

CSE 471: MACHINE LEARNING

**Learning from Examples (Continued)**

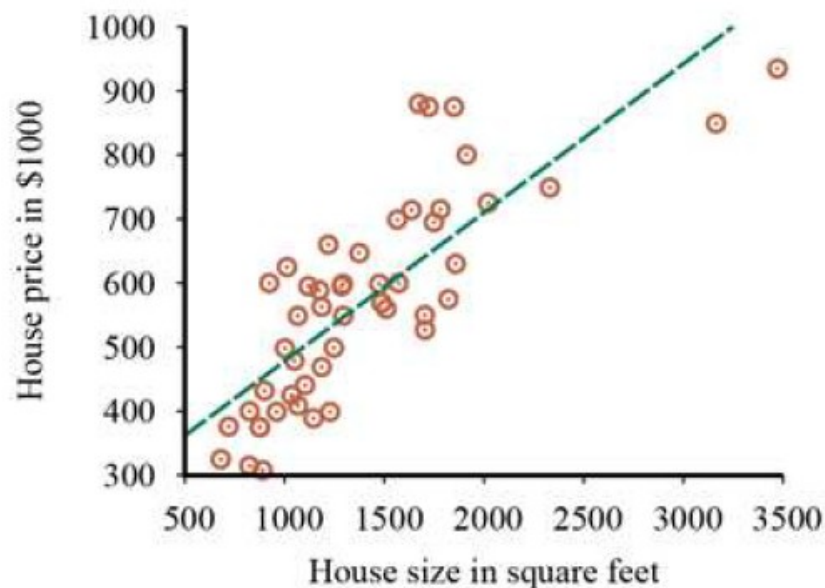
# Outline



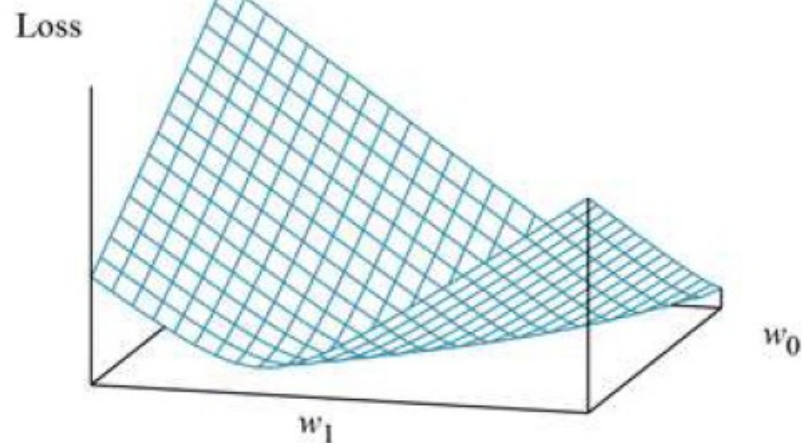
- Linear Regression and Classification
- Ensemble Learning
- Model Selection and Optimization
- The Theory of Learning
- Nonparametric Models
- Developing Machine Learning Systems

# Linear Regression & Classification

- $h_w(x) = w_0 + w_1 x$
- Squared error =  $\sum_j (y_j - h(x_j))^2$  [L2 loss function]
- Use Gradient Descent to minimize squared error



(a)



(b)

# Linear Regression

- $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \operatorname{Loss}(\mathbf{h}_{\mathbf{w}})$
- Set partial derivatives to 0 to obtain the minima

$$\frac{\partial}{\partial w_0} \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 = 0$$

$$\frac{\partial}{\partial w_1} \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2 = 0$$

# Linear Regression

$$w_0 = \left( \sum y_j - w_1 \left( \sum x_j \right) \right) / N$$

$$w_1 = \frac{N \left( \sum x_j y_j \right) - \left( \sum x_j \right) \left( \sum y_j \right)}{N \left( \sum x_j^2 \right) - \left( \sum x_j \right)^2}$$

# Gradient Descent

**$\mathbf{w} \leftarrow$  any point in the parameter space**

**while not converged do**

**for each  $w_i$  in  $\mathbf{w}$  do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

□  $\alpha$  - Learning rate

□ Constant

□ Decaying

# The partial derivatives

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2 = 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0)).\end{aligned}$$

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))$$

$$\frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

# The learning rules

*For single  
example*

$$w_0 \leftarrow w_0 + \alpha (y - h_{\mathbf{w}}(x))$$

$$w_1 \leftarrow w_1 + \alpha (y - h_{\mathbf{w}}(x)) \times x$$

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

*For Entire  
training set*



# Stochastic gradient descent (SGD)

- Use a random sample of examples at each step
  - ▣ A minibatch of  $m$  out of  $N$  samples
- Convergence of minibatch SGD is not strictly guaranteed
  - ▣ It can oscillate around the minimum without settling down
  - ▣ Decaying learning rate can guarantee convergence
- Useful in online learning

# Multivariable Linear Regression

$$h_{\mathbf{w}}(\mathbf{x}_j) = w_0 + w_1x_{j,1} + \cdots + w_nx_{j,n} = w_0 + \sum_i w_ix_{j,i}$$

$$w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(\mathbf{x}_j)) \times x_{j,i}$$

***Gradient Descent Step***

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

***Recall  
[for univariable  
case]***

# Multivariable LR Vectorized

Assuming a dummy input attribute  $x_{i,0} = 1$ ,  
We can vectorize the formulation

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^\top \mathbf{x}_j = \sum_i w_i x_{j,i}$$

**The optimization  
Problem**

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)$$

# Multivariable LR Vectorized

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

- **$\mathbf{\hat{y}}$**  – prediction on all examples
- **$\mathbf{X}$**  - Data matrix. Each row is a multivariable sample
- **$\mathbf{w}$**  - Weight vector

