

# Computer Vision

## Introduction to Classifiers

**Dr. Mrinmoy Ghorai**

**Indian Institute of Information Technology  
Sri City, Chittoor**



# Previous Class

---

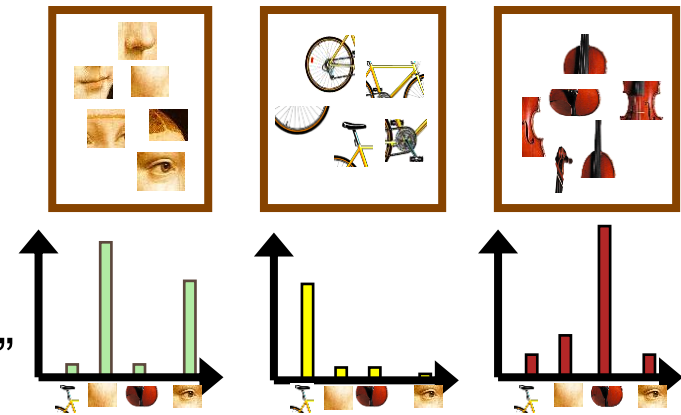
## Image features and categorization

Choosing right features  
Object, Scene, Action, etc.



## Bag-of-visual-words

Extract local features  
Learn “visual vocabulary”  
Quantize features using visual vocabulary  
Represent by frequencies of “visual words”



# Today's Class

---

Training  
Images



Training

Training  
Labels

Image  
Features

Classifier  
Training

Trained  
Classifier

Testing

Image  
Features

Trained  
Classifier

Prediction  
**Outdoor**



Test Image

# Today's Class

---

**Nearest Neighbor Classifier**

**K - Nearest Neighbor Classifier**

**Linear Classifier**

**Support Vector Machine**

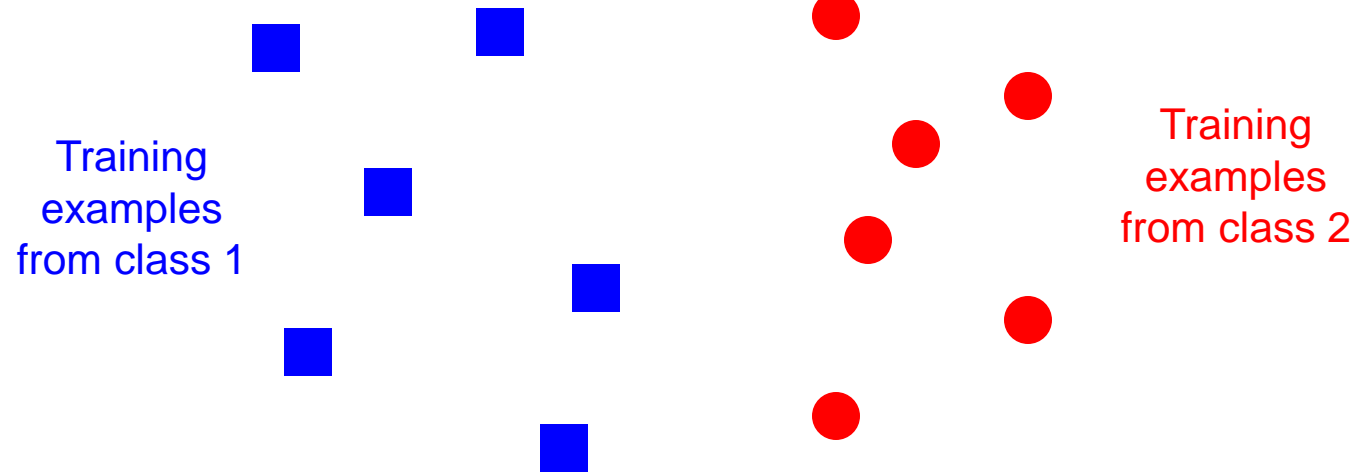
**Non-linear SVM**

**Multi-class SVM**

**Softmax Classifier**

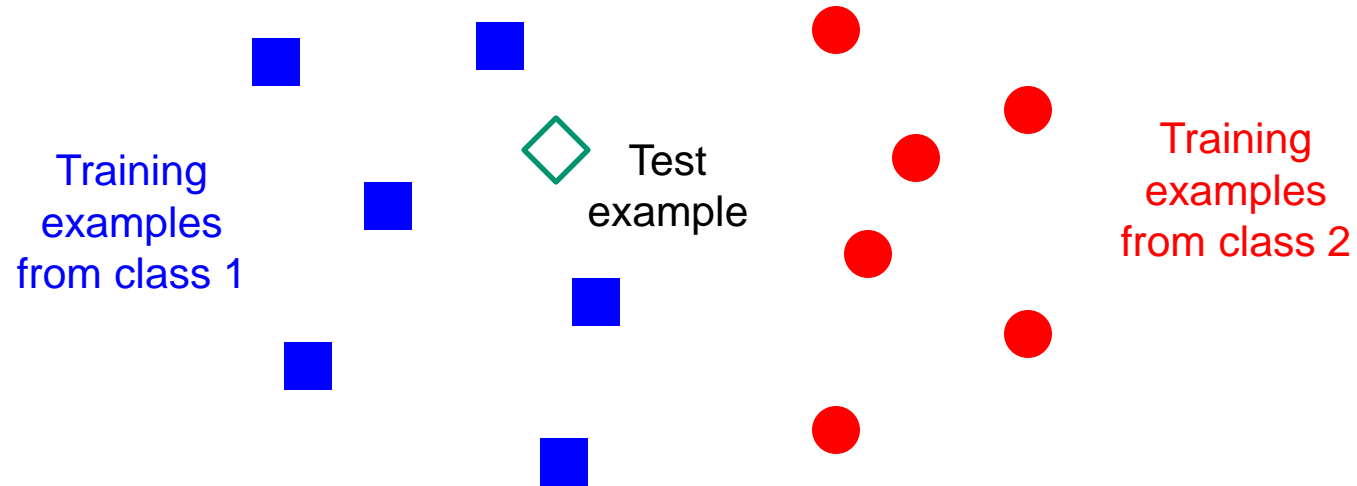
# Classifiers: Nearest neighbor

---



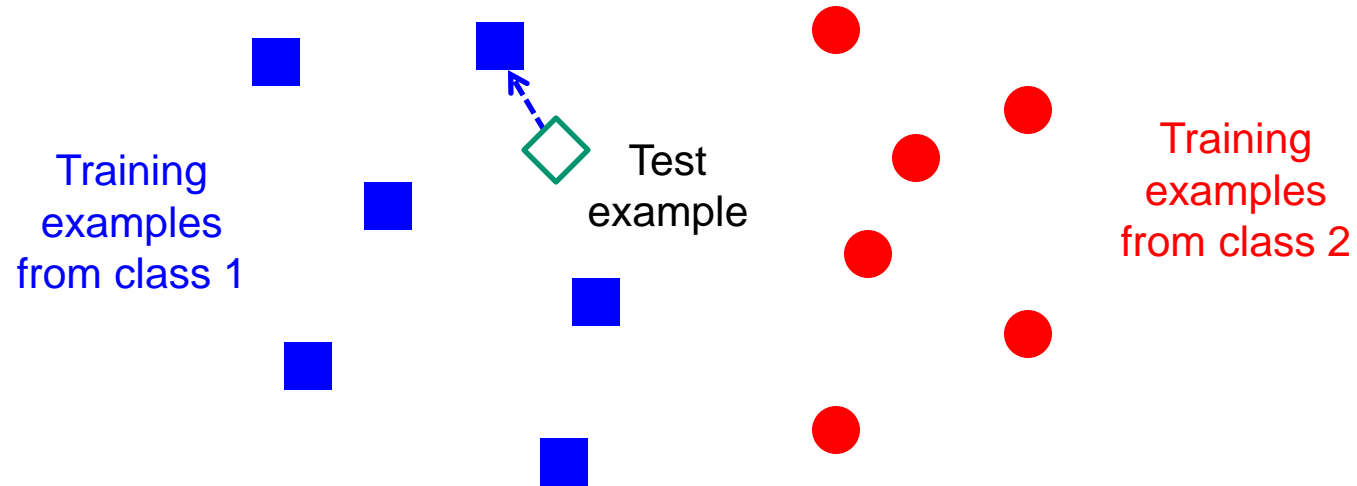
# Classifiers: Nearest neighbor

---



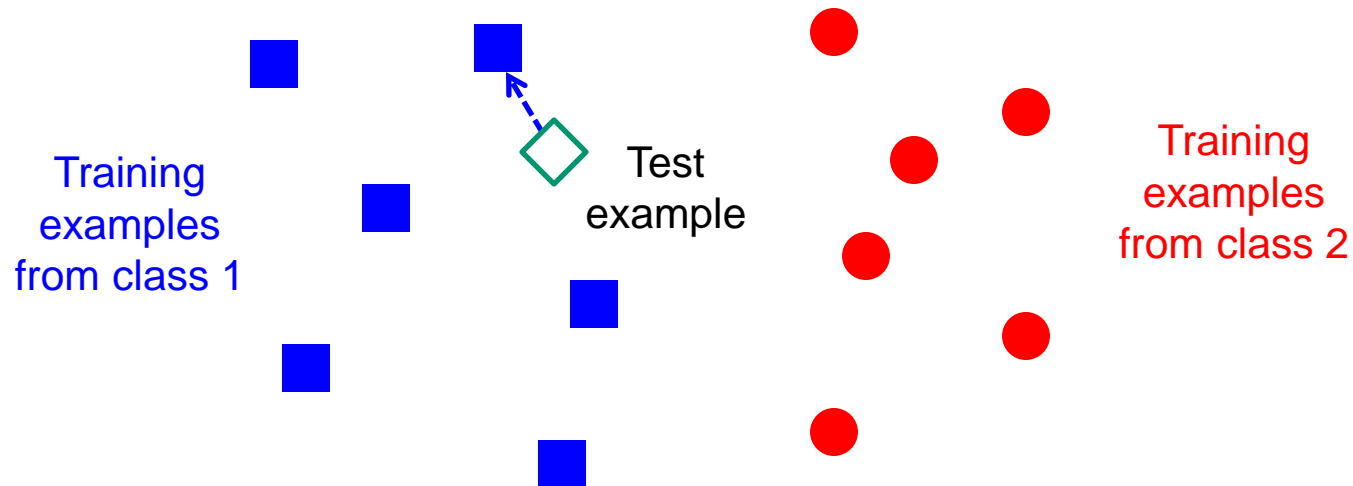
# Classifiers: Nearest neighbor

---



# Classifiers: Nearest neighbor

---



$f(\mathbf{x}) = \text{label of the training example nearest to } \mathbf{x}$

All we need is a distance function for our inputs

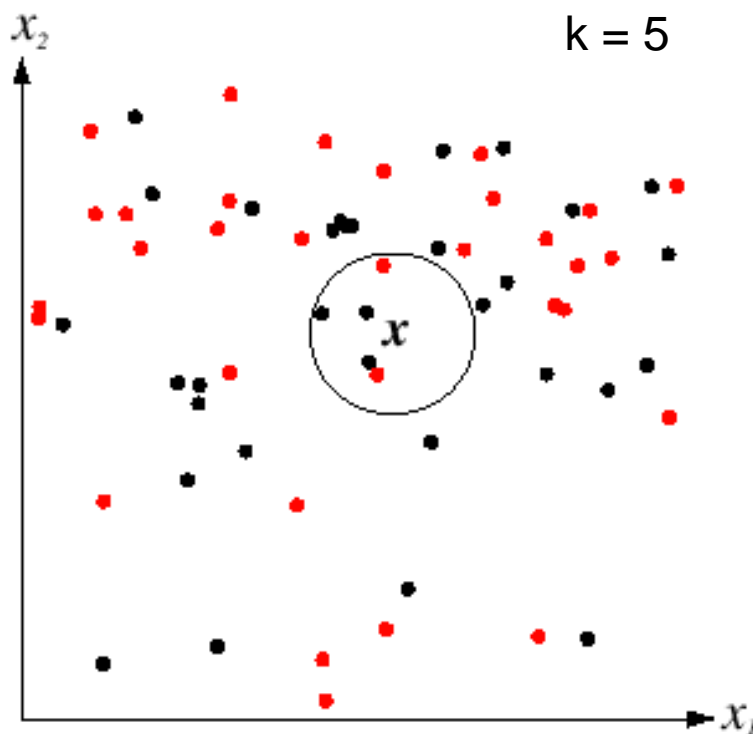
No training required!



