImageNet classification benchmarks.



(a) learning curves (b) convergence slowdown (c) generalization error growth

Figure 1: Fitting random labels and random pixels on CIFAR10. (a) shows the training loss of various experiment settings decaying with the training steps. (b) shows the relative convergence time with different label corruption ratio. (c) shows the test error (also the generalization error since training error is 0) under different label corruptions.

1. The effective capacity of neural networks is sufficient for memorizing the entire data set.
2. Even optimization on random labels remains easy. In fact, training time increases only by a small constant factor compared with training on the true labels.
3. Randomizing labels is solely a data transformation, leaving all other properties of the learning problem unchanged.

UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION, ICLR 2017.

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- For sufficiently large positive λ, matrix inverse above becomes "well-posed" and can be inverted (Motivation 1)

Regularization

- Regularization is an umbrella term that includes methods forcing learning algorithm to build less complex models.
- Motivation 1: Solve an ill-posed problem
  - For example, estimate 10th order polynomial with just 5 datapoints
- Motivation 2: Reduce overfitting
- For example, in previous lecture, we added $\lambda \mathbf{w}^T \mathbf{w}$:

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

- Minimizing with respect to $\mathbf{w}$, primal solution is

$$\hat{\mathbf{w}} = (\mathbf{P}_{train}^T \mathbf{P}_{train} + \lambda \mathbf{I})^{-1} \mathbf{P}_{train}^T \mathbf{y}_{train}$$

- $\hat{\mathbf{w}}$ might also perform better in test set, i.e., reduces overfitting (Motivation 2) – will show example later

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Loss function quantifying data fitting error in training set

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}} (\mathbf{P}\mathbf{w} - \mathbf{y})^T (\mathbf{P}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Loss function quantifying data fitting error in training set

Regularization

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w}=w_0^2+w_1^2+\cdots+w_d^2$

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{P}\mathbf{w}-\mathbf{y})^T(\mathbf{P}\mathbf{w}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ — L2 - Regularization

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{Pw}-\mathbf{y})^T(\mathbf{Pw}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w} = w_0^2 + w_1^2 + \cdots + w_d^2$ — L2 - Regularization

- Encourage $w_0, \cdots, w_d$ to be small (also called shrinkage or weight-decay)

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{Pw}-\mathbf{y})^T(\mathbf{Pw}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w}=w_0^2+w_1^2+\cdots+w_d^2$ — L2 - Regularization

- Encourage $w_0,\cdots,w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{Pw}-\mathbf{y})^T(\mathbf{Pw}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w}=w_0^2+w_1^2+\cdots+w_d^2$ — L2 - Regularization
- Encourage $w_0,\cdots,w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}}\,\mathbf{Data\text{-}Loss}(\mathbf{w})+\lambda\mathbf{Regularization}(\mathbf{w})$$

Regularization

- Consider minimization from previous slide

$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{Pw}-\mathbf{y})^T(\mathbf{Pw}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w}=w_0^2+w_1^2+\cdots+w_d^2$ — L2 - Regularization
- Encourage $w_0,\cdots,w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}}\,\mathbf{Data\text{-}Loss}(\mathbf{w})+\lambda\mathbf{Regularization}(\mathbf{w})$$

- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data

Regularization

- Consider minimization from previous slide

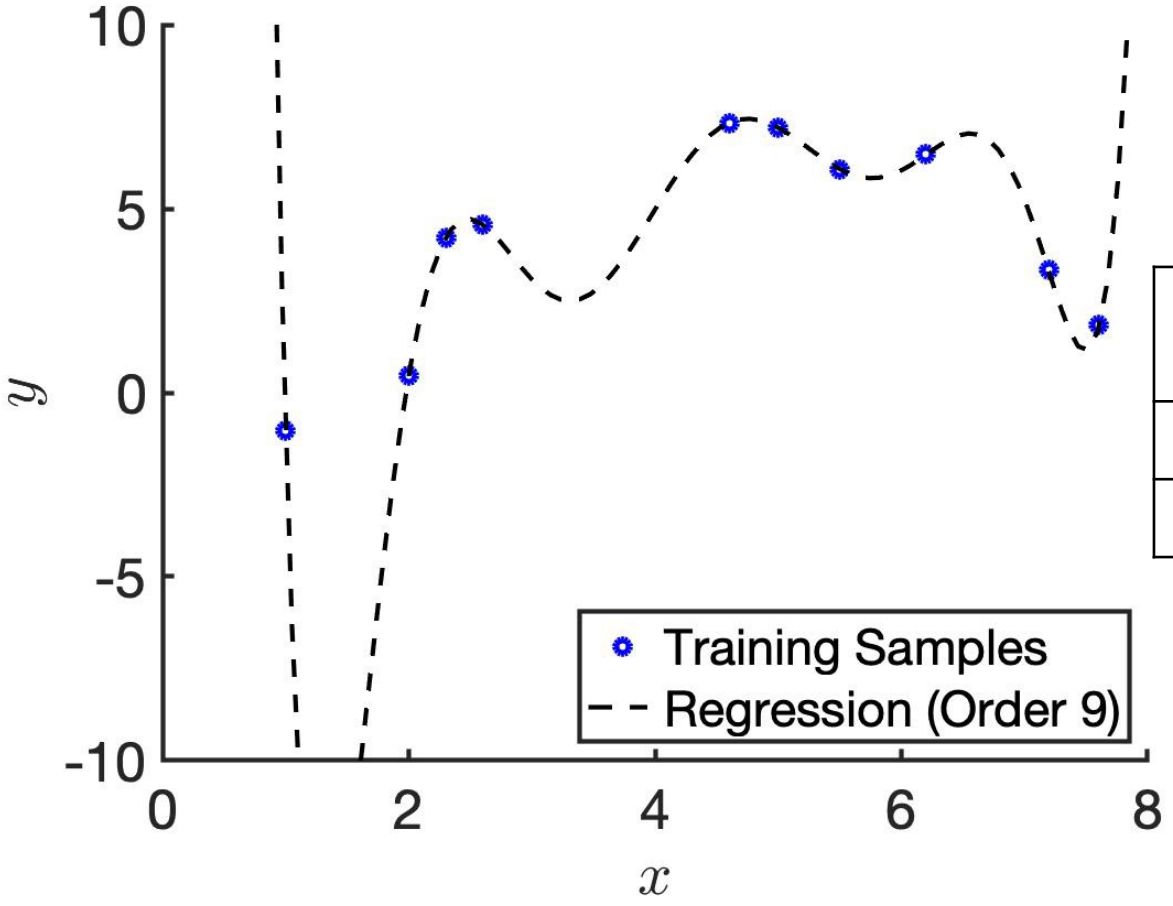
$$\underset{\mathbf{w}}{\operatorname{argmin}}(\mathbf{Pw}-\mathbf{y})^T(\mathbf{Pw}-\mathbf{y})+\lambda\mathbf{w}^T\mathbf{w}$$

- $\mathbf{w}^T\mathbf{w}=w_0^2+w_1^2+\cdots+w_d^2$ — L2 - Regularization
- Encourage $w_0,\cdots,w_d$ to be small (also called shrinkage or weight-decay) => constrain model complexity
- More generally, most machine learning algorithms can be formulated as the following optimization problem

$$\underset{\mathbf{w}}{\operatorname{argmin}}\,\mathbf{Data\text{-}Loss}(\mathbf{w})+\lambda\mathbf{Regularization}(\mathbf{w})$$