$$L(\mathbf{w}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

- Loss over training dataset

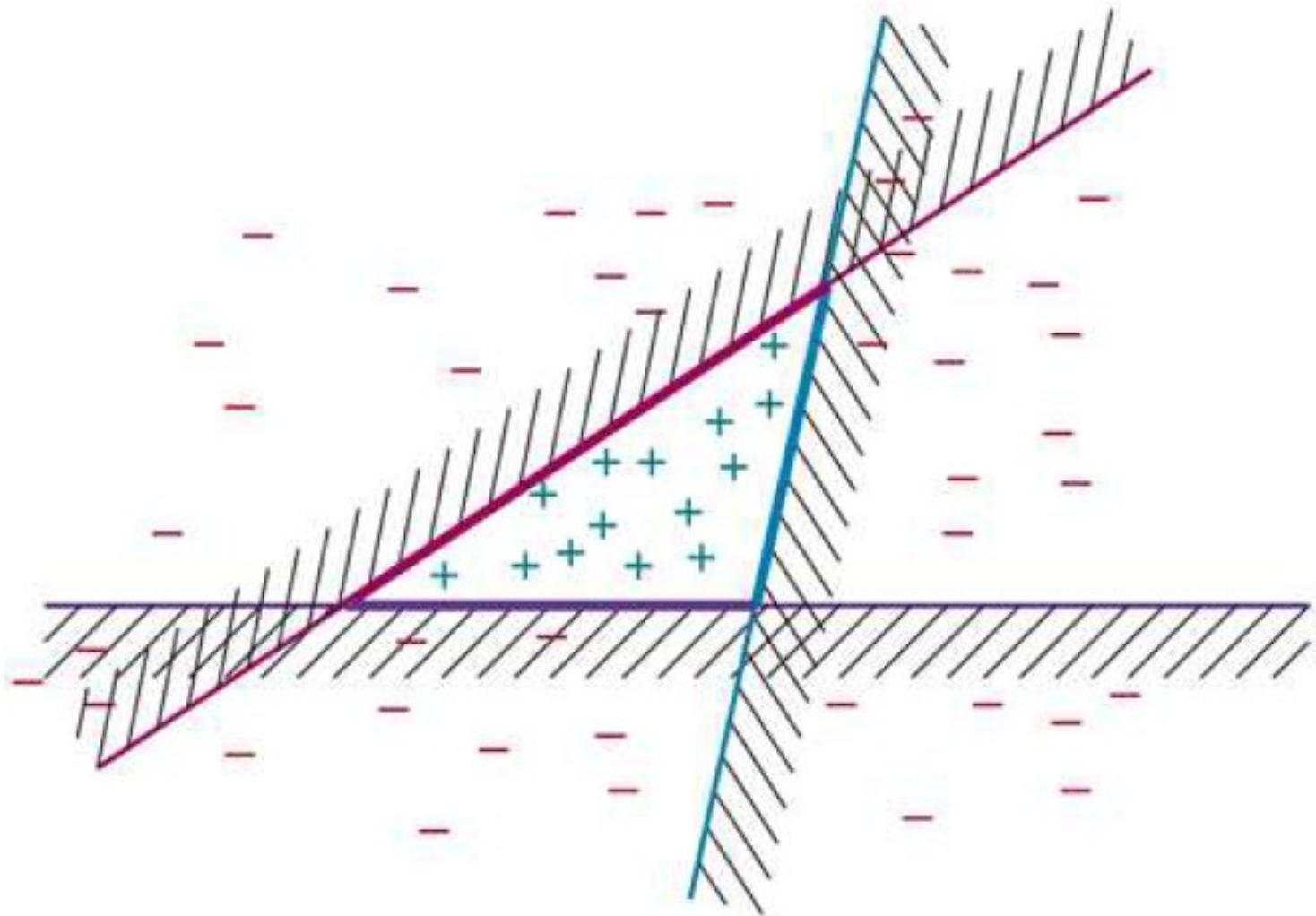
$$\nabla_{\mathbf{w}} L(\mathbf{w}) = 2\mathbf{X}^{\top}(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$$