# K-nearest neighbor classifier

---

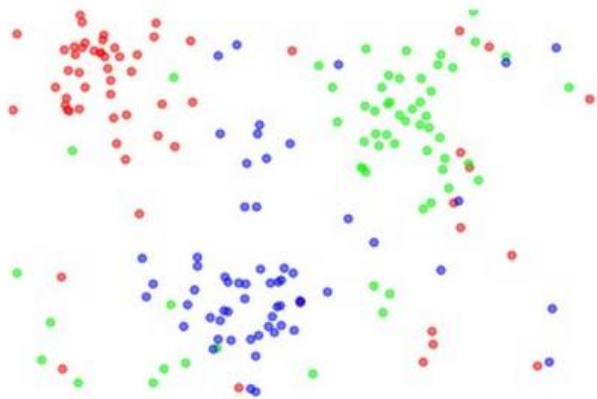
- For a new point, find the  $k$  closest points from training data
- Vote for class label with labels of the  $k$  points



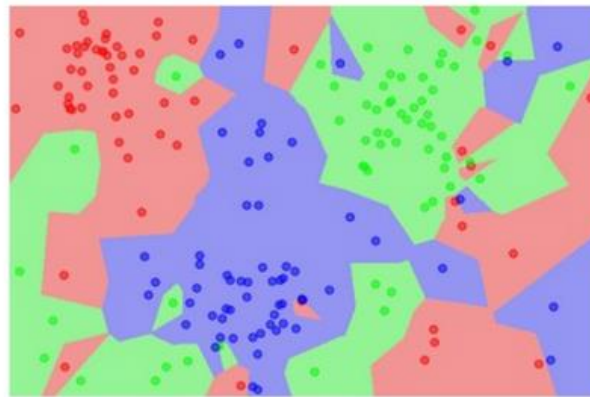
# K-nearest neighbor classifier

---

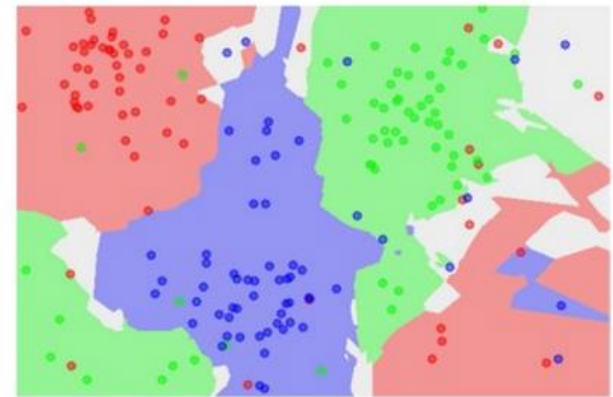
the data



NN classifier



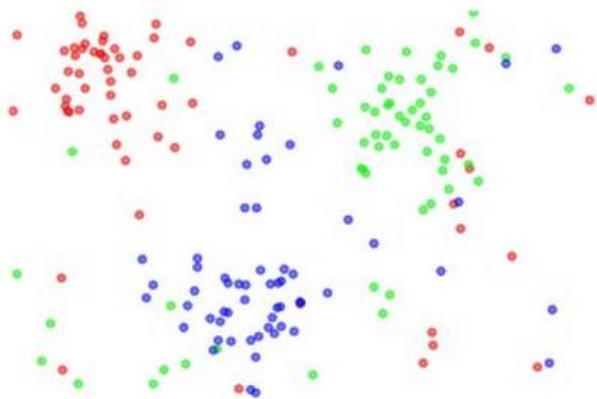
5-NN classifier



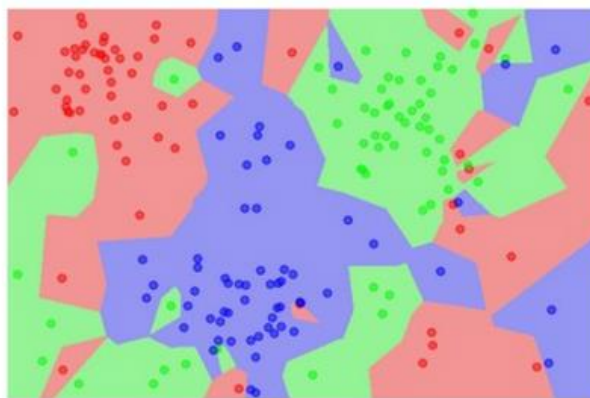
# K-nearest neighbor classifier

---

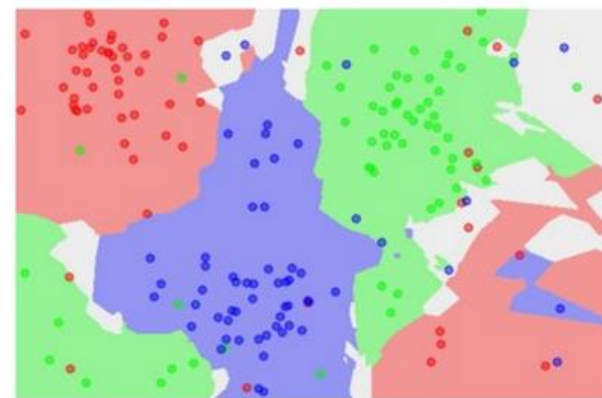
the data



NN classifier



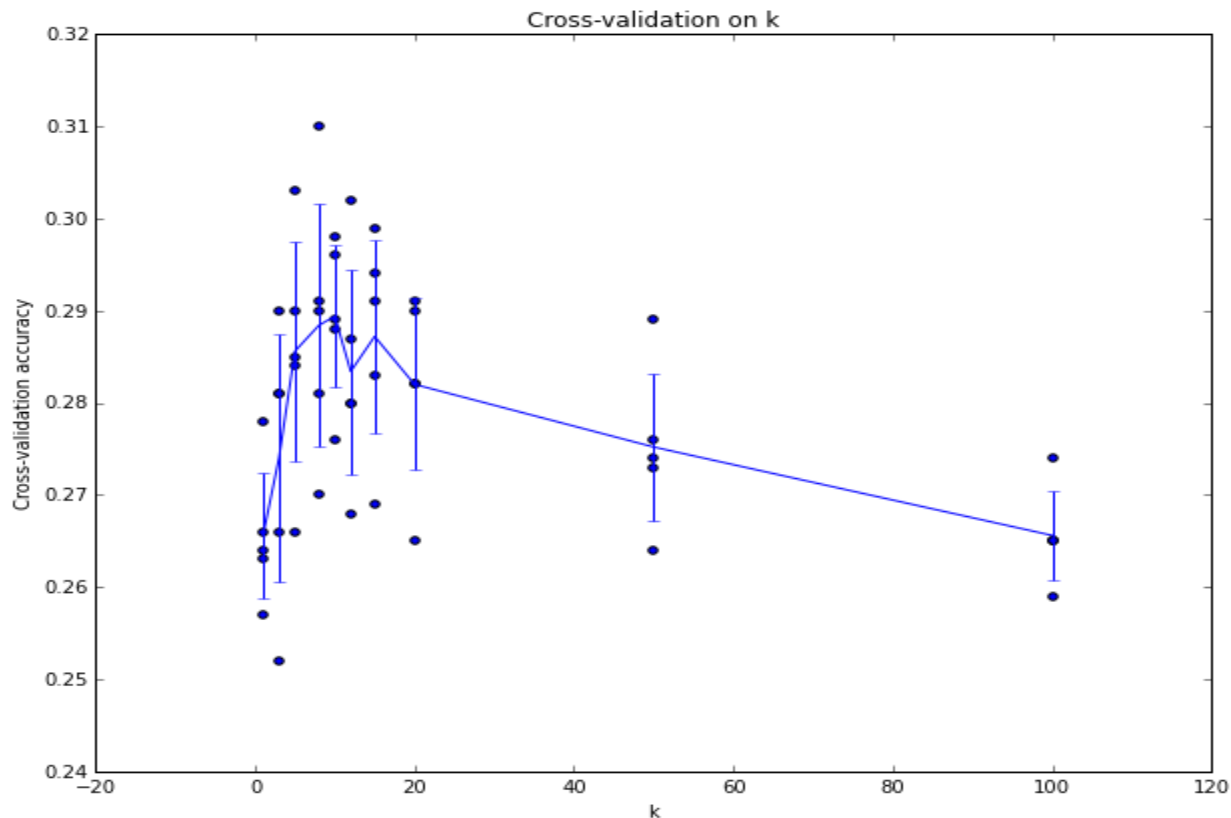
5-NN classifier



Which classifier is more robust to *outliers*?

# Choice of K in KNN classifier

---



Example of a 5-fold cross-validation run for the parameter  $k$ . Note that in this particular case, the cross-validation suggests that a value of about  $k = 7$  works best on this particular dataset (corresponding to the peak in the plot).

# Classifiers: K-Nearest neighbor

---

“Non-parametric” classifier: the entire training set is essentially the model parameters.

# Classifiers: K-Nearest neighbor

---

“Non-parametric” classifier: the entire training set is essentially the model parameters.

## Pros:

- **Very fast at training** time
- **Flexible**: all it requires is a way to compute similarity or distances between pairs of features. Applies to many different kinds of features.
- Works with **any number of classes**.
- Works well in practice for large datasets (but see cons)

# Classifiers: K-Nearest neighbor

---

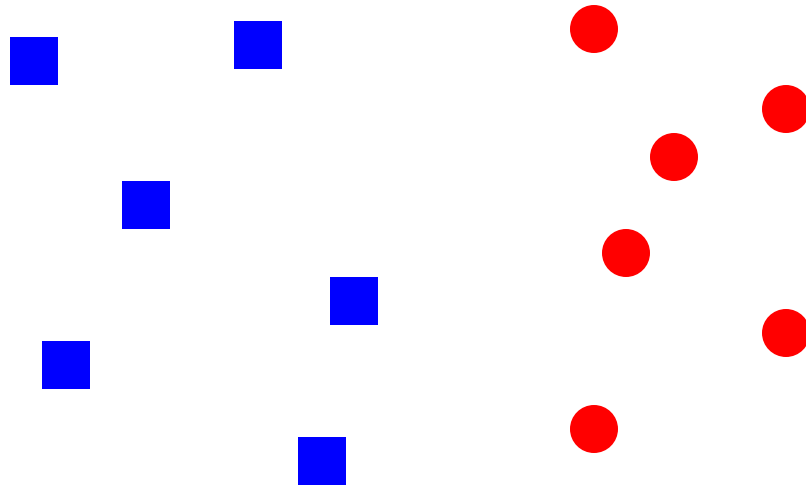
“Non-parametric” classifier: the entire training set is essentially the model parameters.

## Cons:

- The classifier must ***remember*** all of the training data and store it for future comparisons with the test data.
- This is **space inefficient** because datasets may easily be gigabytes in size.
- **Slow at test time** (need to compute distances between test example and every training example)
- Optimum value of **K is not known**.

# Linear classifiers – 2 class problem

---

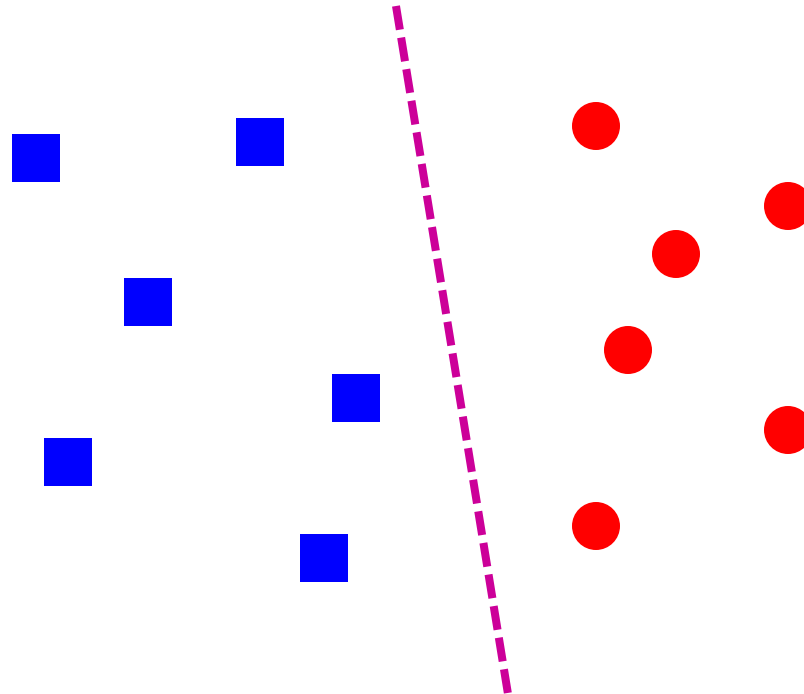


Find a *linear function* to separate the classes:



# Linear classifiers – 2 class problem

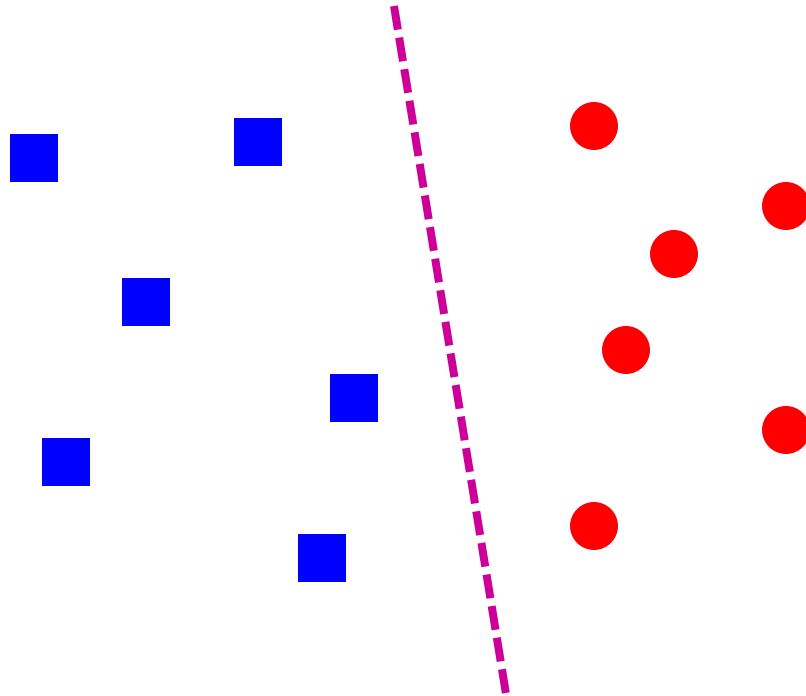
---



Find a *linear function* to separate the classes:

# Linear classifiers – 2 class problem

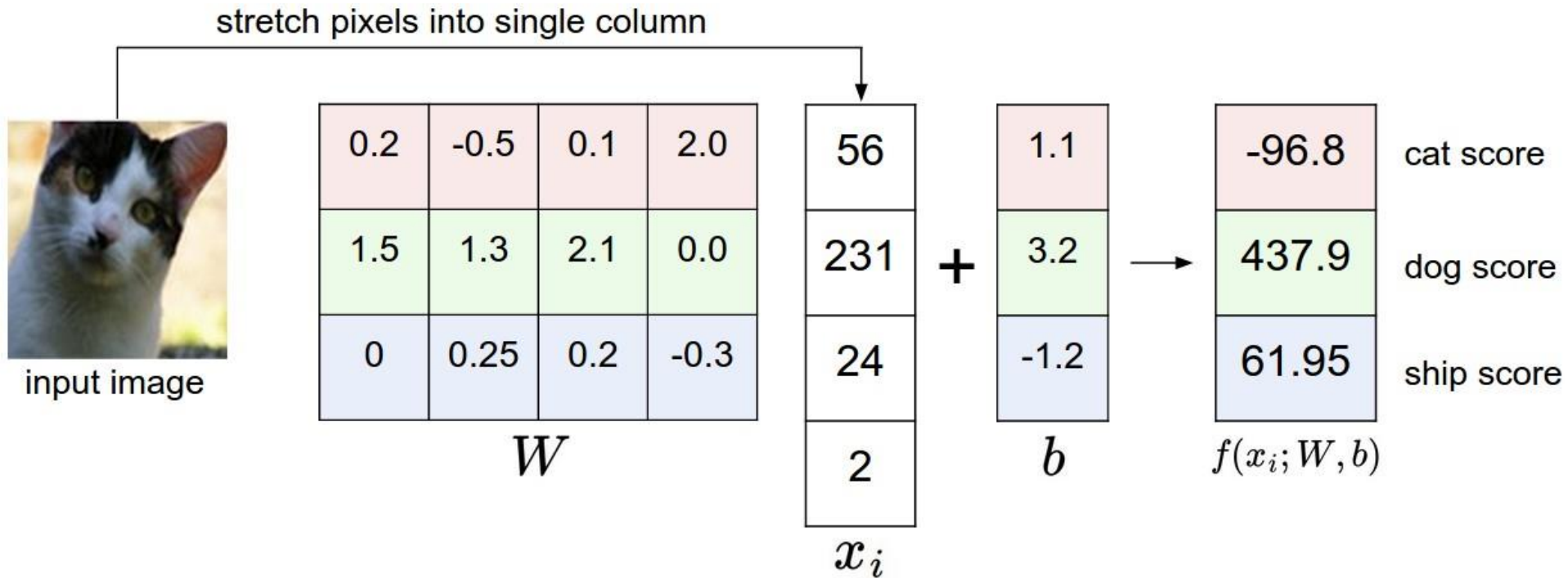
---



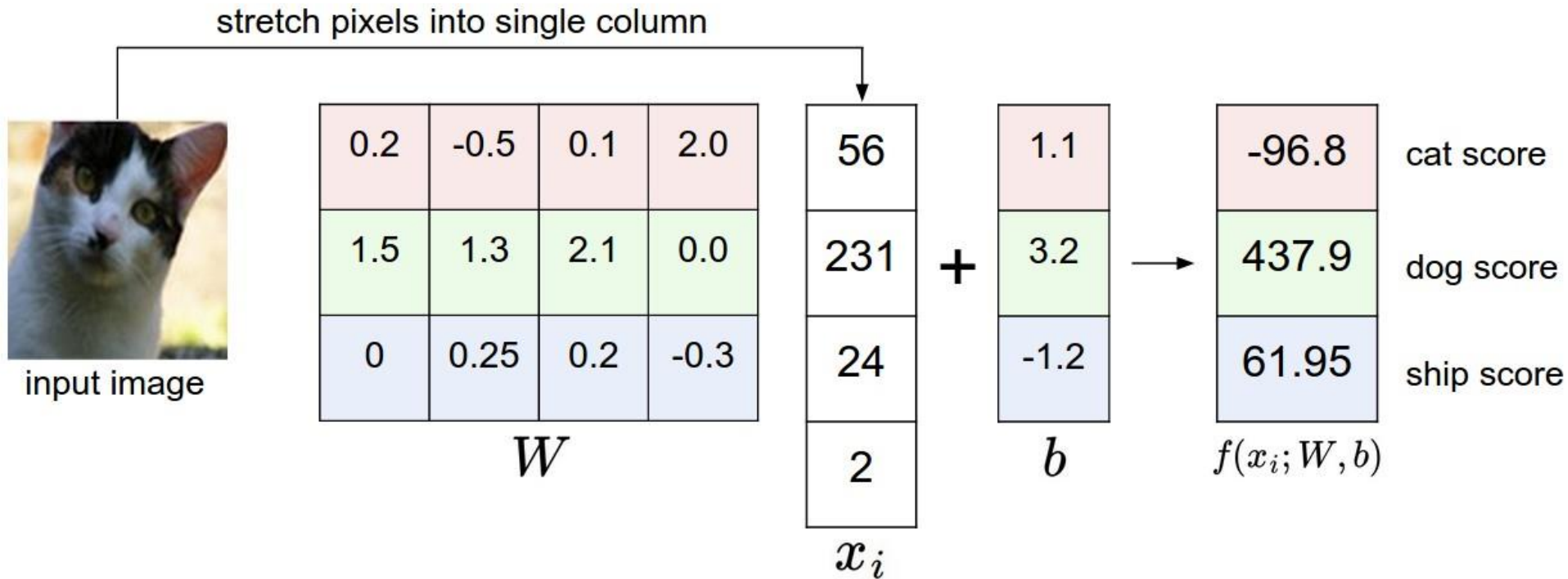
Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

# Linear classifiers – more than 2 class

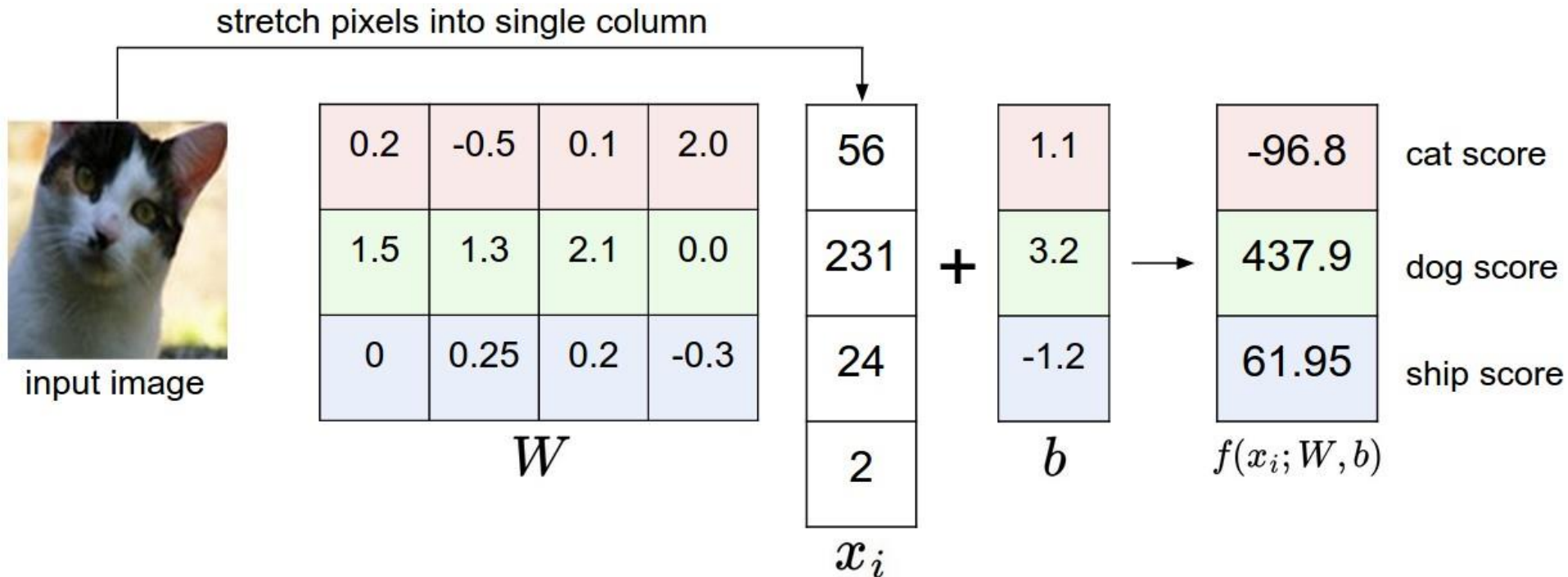


# Linear classifiers – more than 2 class



$$f(x_i, W, b) = Wx_i + b$$

# Linear classifiers – more than 2 class



$$f(x_i, W, b) = Wx_i + b$$

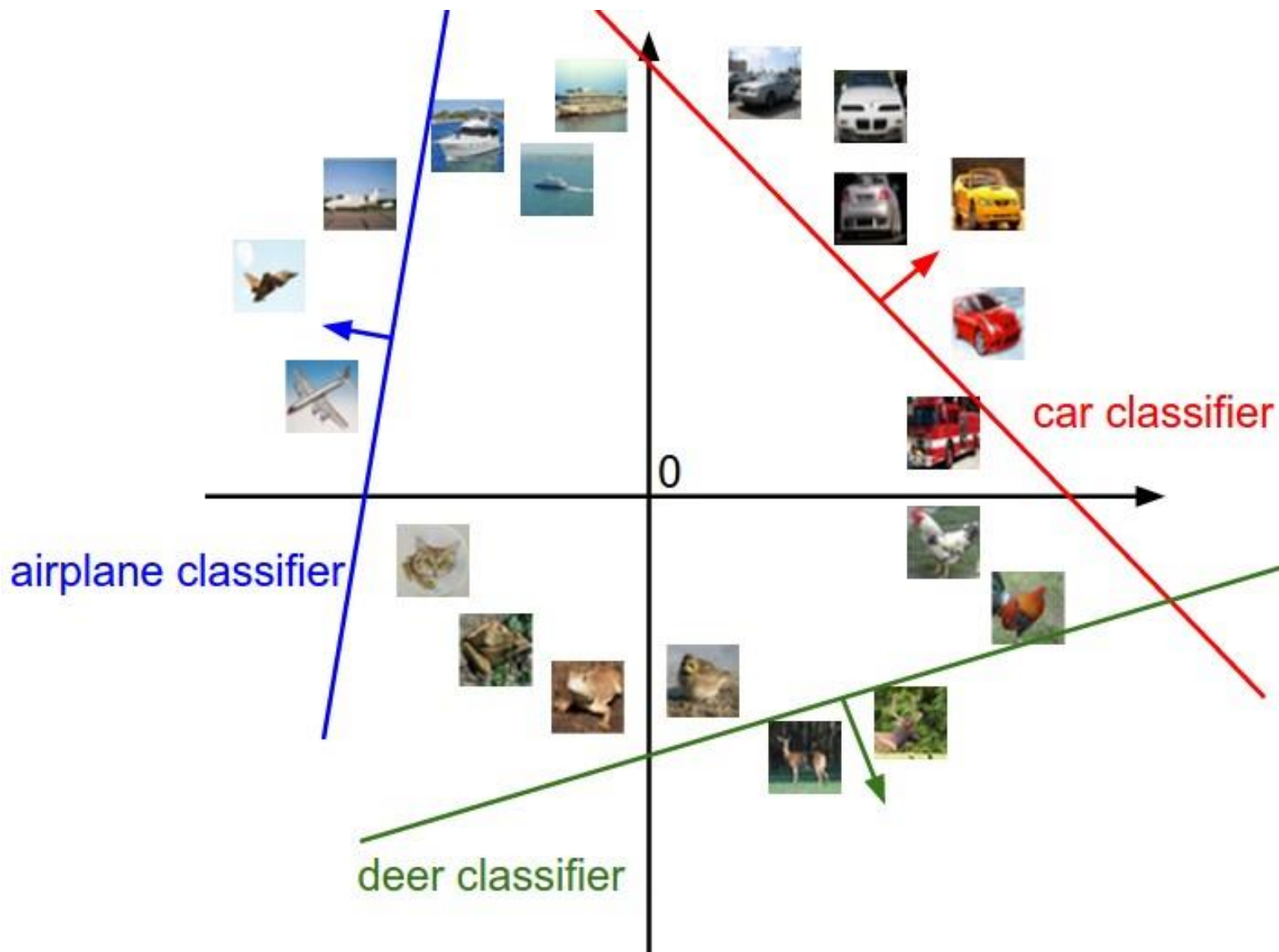
Image  $x_i$  has all of its pixels flattened out to a single column vector of shape  $[D \times 1]$ .