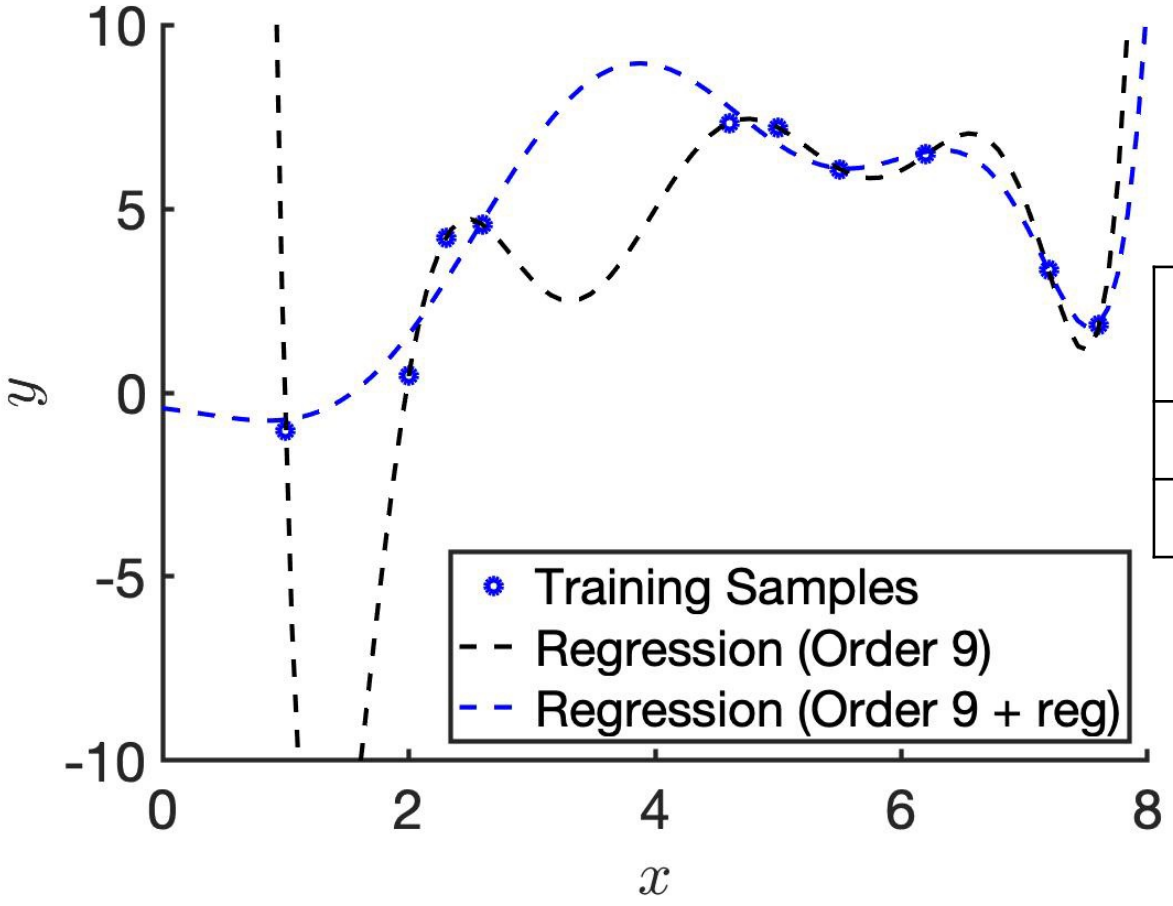
- **Data-Loss(w)** quantifies fitting error to training set given parameters **w:** smaller error => better fit to training data
- **Regularization(w)** penalizes more complex models