- Set the gradient to 0

$$\mathbf{w}^* = (\mathbf{X}^{\top}\mathbf{X})^{-1}\mathbf{X}^{\top}\mathbf{y}$$

- **Normal Equation**

# Ensemble Learning

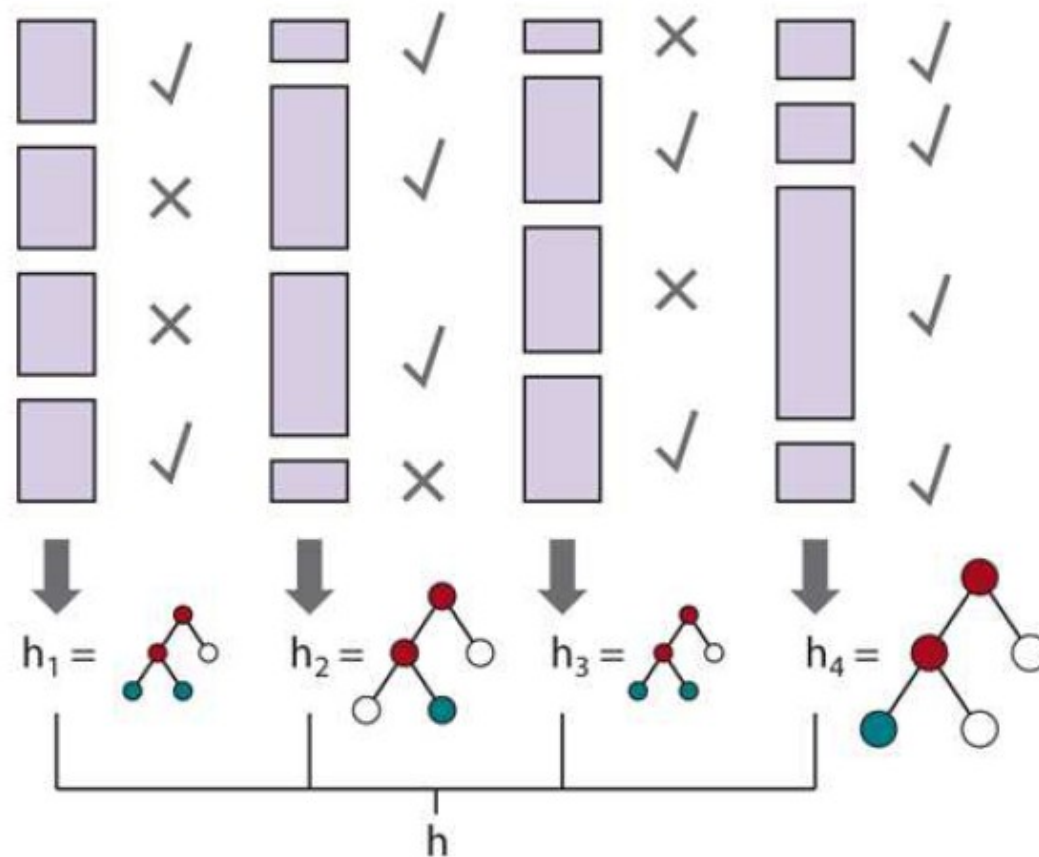


# Ensemble Learning

---

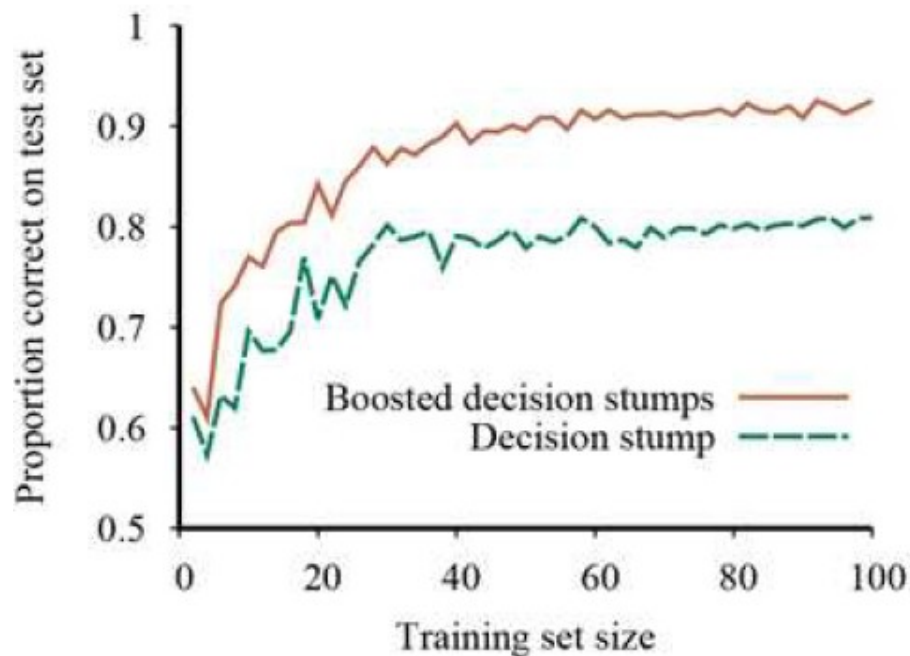
- Boosting
- Bootstrap aggregating (Bagging)
- Random forests
- Stacking

# Boosting

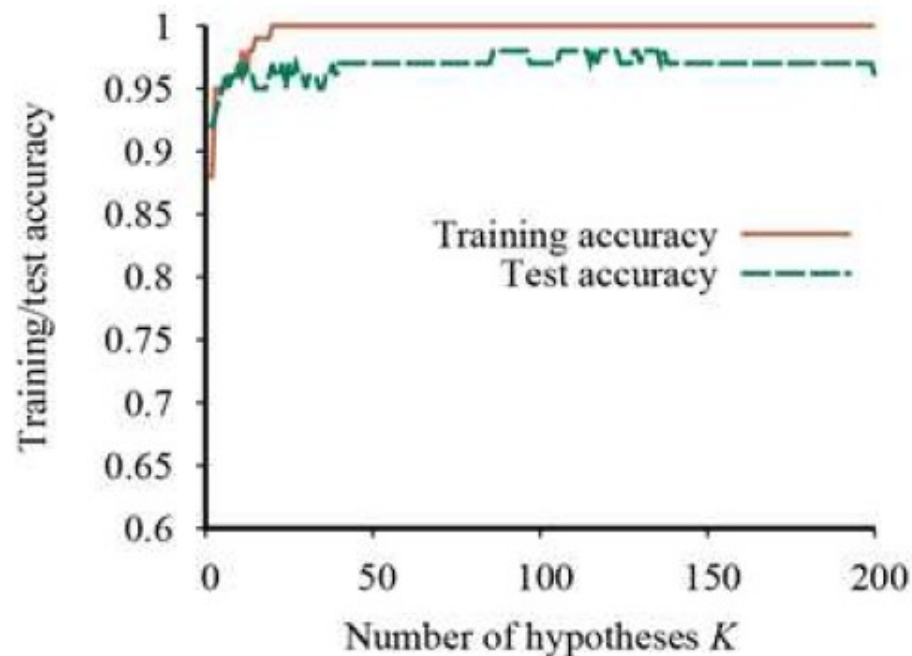


How the boosting algorithm works. Each shaded rectangle corresponds to an example; the height of the rectangle corresponds to the weight. The checks and crosses indicate whether the example was classified correctly by the current hypothesis. The size of the decision tree indicates the weight of that hypothesis in the final ensemble.

# Boosting Decision Stumps



(a)



(b)

- ***A decision stump is a DT with just one test (i.e. only the root)***
- ***Notice that the test set accuracy improves slightly even after the training accuracy reaches 1, i.e., after the ensemble fits the data exactly.***



# AdaBoost

**function** ADABOOST(*examples*, *L*, *K*) **returns** a hypothesis

**inputs:** *examples*, set of *N* labeled examples  $(x_1, y_1), \dots, (x_N, y_N)$

*L*, a learning algorithm

*K*, the number of hypotheses in the ensemble

**local variables:** **w**, a vector of *N* example weights, initially all  $1/N$

**h**, a vector of *K* hypotheses

**z**, a vector of *K* hypothesis weights

$\epsilon \leftarrow$  a small positive number, used to avoid division by zero

**for**  $k = 1$  **to** *K* **do**

$\mathbf{h}[k] \leftarrow L(\text{examples}, \mathbf{w})$

$\text{error} \leftarrow 0$

**for**  $j = 1$  **to** *N* **do**      *// Compute the total error for h[k]*

**if**  $\mathbf{h}[k](x_j) \neq y_j$  **then**  $\text{error} \leftarrow \text{error} + \mathbf{w}[j]$

**if**  $\text{error} > 1/2$  **then break** from loop

$\text{error} \leftarrow \min(\text{error}, 1 - \epsilon)$

**for**  $j = 1$  **to** *N* **do**      *// Give more weight to the examples h[k] got wrong*

**if**  $\mathbf{h}[k](x_j) = y_j$  **then**  $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot \text{error} / (1 - \text{error})$

$\mathbf{w} \leftarrow \text{NORMALIZE}(\mathbf{w})$

$\mathbf{z}[k] \leftarrow \frac{1}{2} \log((1 - \text{error}) / \text{error})$       *// Give more weight to accurate h[k]*

**return**  $\text{Function}(x) : \sum \mathbf{z}_i \mathbf{h}_i(x)$

$$h(\mathbf{x}) = \sum_{i=1}^K z_i h_i(\mathbf{x})$$

# AdaBoost

**function** ADABOOST(*examples*, *L*, *K*) **returns** a hypothesis

**inputs:** *examples*, set of *N* labeled examples  $(x_1, y_1), \dots, (x_N, y_N)$

*L*, a learning algorithm

*K*, the number of hypotheses in the ensemble

**local variables:** **w**, a vector of *N* example weights, initially all  $1/N$

**h**, a vector of *K* hypotheses

**z**, a vector of *K* hypothesis weights

$\epsilon \leftarrow$  a small positive number, used to avoid division by zero


**for**  $k = 1$  **to** *K* **do**

$\mathbf{h}[k] \leftarrow L(\text{examples}, \mathbf{w})$

$\text{error} \leftarrow 0$

**for**  $j = 1$  **to** *N* **do**      *// Compute the total error for h[k]*

**if**  $\mathbf{h}[k](x_j) \neq y_j$  **then**  $\text{error} \leftarrow \text{error} + \mathbf{w}[j]$

**if**  $\text{error} > 1/2$  **then continue** 

$\text{error} \leftarrow \min(\text{error}, 1 - \epsilon)$

**for**  $j = 1$  **to** *N* **do**      *// Give more weight to the examples h[k] got wrong*

**if**  $\mathbf{h}[k](x_j) = y_j$  **then**  $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot \text{error} / (1 - \text{error})$