Matrix  $W$  (of size  $[K \times D]$ ), and vector  $b$  (of size  $[K \times 1]$ ) are the **parameters**.

$K$  is the number of classes.

# Analogy of images as high-dimensional points

---



Source: cs231n, <http://cs231n.github.io/linear-classify/>

# Interpretation of linear classifiers as template matching

---

plane



car



bird



cat



deer



dog



frog



horse



ship



truck

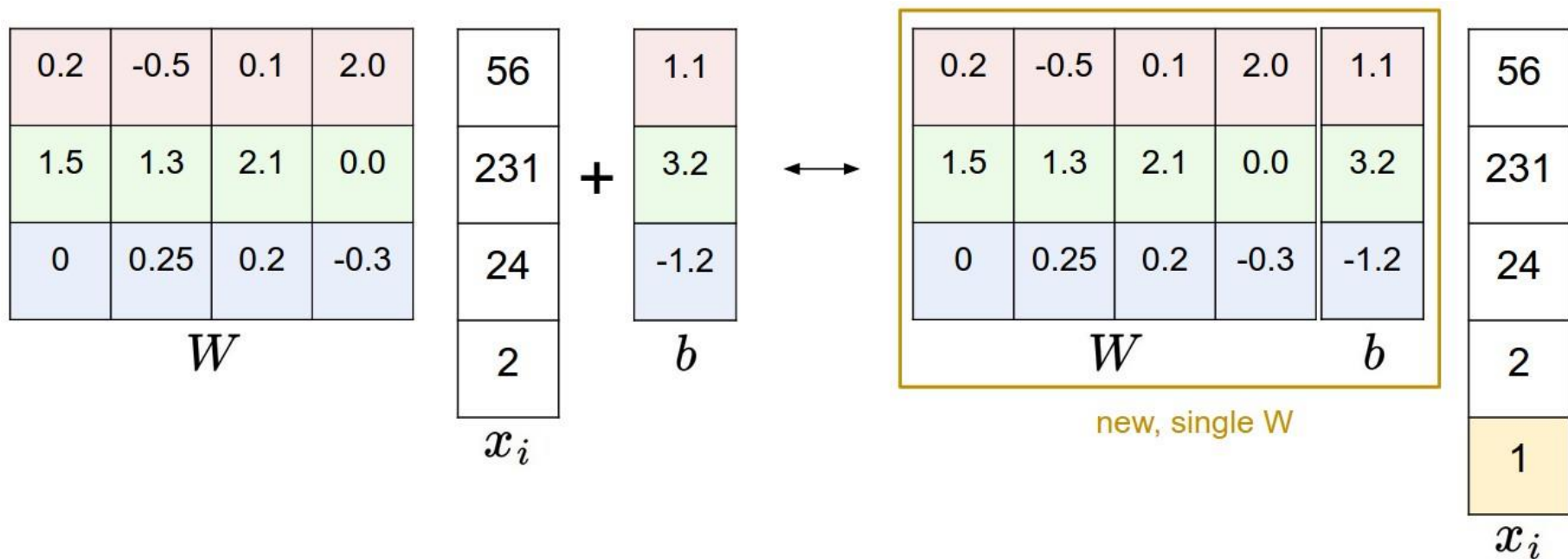


Example learned weights at the end of learning for CIFAR-10. Note that, for example, the ship template contains a lot of blue pixels as expected. This template will therefore give a high score once it is matched against images of ships on the ocean with an inner product.

Source: cs231n, <http://cs231n.github.io/linear-classify/>

# Bias Trick

---





# Linear classifiers

---

“Parametric” classifier: model defined by a small number of parameters ( $w$ ,  $b$ )

Pros:

- Very fast at test time

Cons:

- Slow at training time: need to estimate the parameters
- Data may not be linearly separable

# Nearest neighbor vs. linear classifiers

---

- NN pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method

# Nearest neighbor vs. linear classifiers

---

- NN pros:
  - Simple to implement
  - Decision boundaries not necessarily linear
  - Works for any number of classes
  - *Nonparametric* method
- NN cons:
  - Need good distance function
  - Slow at test time, Memory in-efficient

# Nearest neighbor vs. linear classifiers

---

- NN pros:

- Simple to implement
- Decision boundaries not necessarily linear
- Works for any number of classes
- *Nonparametric* method

- NN cons:

- Need good distance function
- Slow at test time, Memory in-efficient

- Linear pros:

- Low-dimensional *parametric* representation
- Very fast at test time

# Nearest neighbor vs. linear classifiers

---

- NN pros:

- Simple to implement
- Decision boundaries not necessarily linear
- Works for any number of classes
- *Nonparametric* method

- NN cons:

- Need good distance function
- Slow at test time, Memory in-efficient

- Linear pros:

- Low-dimensional *parametric* representation
- Very fast at test time

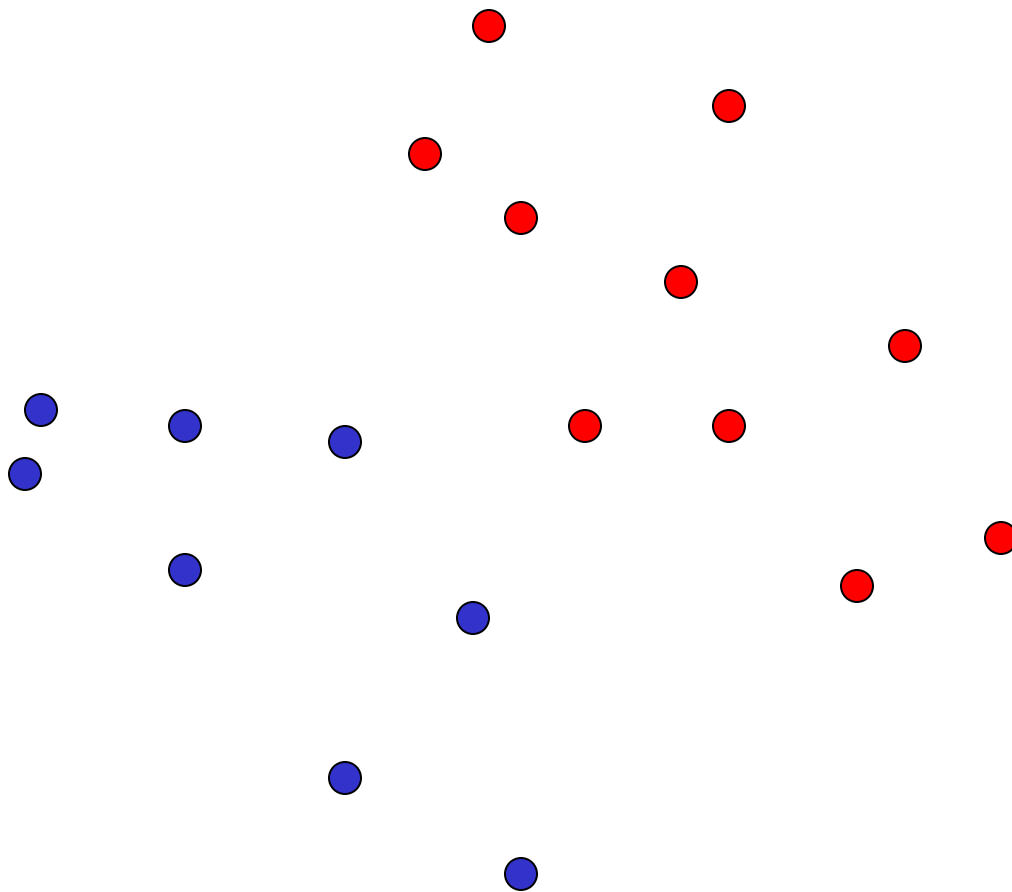
- Linear cons:

- How to train the linear function?
- What if data is not linearly separable?

# Support vector machines

---

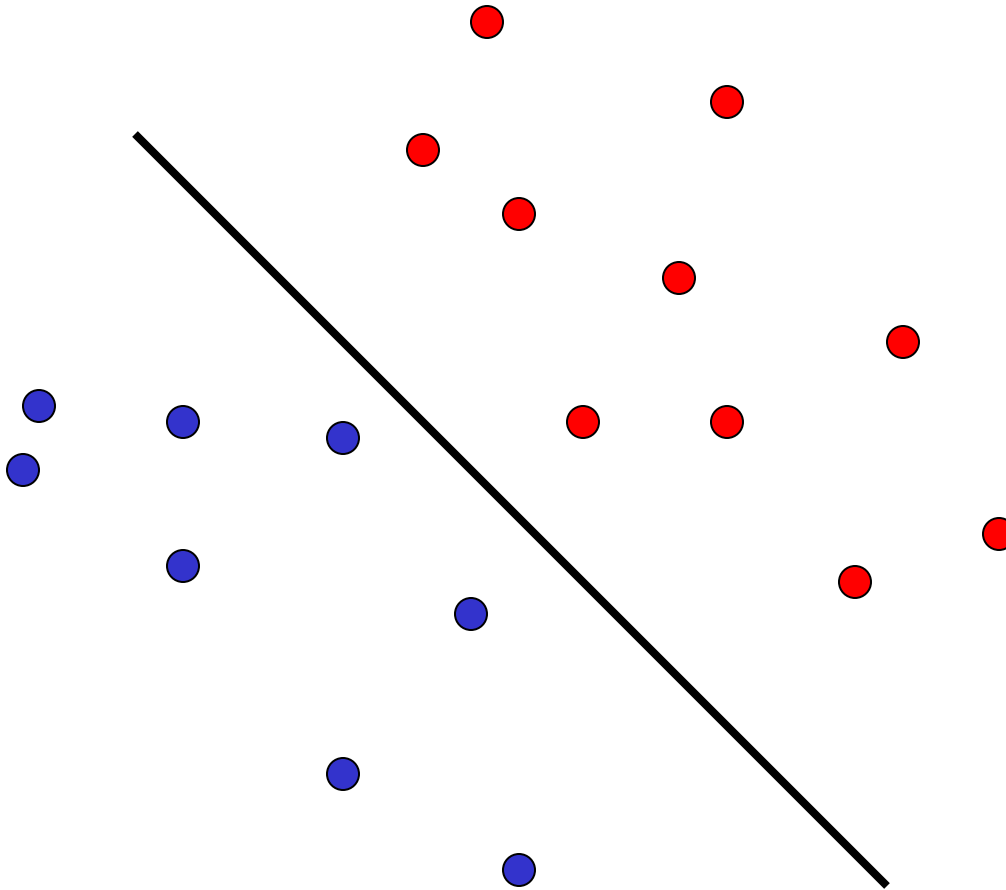
- When the data is linearly separable, there may be more than one separator (hyperplane)