Regularization Example



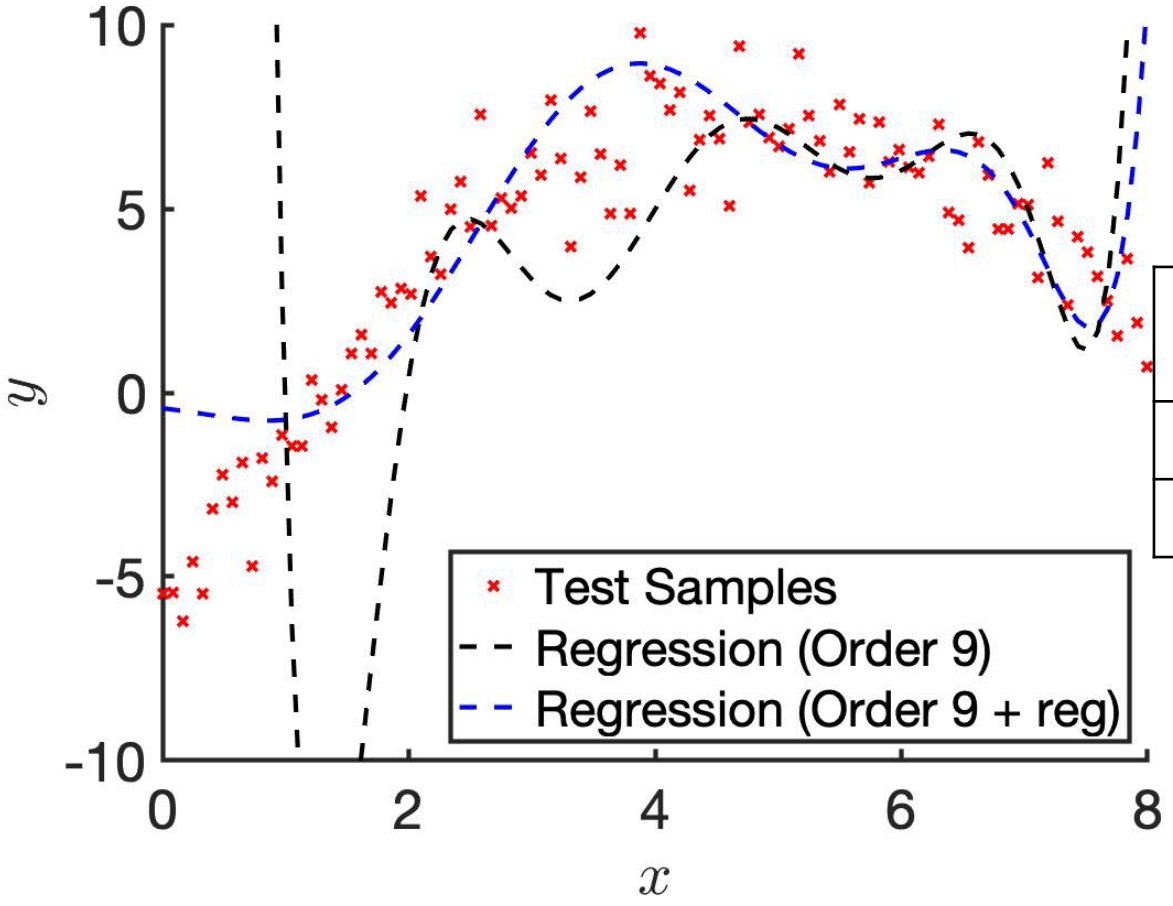
	Training Set Fit	Test Set Fit
Order 9	Good	Bad

Regularization Example



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 9 + $\lambda = 1$	Good	

Regularization Example



	Training Set Fit	Test Set Fit
Order 9	Good	Bad
Order 9 + $\lambda = 1$	Good	Good

Recall: Soft Margin Classification

If training data is **NOT linearly separable**, *slack variables* ξ_i can be added to allow misclassification of difficult/noisy examples.

Resulting margin called ***'Soft'***.