$\mathbf{w} \leftarrow \text{NORMALIZE}(\mathbf{w})$

$\mathbf{z}[k] \leftarrow \frac{1}{2} \log((1 - \text{error}) / \text{error})$       *// Give more weight to accurate h[k]*

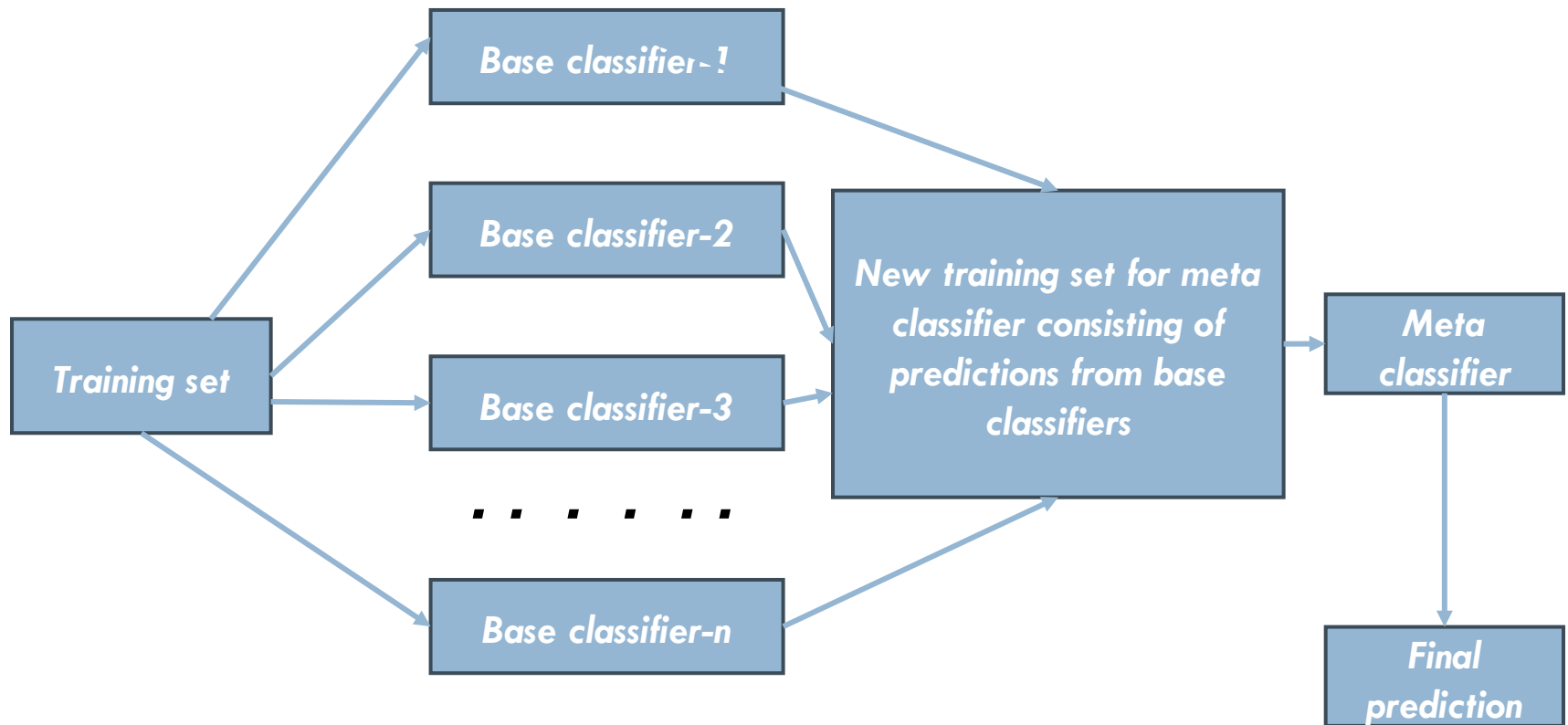
**return**  $\text{Function}(x) : \sum \mathbf{z}_i \mathbf{h}_i(x)$