# Support vector machines

---

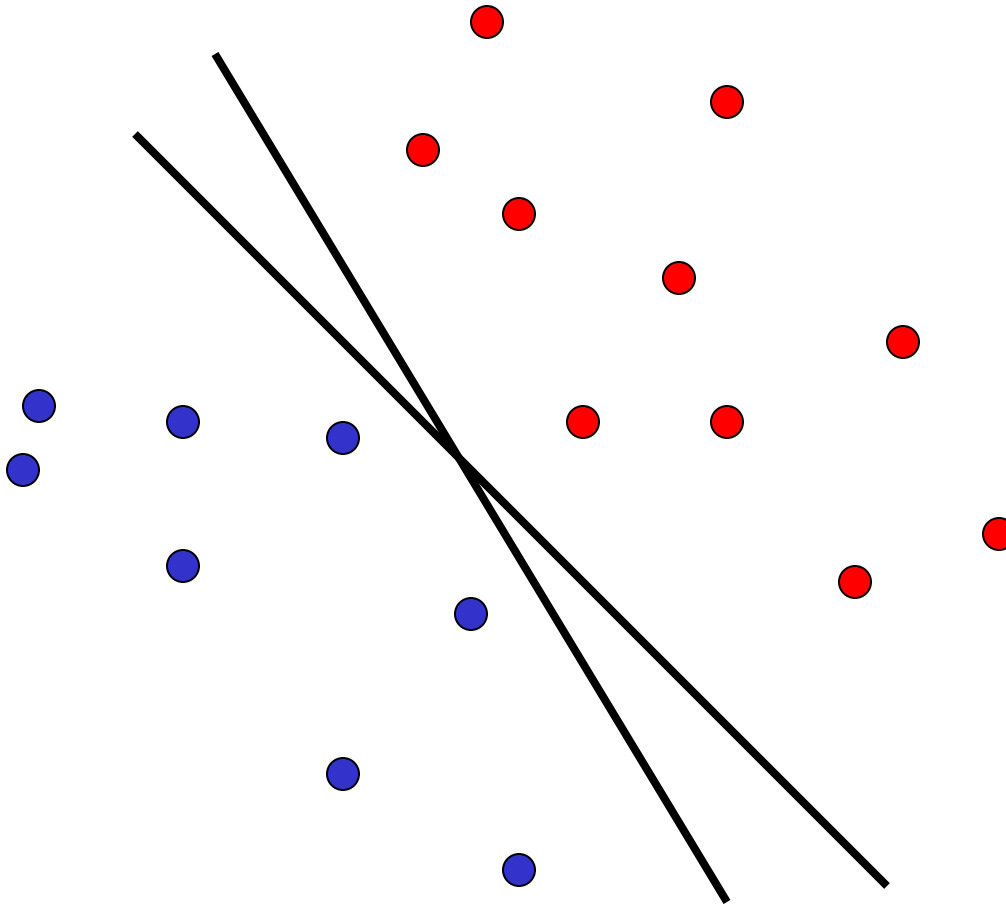
- When the data is linearly separable, there may be more than one separator (hyperplane)



# Support vector machines

---

- When the data is linearly separable, there may be more than one separator (hyperplane)

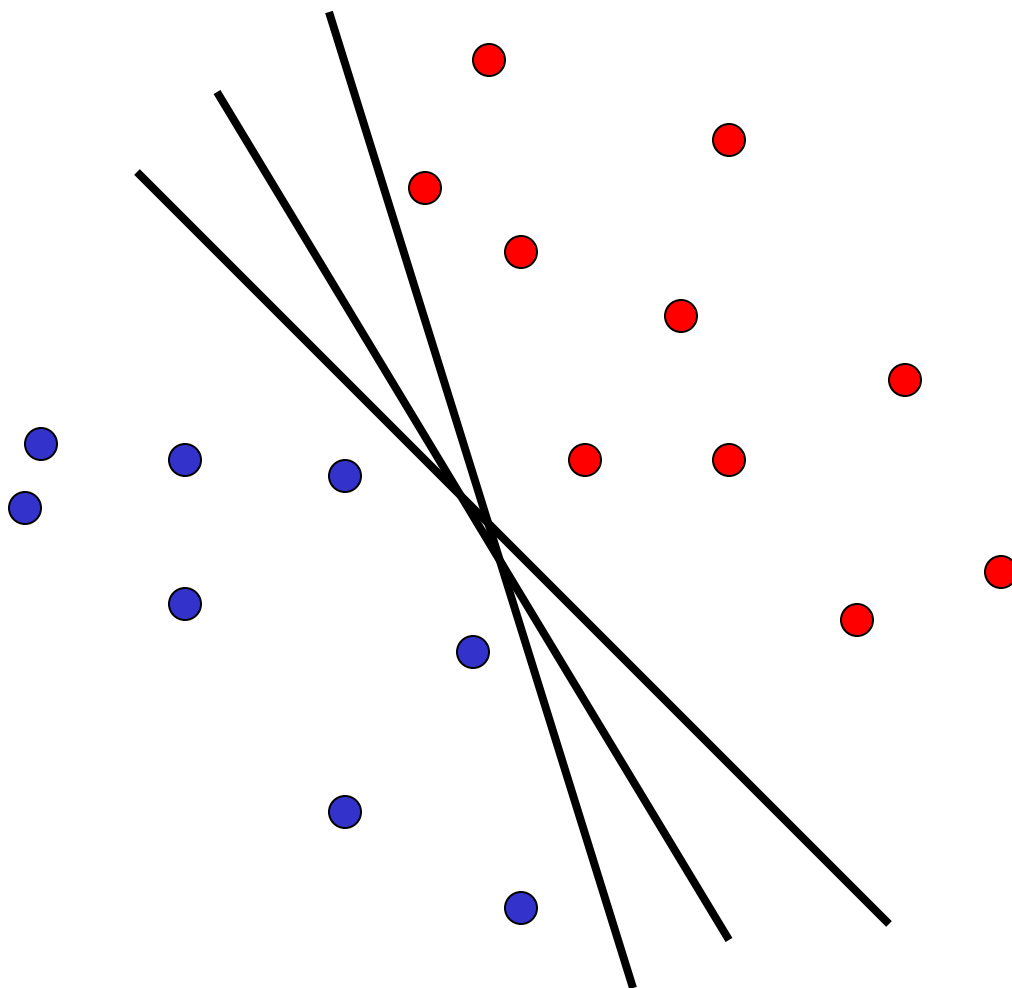




# Support vector machines

---

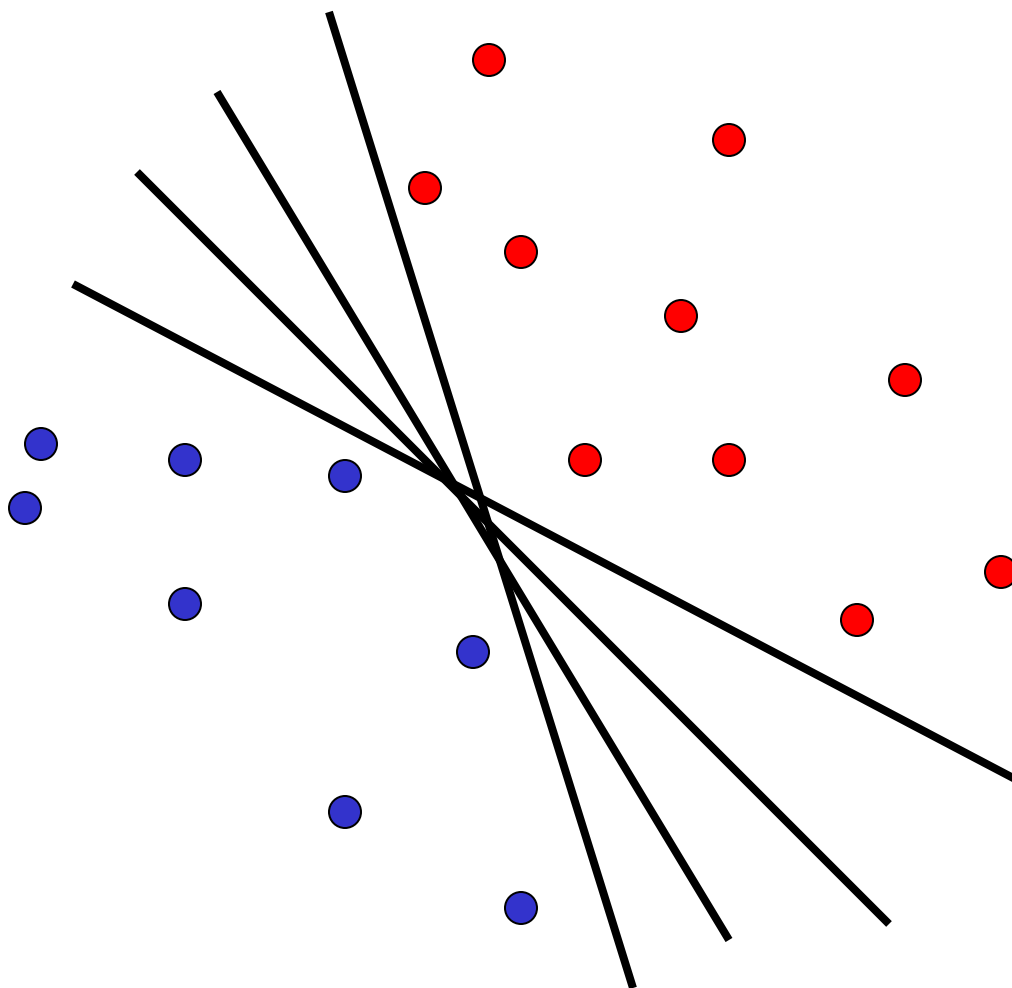
- When the data is linearly separable, there may be more than one separator (hyperplane)



# Support vector machines

---

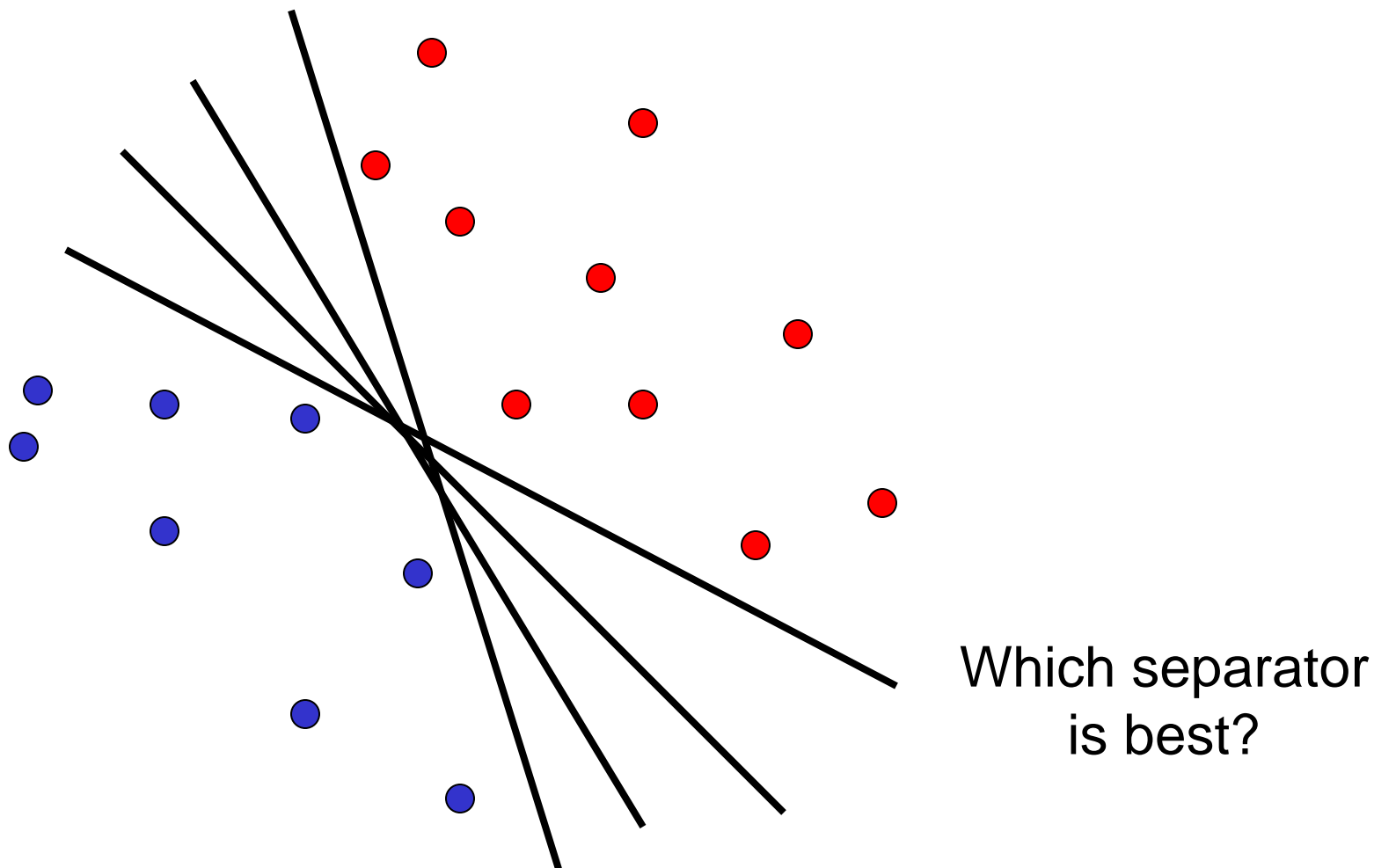
- When the data is linearly separable, there may be more than one separator (hyperplane)