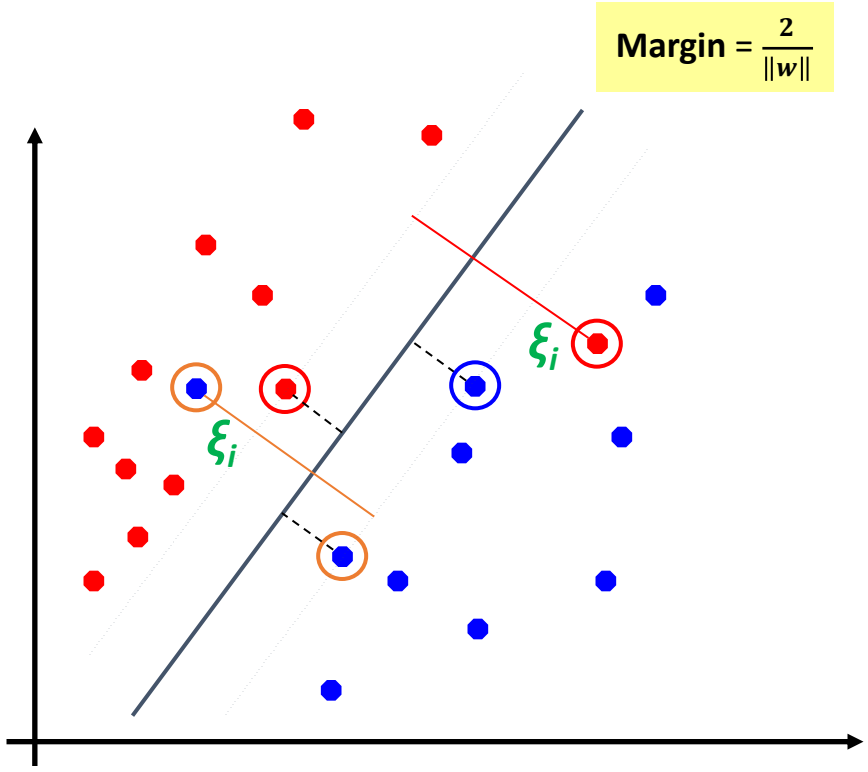
move some points to where they belong; at a cost / penalty:

$$\min_{\mathbf{w}, b, \xi_i > 0} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i$$

$$\text{s.t. } \forall i: y_i\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b\right) \geq 1 - \xi_i$$

parameter *C* can be viewed as a way to control overfitting i.e., a **regularization** term.

small C allows constraints to be easily ignored → **large margin**
large C makes constraints hard to ignore → **narrow margin**
C = ∞ enforces all constraints → **hard margin**.



Recall: K Nearest Neighbour Classifier

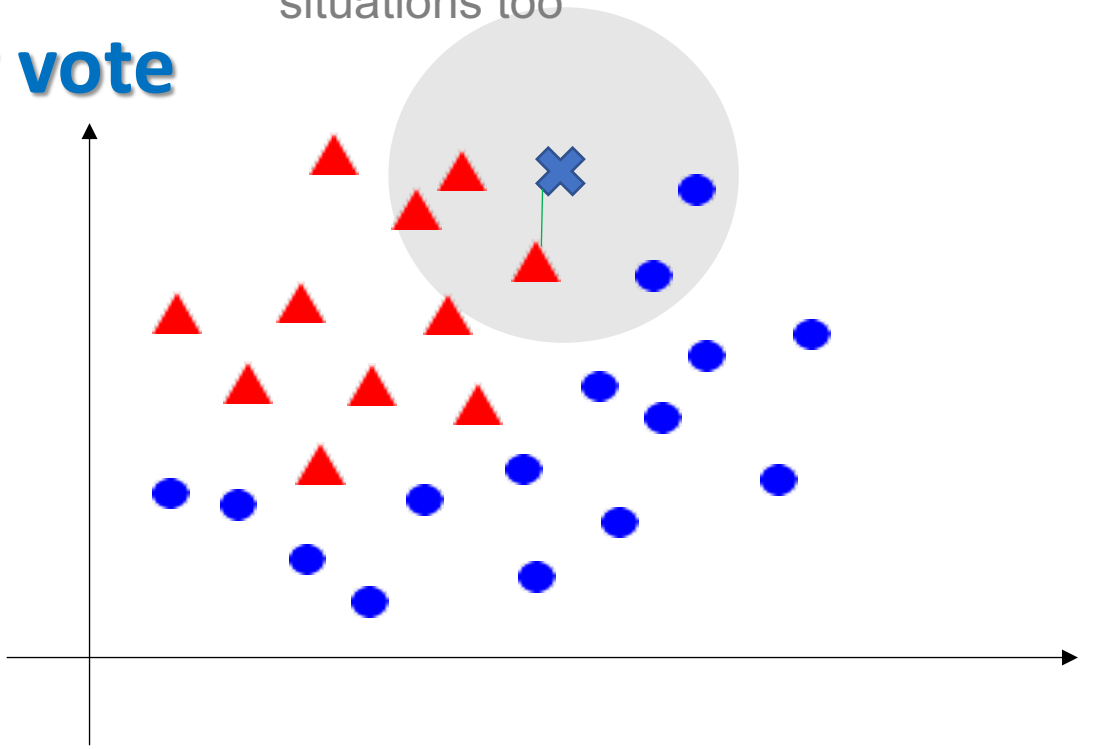
For each test point x to be classified, find the **K** nearest samples in the training data