$$h(\mathbf{x}) = \sum_{i=1}^K z_i h_i(\mathbf{x})$$

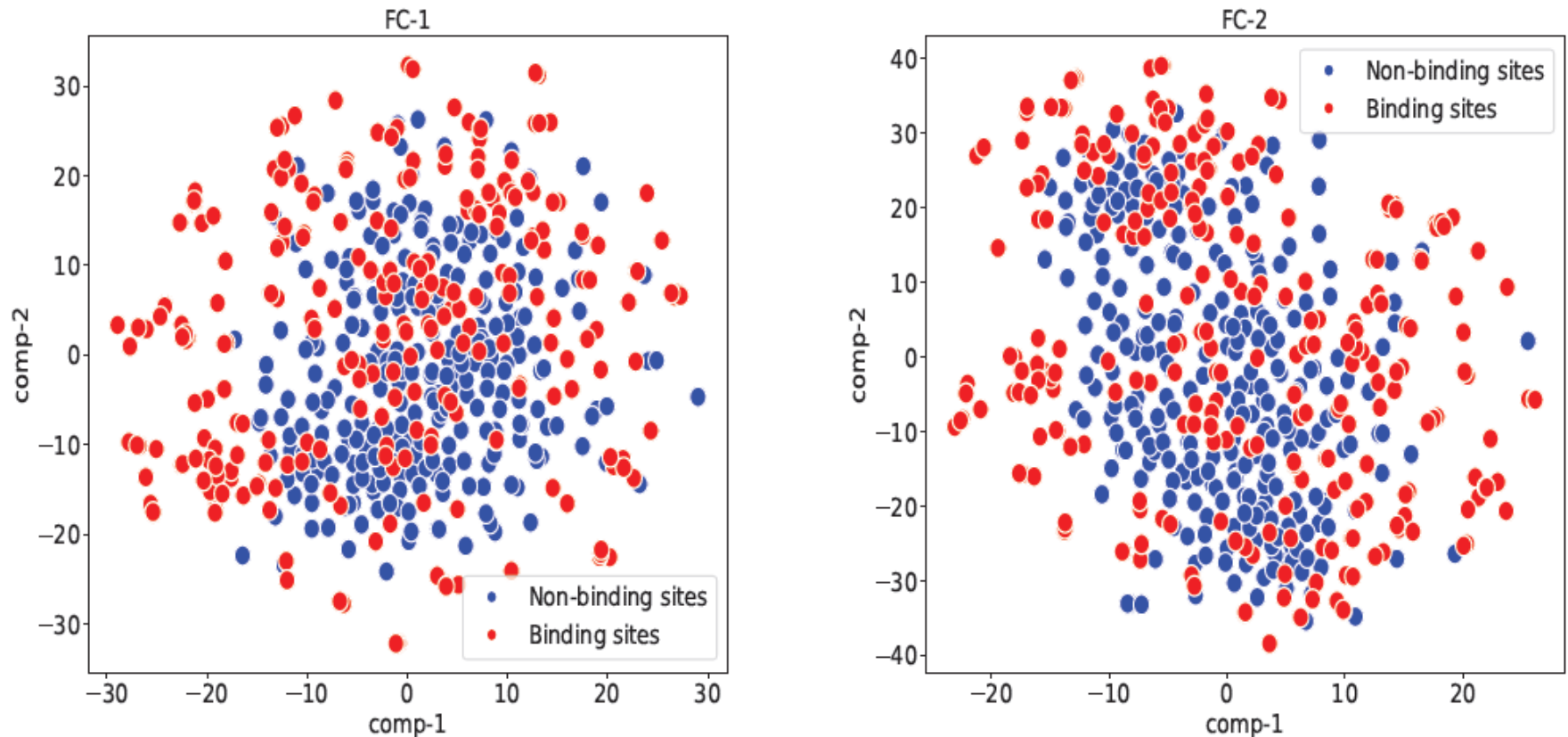
# Stacking

- Base models from different model classes
  - ▣ SVM
  - ▣ Logistic regression
  - ▣ Decision Tree
- Augment feature vector with output of the base models
- Train the ensemble model (say Logistic regression)
  - ▣ **On the Validation Set**
  - ▣ Cross validation can be done

# Stacking

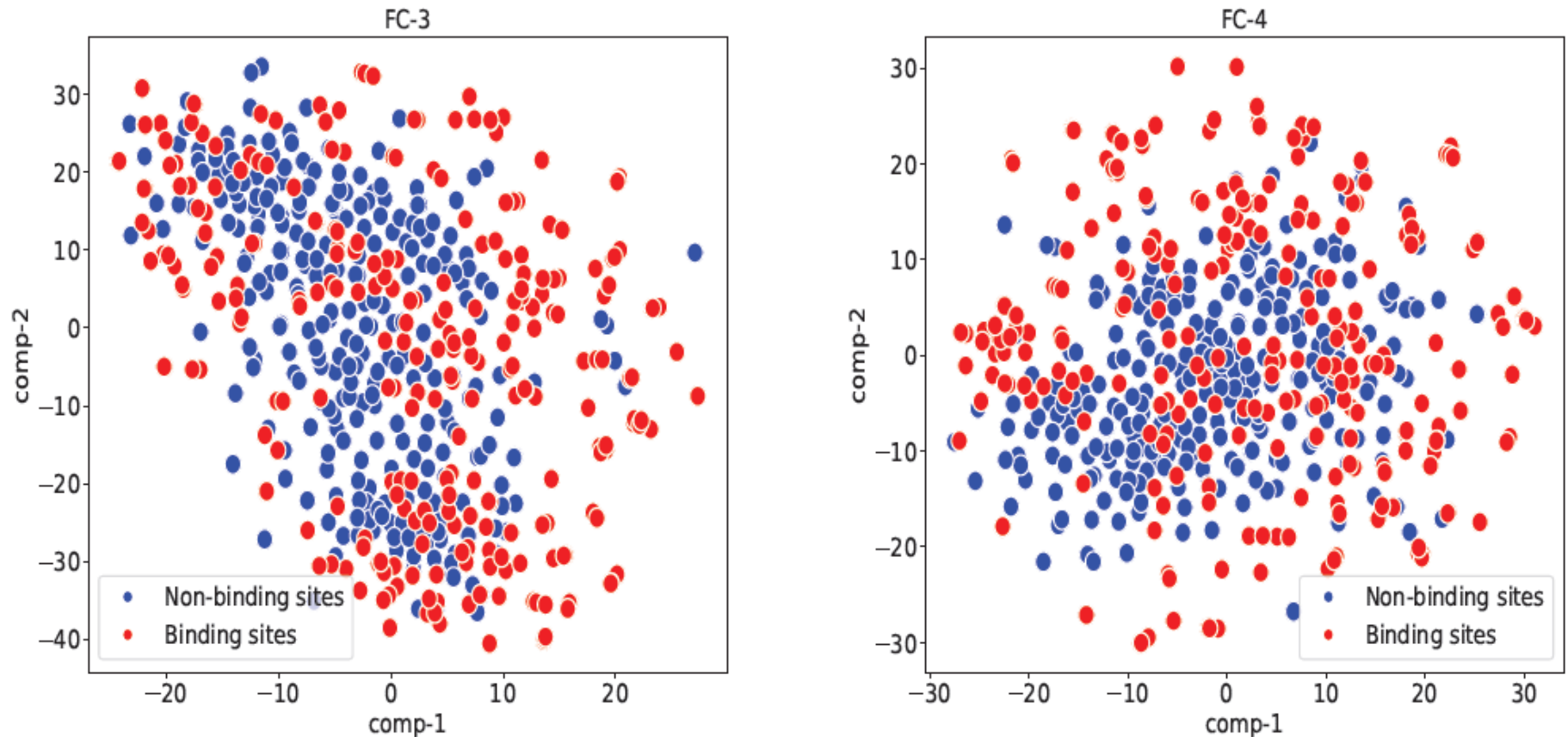


# Stacking improves class separation



***This is a t-SNE plot from protein carbohydrate binding site prediction problem. FC-1 to FC-5 are best 5 feature spaces. FC-6 is FC-1 augmented with prediction values from the base layer (i.e., what the meta layer of stacking ensemble will see). (Nawar et al.)***