# Support vector machines

---

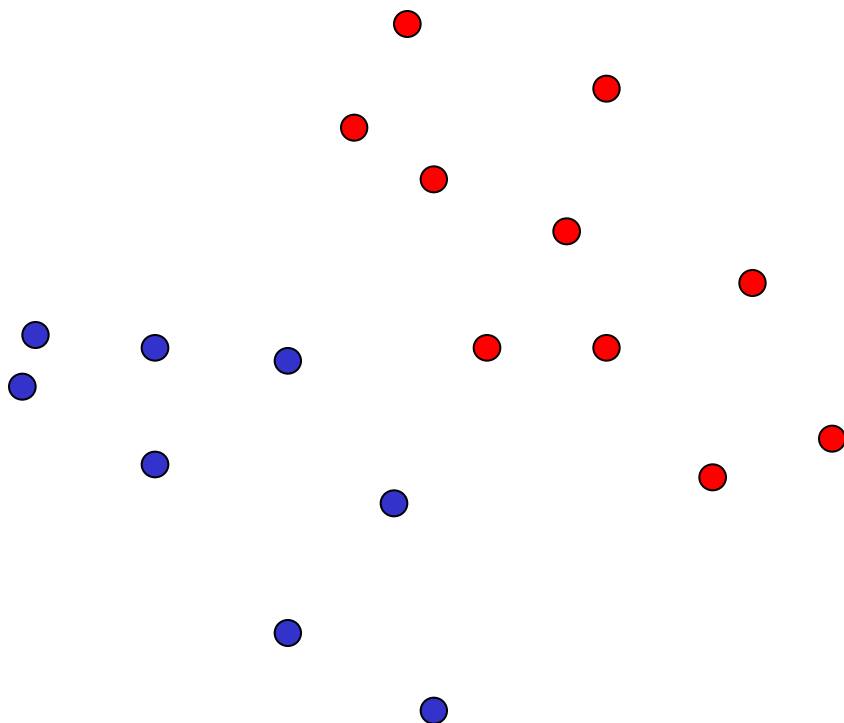
- When the data is linearly separable, there may be more than one separator (hyperplane)



# Support vector machines

---

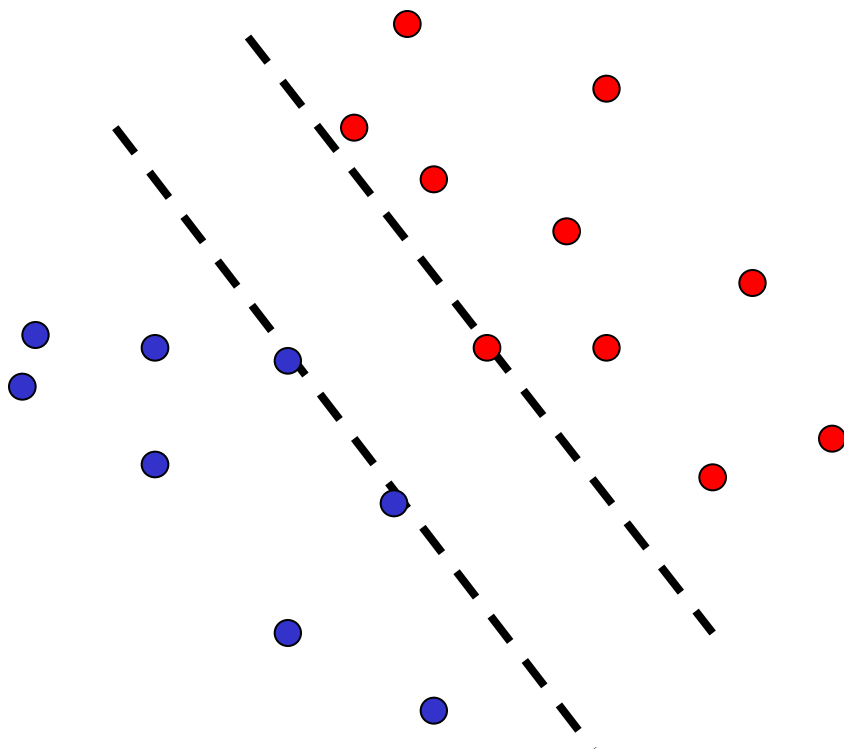
- Find hyperplane that maximizes the *margin* between the positive and negative examples



# Support vector machines

---

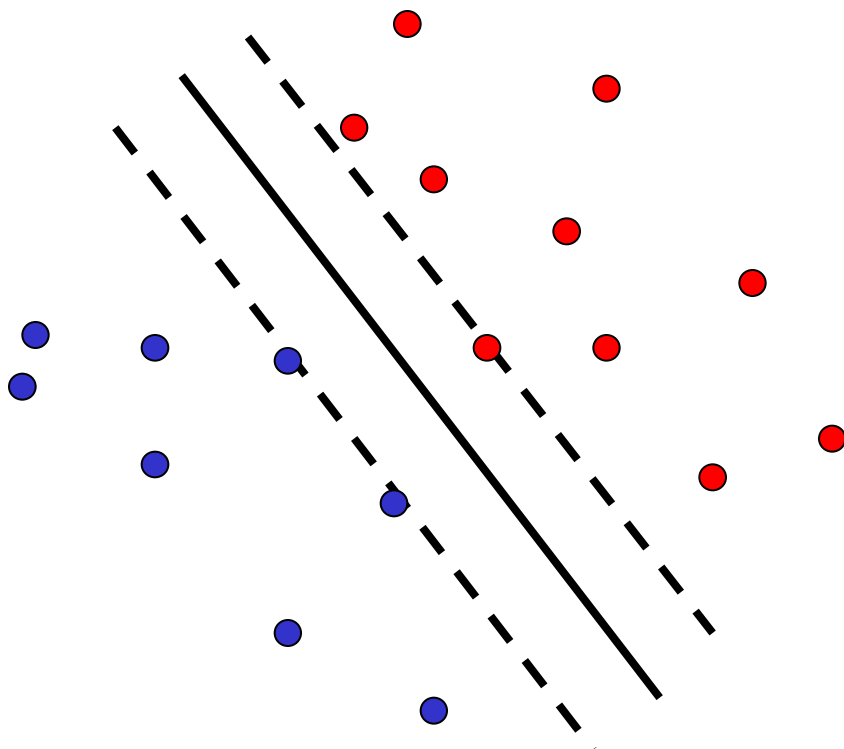
- Find hyperplane that maximizes the *margin* between the positive and negative examples



# Support vector machines

---

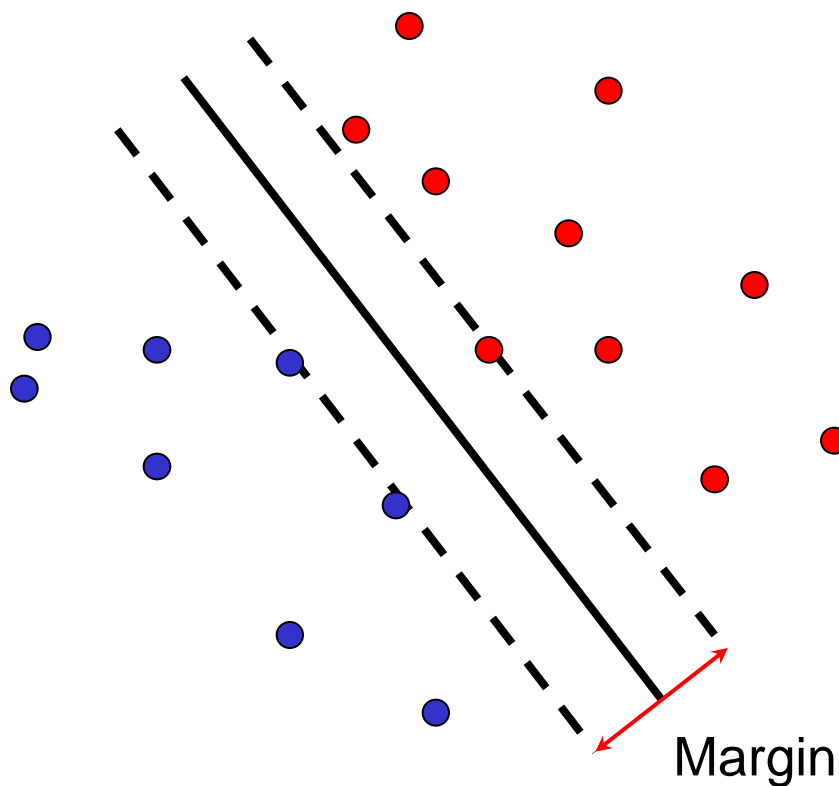
- Find hyperplane that maximizes the *margin* between the positive and negative examples



# Support vector machines

---

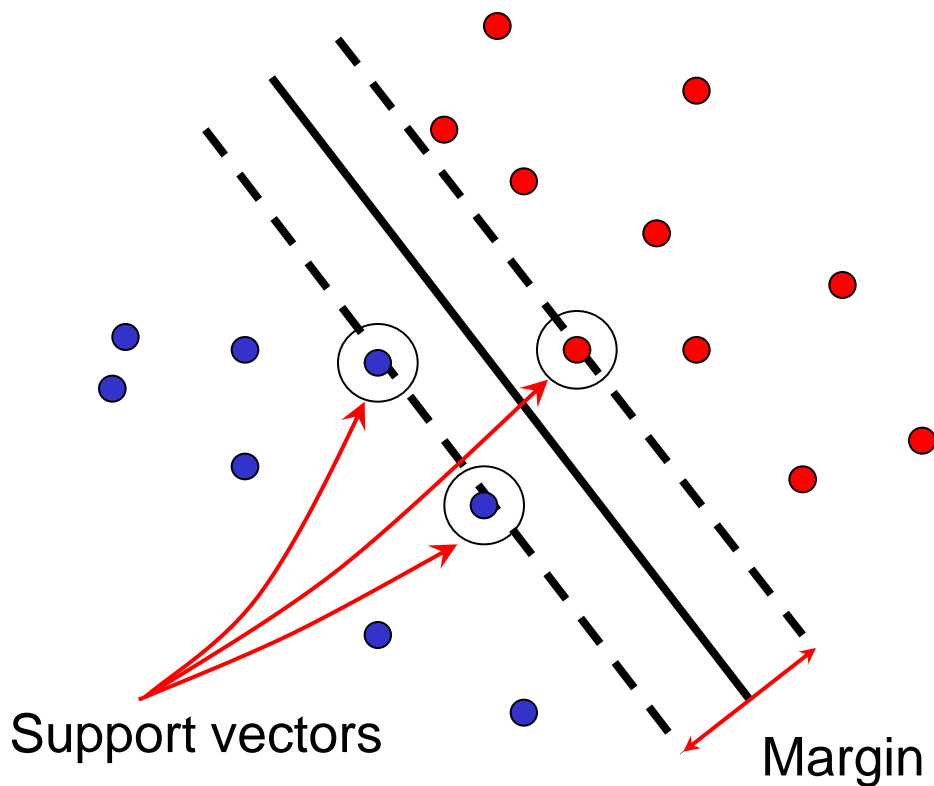
- Find hyperplane that maximizes the *margin* between the positive and negative examples



# Support vector machines

---

- Find hyperplane that maximizes the *margin* between the positive and negative examples