Classify x according to the **majority vote** of their class labels

Aim of supervised learning:

to do well on test data which is **not known** during learning..

e.g. K = 5
applicable to multi-class situations too



Hyperparameters..

What‘s the **best value** of **K** to use?

What's the **best distance measure** to use?

These **hyperparameter** choices are **SET** rather than learnt:

very problem-dependent..

must try them all out … see what works best

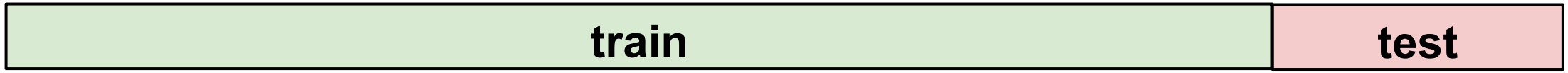
Setting.. hyperparameters

Approach#1: choose hyperparameters that work best on the data..

BAD: K = 1 always works perfectly on training data.

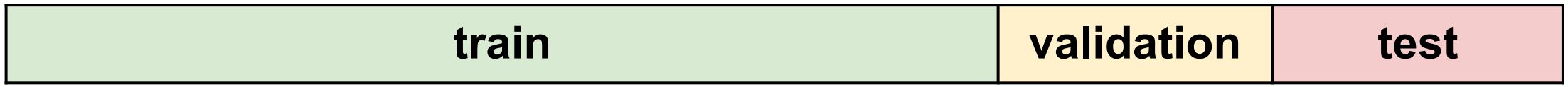
Approach#2: Split data into **train** & **test**, choose hyperparameters that work best on test data

No idea how algorithm will perform on new data



Approach #3: Split data into **train**, **validation** & **test**; choose hyperparameters on validation and evaluate on test (*finally*).

Better!



Setting.. hyperparameters

dataset

Approach #4 : **Cross-Validation**: Split data into **folds**; each fold tried out as validation and the **results averaged**.

fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test
fold 1	fold 2	fold 3	fold 4	fold 5	test

Useful for small datasets, but not used too frequently in Deep Learning.

Setting.. hyperparameters

TRAINING SET: set of examples used for learning; to ***fit the parameters*** *of the classifier*. In MLP: training set used to find "optimal" weights with back-propagation rule.

VALIDATION SET: set of examples used to ***tune the parameters*** *of a classifier.* In MLP: the validation set used to find the "optimal" number of hidden units *or* determine a stopping point for back-propagation algorithm. Error rate estimate of final model on validation set is biased (< *true error rate*) since its is used to select final model.

TEST SET: set of examples used **ONLY to assess performance of a fully-trained classifier**. In MLP: the test set used to estimate the error rate after final model is chosen (MLP size & actual weights). After assessing final model on test set, MUST NOT tune the model any further!

Nearest Neighbour Classifier

As ***K*** increases, classification boundary becomes smoother but training error can increase. Choose (learn) ***K*** by cross-validation:

- split training data into **training** & validation
- choose ***hyperparameters*** on validation and evaluate **ONCE** on **test** data

ADVANTAGES: ***K*** –NN is a simple & effective classification procedure; applicable for multi-class classification. With just **a single parameter *K*** which is easily tuned by cross-validation, its decision surfaces are non-linear and quality of predictions improves with more training data.

DISADVANTAGES: **High computational cost** as must store & **search through the entire training set at test time** & etc. Ideally, a classifier should be **FAST** for prediction.. but **SLOW** for training is fine. **NN hardly used for images**.