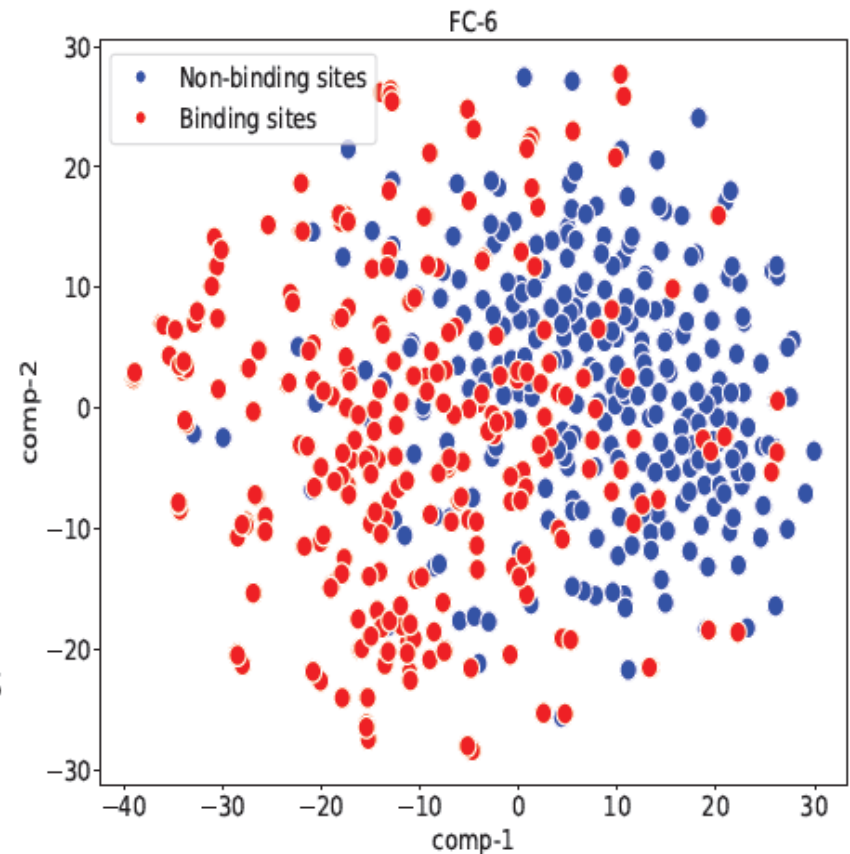
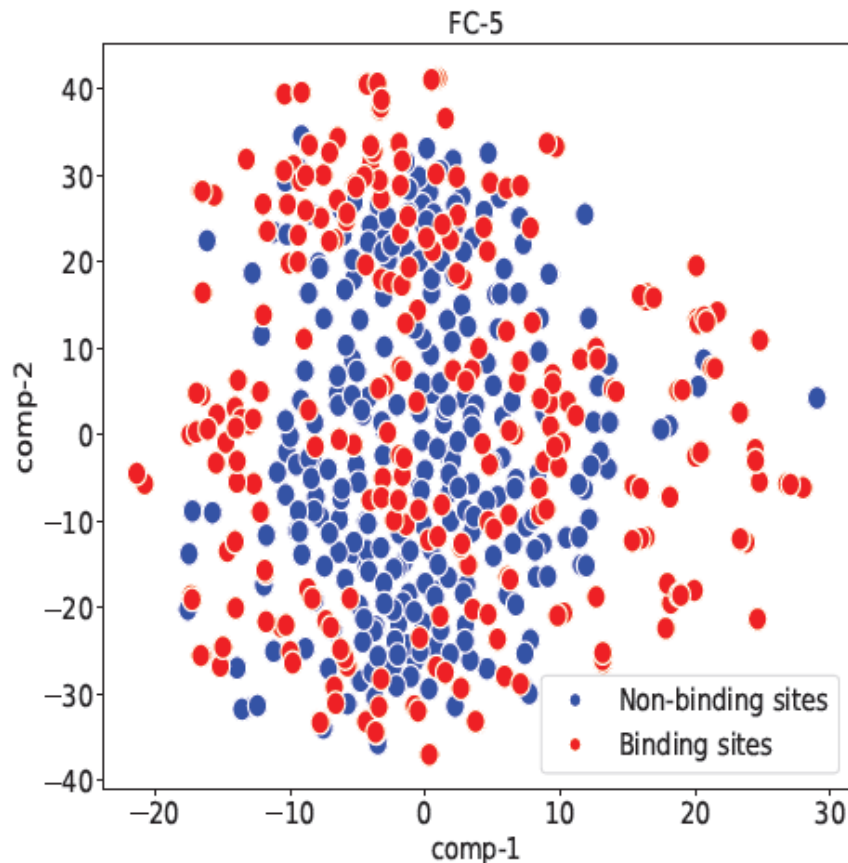
# Stacking improves class separation



***This is a t-SNE plot from protein carbohydrate binding site prediction problem. FC-1 to FC-5 are best 5 feature spaces. FC-6 is FC-1 augmented with prediction values from the base layer (i.e., what the meta layer of stacking ensemble will see). (Nawar et al.)***



# Stacking improves class separation



13

***This is a t-SNE plot from protein carbohydrate binding site prediction problem. FC-1 to FC-5 are best 5 feature spaces. FC-6 is FC-1 augmented with prediction values from the base layer (i.e., what the meta layer of stacking ensemble will see . (Nawar et al.)***

# Bagging

- K distinct training sets by subsampling
  - ▣ With replacement
- Reduces variance
- Most commonly used with decision trees
  - ▣ DTs are very sensitive to changes in examples
- Can be applied in parallel

$$h(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^K h_i(\mathbf{x})$$



# Random forests

- A form of decision tree bagging
- Takes extra measures to diversify the base DTs
- At each split point, attributes are sampled.
  - ▣ Attribute with highest IG from the sample is chosen
- Extremely Randomized Tree (ExtraTree)
  - ▣ For an attribute, we randomly sample several candidate values from a uniform distribution over the attribute's range
    - Choose the value that results in highest IG

# Random forests

- Can be created efficiently
  - ▣ Sampled attribute space
  - ▣ No need for DT pruning
  - ▣ The trees can be built in parallel
- Out-of-bag error
  - ▣ Mean error on each example, using only the trees whose example set didn't include that particular example
  - ▣ Can be used instead of CV error