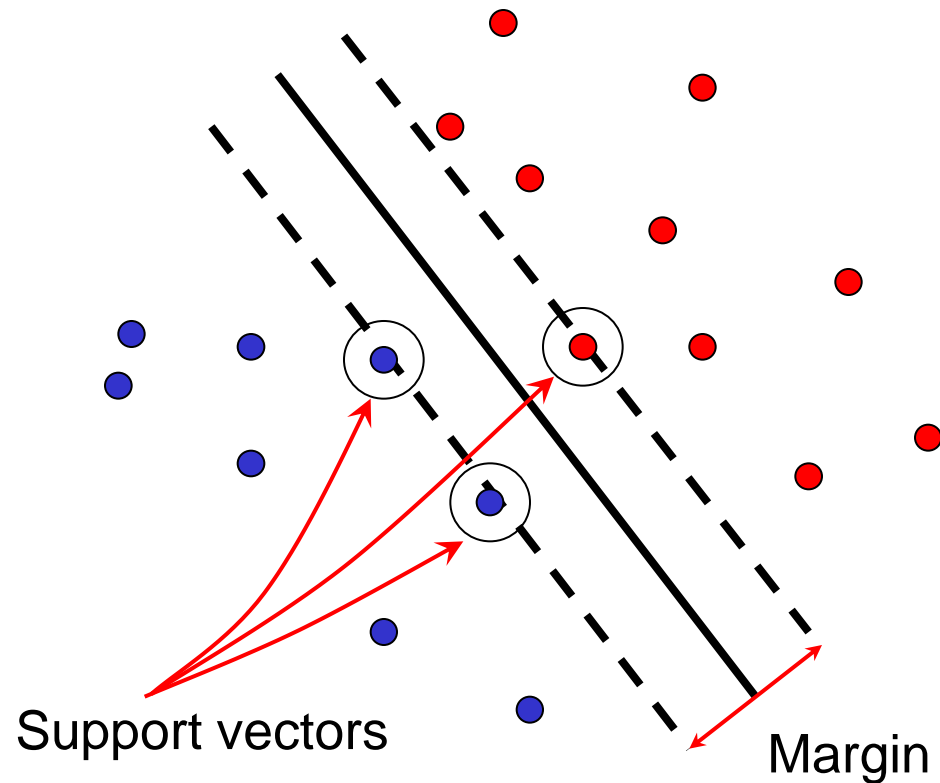




# Support vector machines

---

- Find hyperplane that maximizes the *margin* between the positive and negative examples



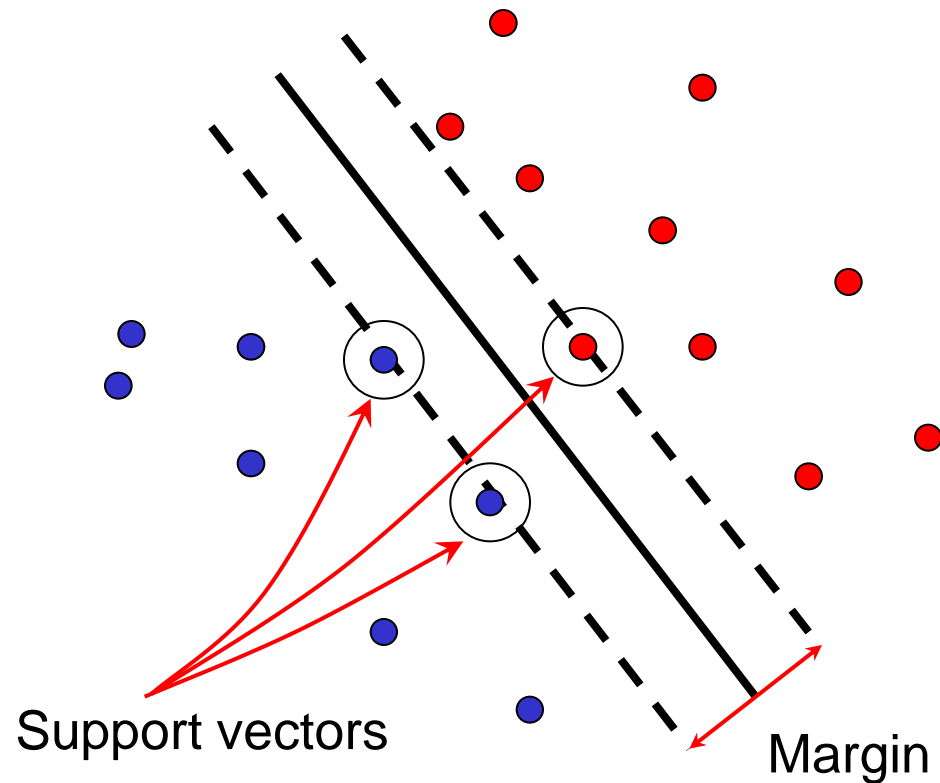
$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

# Support vector machines

---

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

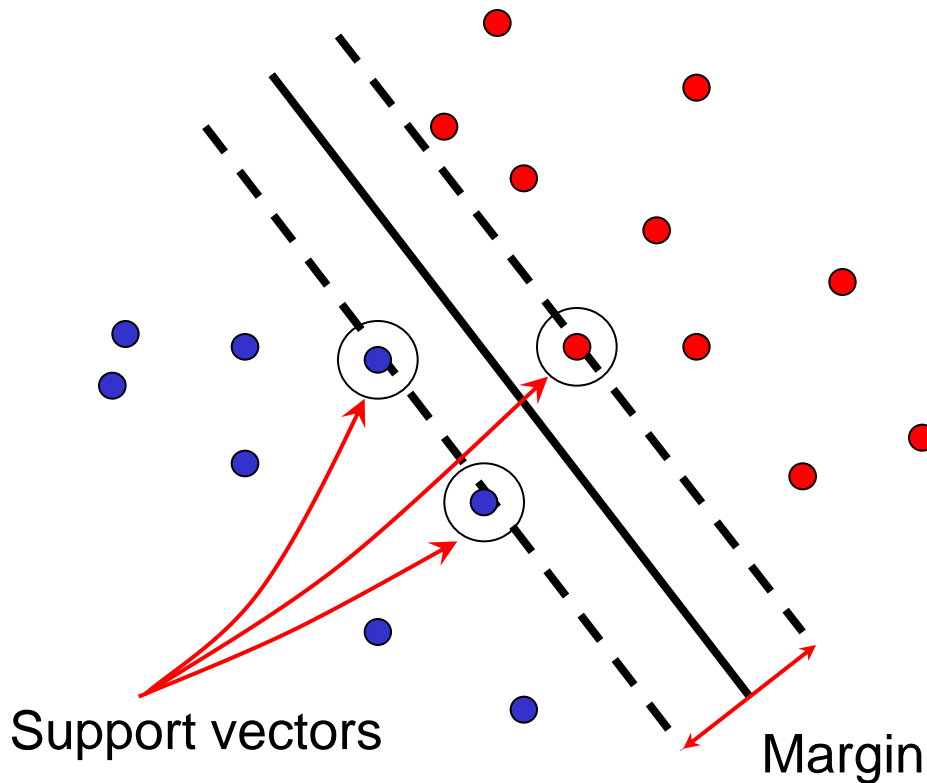
$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

$$\text{For support vectors, } \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$$

# Support vector machines

---

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

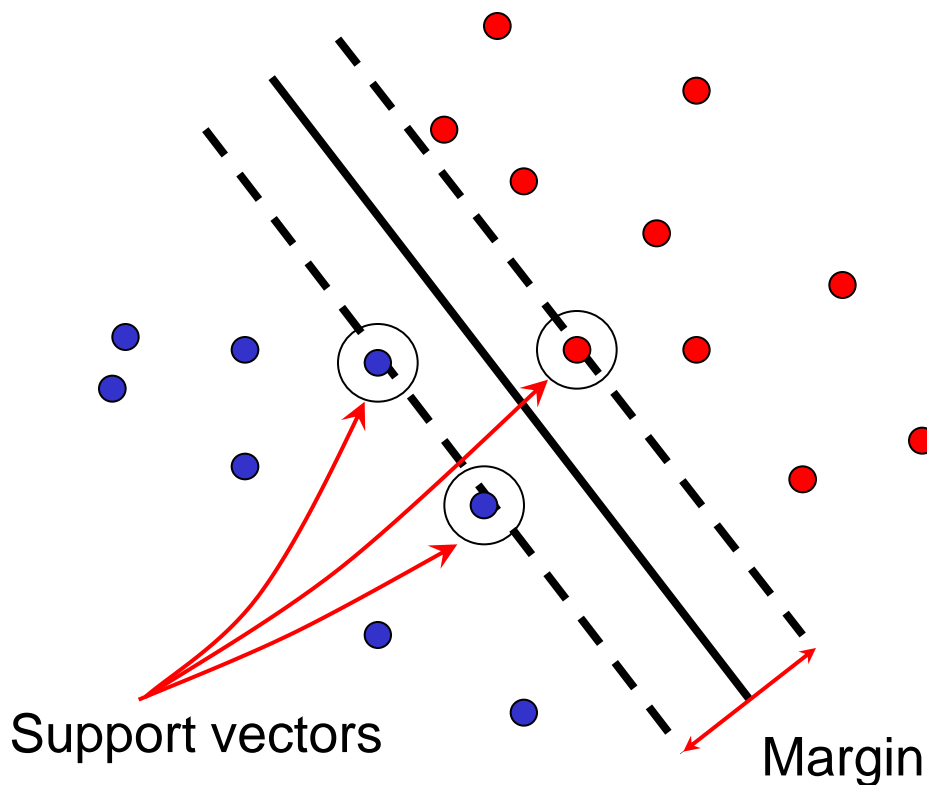
$$\text{For support vectors, } \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$$

$$\text{Distance between point and hyperplane: } \frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

# Support vector machines

---

- Find hyperplane that maximizes the *margin* between the positive and negative examples



$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

$$\text{For support vectors, } \mathbf{x}_i \cdot \mathbf{w} + b = \pm 1$$

$$\text{Distance between point and hyperplane: } \frac{|\mathbf{x}_i \cdot \mathbf{w} + b|}{\|\mathbf{w}\|}$$

$$\text{Therefore, the margin is } 2 / \|\mathbf{w}\|$$

# Finding the maximum margin hyperplane

---

1. Maximize margin  $2 / \|\mathbf{w}\|$

# Finding the maximum margin hyperplane

---

1. Maximize margin  $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

# Finding the maximum margin hyperplane

---

1. Maximize margin  $2 / \|\mathbf{w}\|$
2. Correctly classify all training data:

$$\mathbf{x}_i \text{ positive } (y_i = 1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \geq 1$$

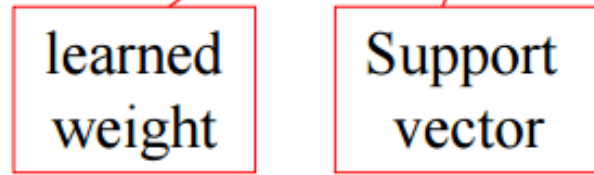
$$\mathbf{x}_i \text{ negative } (y_i = -1): \quad \mathbf{x}_i \cdot \mathbf{w} + b \leq -1$$

*Quadratic optimization problem:*

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

# Finding the maximum margin hyperplane

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$



- The weights  $\alpha_i$  are non-zero only at support vectors.



# Finding the maximum margin hyperplane

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$   
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$  (for any support vector)  
 $\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$

# Finding the maximum margin hyperplane

- Solution:  $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$   
 $b = y_i - \mathbf{w} \cdot \mathbf{x}_i$  (for any support vector)

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Classification function:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \quad \text{If } f(\mathbf{x}) < 0, \text{ classify as negative,}$$
$$= \text{sign}\left(\sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b\right) \quad \text{if } f(\mathbf{x}) > 0, \text{ classify as positive}$$



Dot product only!