# Multivariable LR Regularization

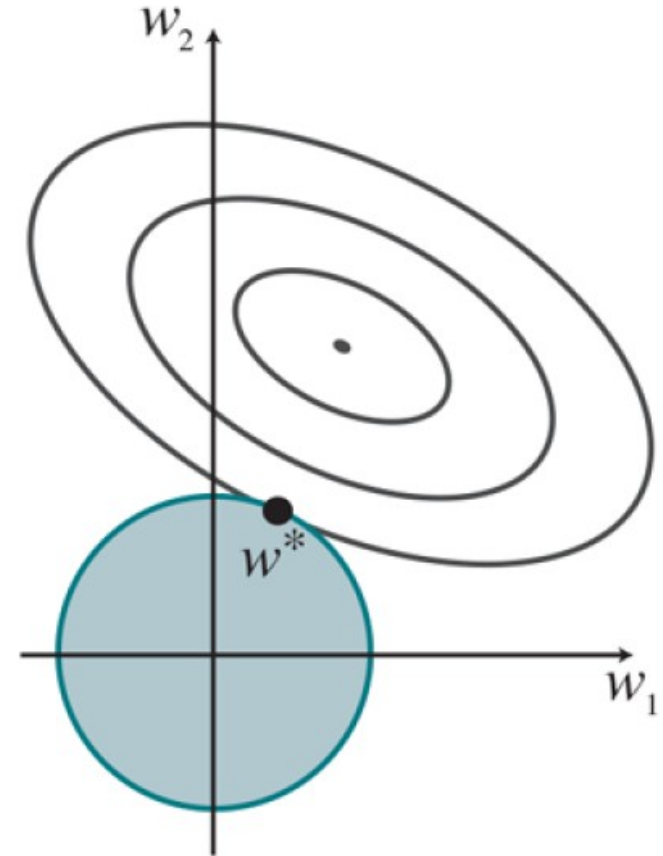
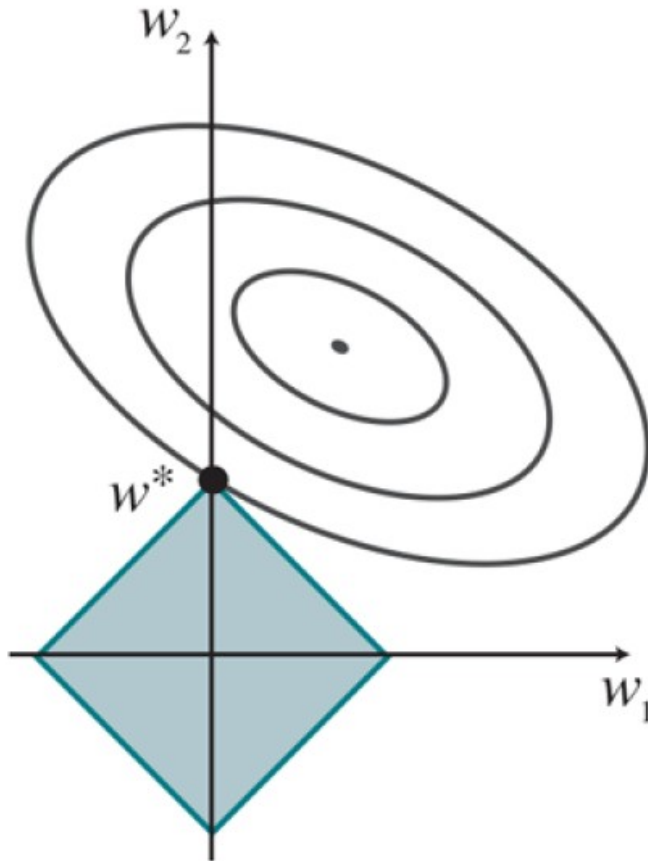
$$\text{Cost}(h) = \text{EmpLoss}(h) + \lambda \text{Complexity}(h)$$

$$\text{Complexity}(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

- *$L_1$  regularization produces sparse model*
  - *Easier for a human to understand*
  - *Less likely to overfit*
- *You can use  $L_1$  loss with  $L_2$  regularization and vice versa.*

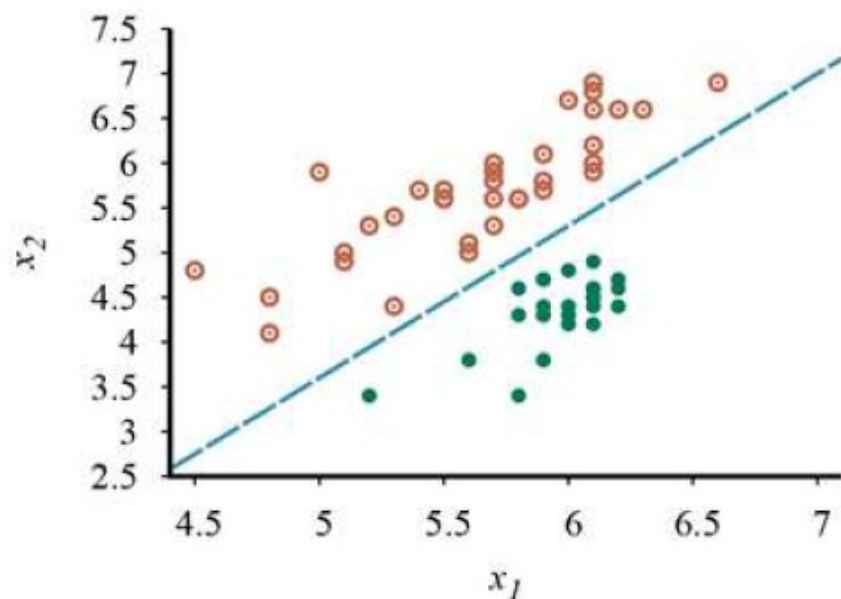
- **$L_1$  regularization – LASSO**
- **$L_2$  regularization – Ridge**

# $L_1$ vs. $L_2$ regularization

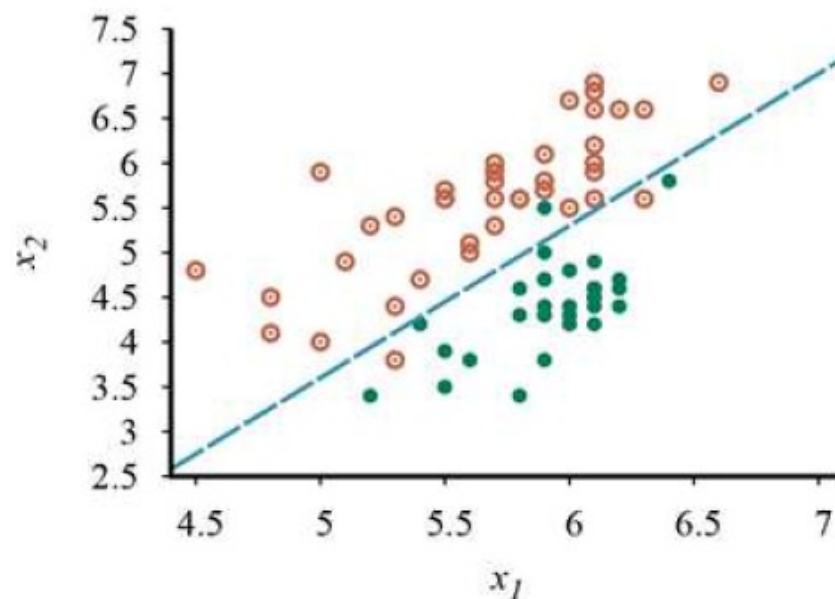


***$L_1$  regularization tends to produce a sparse model.***

# Linear Classification



(a)



(b)

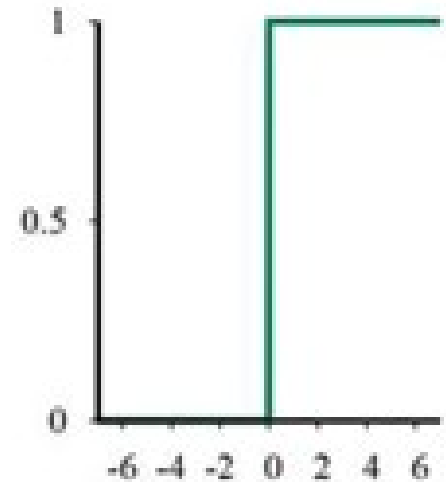
(a) Plot of two seismic data parameters, body wave magnitude  $x_1$  and surface wave magnitude  $x_2$ , for earthquakes (open orange circles) and nuclear explosions (green circles) occurring between 1982 and 1990 in Asia and the Middle East (Kebeasy *et al.*, 1998). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

# Linear Classification

$h_{\mathbf{w}}(\mathbf{x}) = 1$  if  $\mathbf{w} \cdot \mathbf{x} \geq 0$  and 0 otherwise

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x})$$

where  $\text{Threshold}(z) = 1$  if  $z \geq 0$  and 0 otherwise



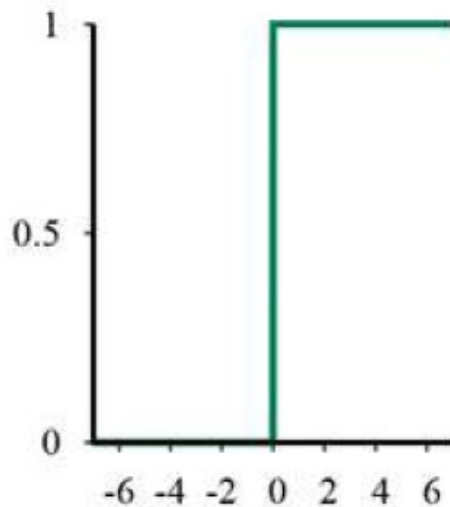
- **Gradient is discontinuous**
- **Gradient is zero almost everywhere in weight space except at those points where and at those points the gradient is undefined**

# Perceptron Learning Rule

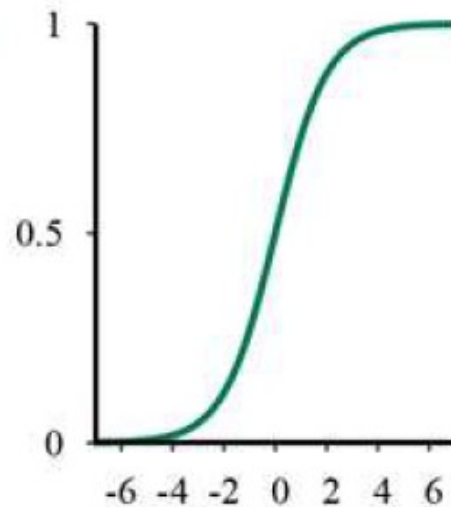
$$w_i \leftarrow w_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

- If the output is correct (i.e., ) then the weights are not changed.
- If  $y$  is 1 but  $h_{\mathbf{w}}(\mathbf{x})$  is 0, then  $w_i$  is *increased* when the corresponding input is positive and *decreased* when is negative. This makes sense, because we want to make  $w \cdot x$  bigger so that outputs a 1.
- If  $y$  is 0 but  $h_{\mathbf{w}}(\mathbf{x})$  is 1, then  $w_i$  is *decreased* when the corresponding input is positive and *increased* when is negative. This makes sense, because we want to make  $w \cdot x$  smaller so that outputs a 0.

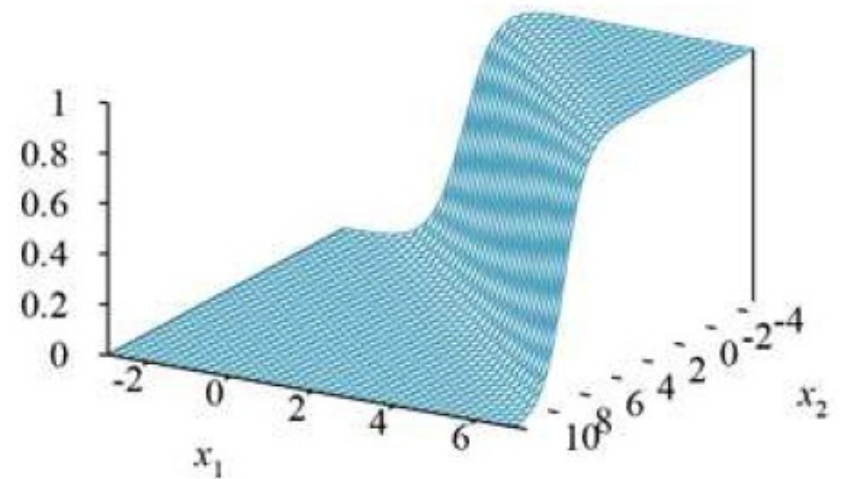
# Logistic Regression



(a)



(b)



(c)

(a) The hard threshold function  $Threshold(z)$  with 0/1 output. Note that the function is nondifferentiable at  $z = 0$ . (b) The logistic function,  $Logistic(z) = \frac{1}{1+e^{-z}}$ , also known as the sigmoid function. (c) Plot of a logistic regression hypothesis  $h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})$  for the data shown in [Figure 19.15\(b\)](#).



# Logistic Regression

- Classification problem
  - ▣ 0 → Negative class
  - ▣ 1 → Positive class
- Modeling it as a linear regression will work poorly
  - ▣  $h(x)$  can go beyond  $\{0,1\}$
- $h_w(x) = g(w^T x) = \frac{1}{1 + e^{-w^T x}}$
- Derivative of  $g(z) = g(z)(1 - g(z))$

# Logistic Regression

$$P(y = 1 \mid x; \theta) = h_{\theta}(x)$$

$$P(y = 0 \mid x; \theta) = 1 - h_{\theta}(x)$$

$$p(y \mid x; \theta) = (h_{\theta}(x))^y (1 - h_{\theta}(x))^{1-y}$$

$$L(\theta) = p(\vec{y} \mid X; \theta)$$

$$= \prod_{i=1}^n p(y^{(i)} \mid x^{(i)}; \theta)$$

$$= \prod_{i=1}^n (h_{\theta}(x^{(i)}))^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{1-y^{(i)}}$$



*Likelihood of  
the parameters*

# Maximizing the log Likelihood

$$\begin{aligned}\ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^n y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))\end{aligned}$$

***For single data point***

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left( y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= (y(1 - g(\theta^T x)) - (1 - y)g(\theta^T x)) x_j \\ &= (y - h_{\theta}(x)) x_j\end{aligned}$$

# Update rule

- $\theta_i = \theta_i + \alpha(y - h_{\theta}(x))x_i$
- Identical to the update rule in linear regression