# SVM parameter learning

---

- Separable data:  $\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$  subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$
- 
- The diagram illustrates the SVM parameter learning problem. It shows the minimization of the margin,  $\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$ , and the constraint for correct classification,  $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ . Red curly braces are used to group the terms. The first brace is under the minimization term, and the second brace is under the constraint term. Below each brace is a red-bordered box containing text.
- |                 |                                  |
|-----------------|----------------------------------|
| Maximize margin | Classify training data correctly |
|-----------------|----------------------------------|

# SVM parameter learning

---

- Separable data:  $\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2$  subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$

Maximize  
margin

Classify training data correctly

- Non-separable data:

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + b))$$

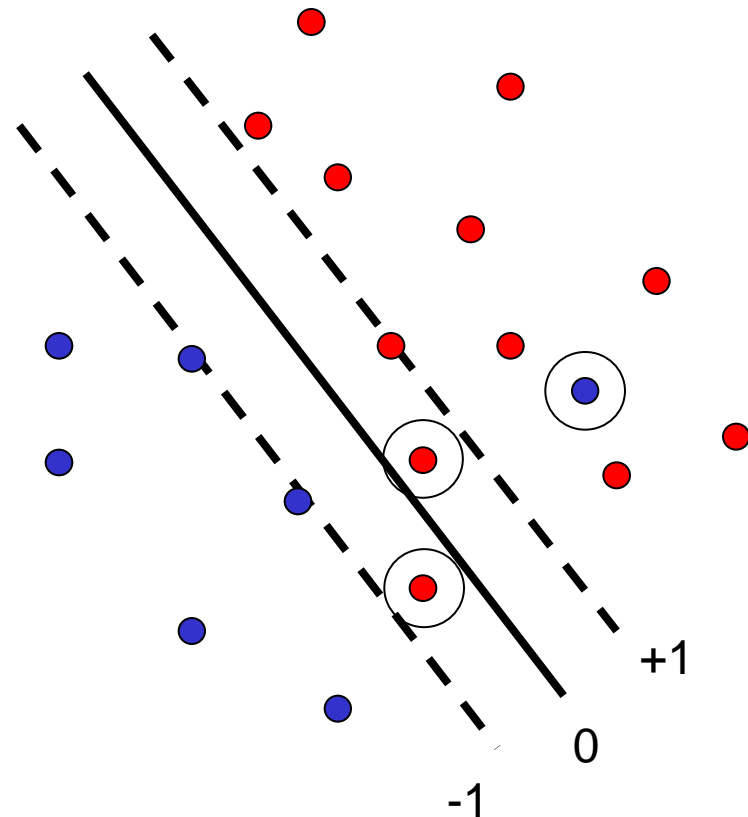
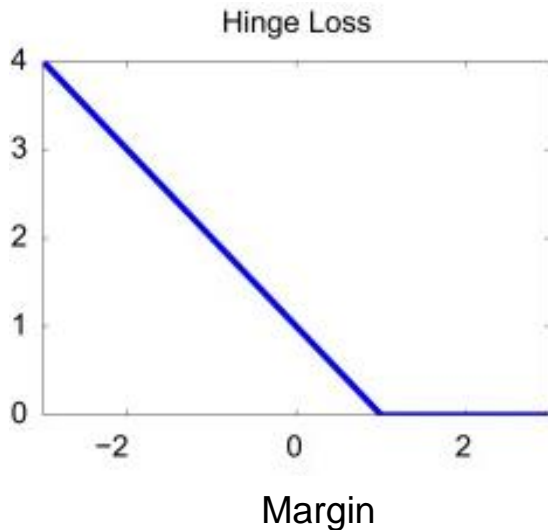
Maximize  
margin

Minimize classification mistakes

# SVM parameter learning

---

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$$

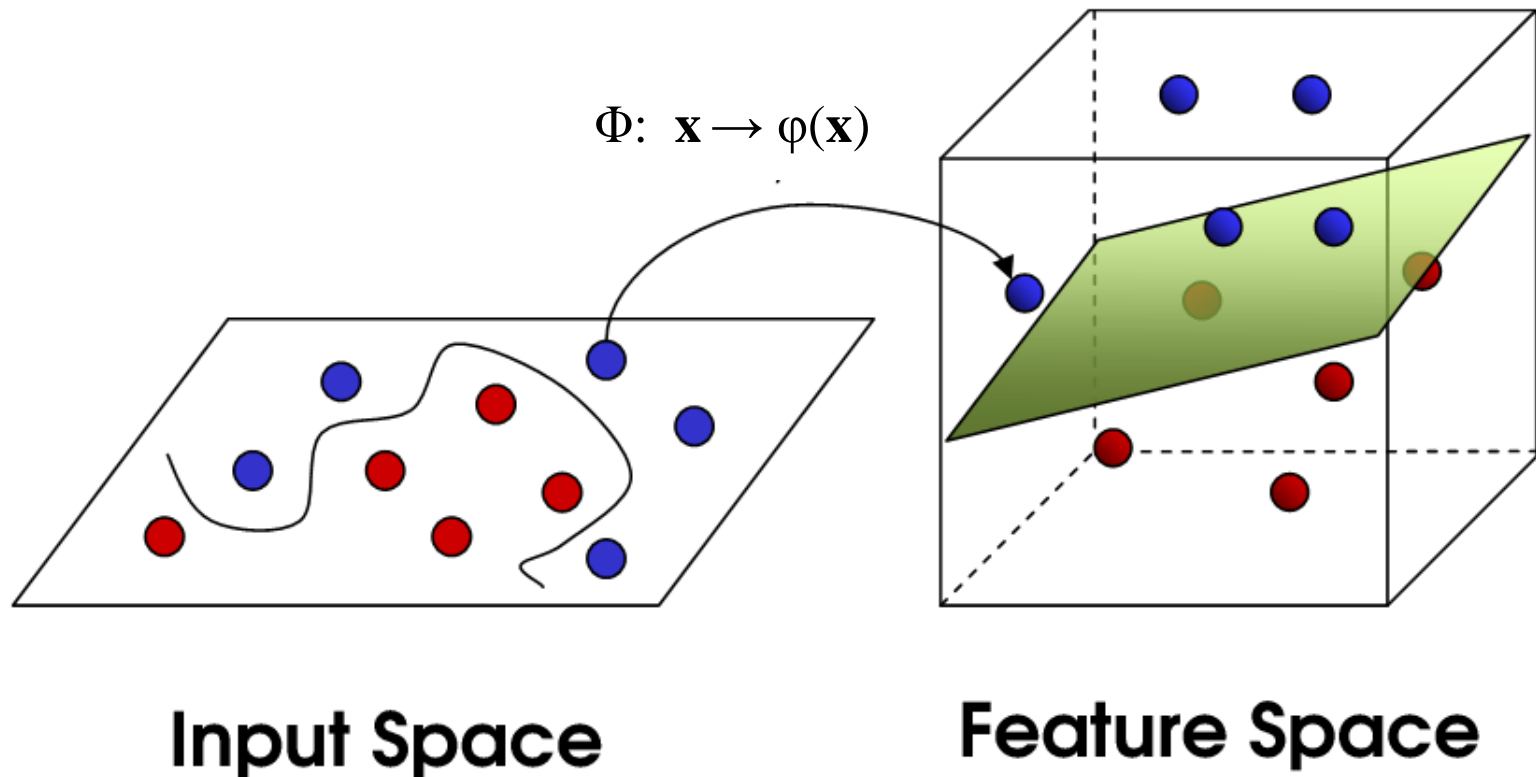


Demo: <http://cs.stanford.edu/people/karpathy/svmjs/demo>

# Nonlinear SVMs

---

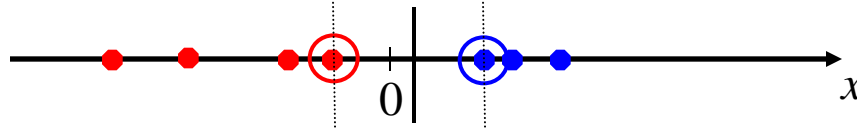
- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable



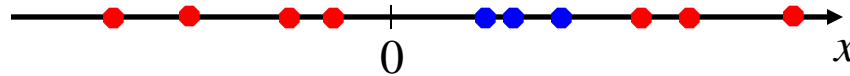
# Nonlinear SVMs

---

- Linearly separable dataset in 1D:



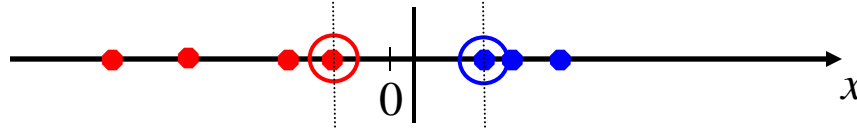
- Non-linearly separable dataset in 1D:



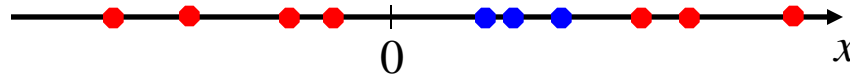
# Nonlinear SVMs

---

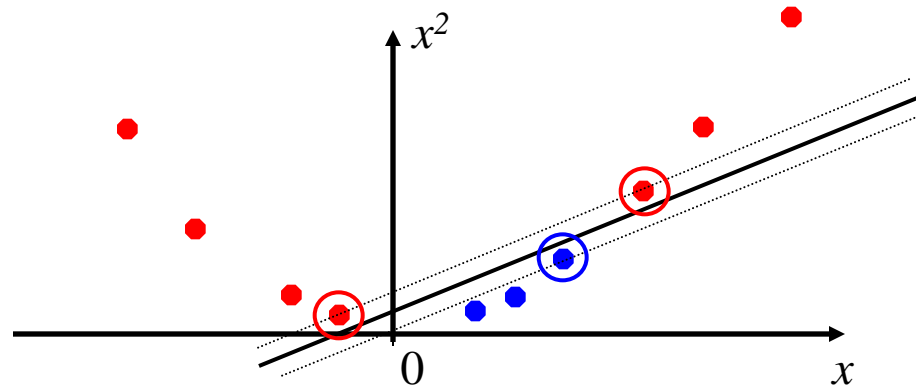
- Linearly separable dataset in 1D:



- Non-linearly separable dataset in 1D:



- We can map the data to a *higher-dimensional space*:





# The kernel trick

---

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable
- **The kernel trick:** instead of explicitly computing the lifting transformation  $\varphi(\mathbf{x})$ , define a kernel function  $K$  such that

$$K(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x}) \cdot \varphi(\mathbf{y})$$

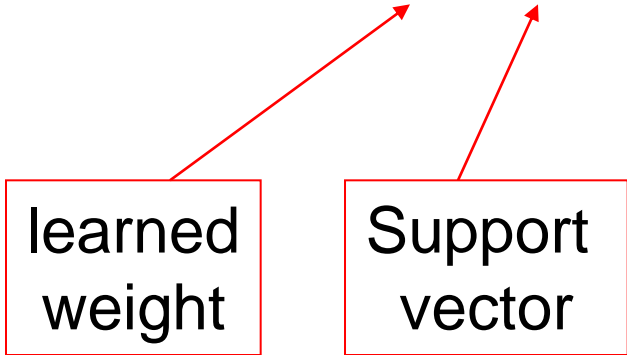
# The kernel trick

---

- Linear SVM decision function:

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

learned  
weight



Support  
vector

# The kernel trick

---

- Linear SVM decision function:

$$\mathbf{w} \cdot \mathbf{x} + b = \sum_i \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} + b$$

- Kernel SVM decision function:

$$\sum_i \alpha_i y_i \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}) + b = \sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

- This gives a nonlinear decision boundary in the original feature space

# Example

---

2-dimensional vectors  $\mathbf{x}=[x_1 \ x_2]$ ;

let  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

# Example

---

2-dimensional vectors  $\mathbf{x}=[x_1 \ x_2]$ ;

let  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ :

# Example

---

2-dimensional vectors  $\mathbf{x}=[x_1 \ x_2]$ ;

let  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ :

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \end{aligned}$$

# Example

---

2-dimensional vectors  $\mathbf{x}=[x_1 \ x_2]$ ;

let  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ :

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= \begin{bmatrix} 1 & x_{i1}^2 & \sqrt{2} x_{i1} x_{i2} & x_{i2}^2 & \sqrt{2} x_{i1} & \sqrt{2} x_{i2} \end{bmatrix}^T \\ &\quad \begin{bmatrix} 1 & x_{j1}^2 & \sqrt{2} x_{j1} x_{j2} & x_{j2}^2 & \sqrt{2} x_{j1} & \sqrt{2} x_{j2} \end{bmatrix} \end{aligned}$$

# Example

---

2-dimensional vectors  $\mathbf{x}=[x_1 \ x_2]$ ;

let  $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^2$

Need to show that  $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j)$ :

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= (1 + \mathbf{x}_i^T \mathbf{x}_j)^2, \\ &= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2} \\ &= [1 \ x_{i1}^2 \ \sqrt{2} x_{i1} x_{i2} \ x_{i2}^2 \ \sqrt{2} x_{i1} \ \sqrt{2} x_{i2}]^T \\ &\quad [1 \ x_{j1}^2 \ \sqrt{2} x_{j1} x_{j2} \ x_{j2}^2 \ \sqrt{2} x_{j1} \ \sqrt{2} x_{j2}] \\ &= \boldsymbol{\varphi}(\mathbf{x}_i)^T \boldsymbol{\varphi}(\mathbf{x}_j), \\ &\quad \text{where } \boldsymbol{\varphi}(\mathbf{x}) = [1 \ x_1^2 \ \sqrt{2} x_1 x_2 \ x_2^2 \ \sqrt{2} x_1 \ \sqrt{2} x_2] \end{aligned}$$



# SVMs: Pros and cons

---

- Pros

- Kernel-based framework is **very powerful**, flexible
- Training is convex optimization, **globally optimal solution** can be found
- SVMs work very well in practice, even with **very small training** sample sizes

- Cons

- No “direct” **multi-class SVM**, must combine two-class SVMs (e.g., with one-vs-others)
- **Computation, memory** (esp. for nonlinear SVMs)

# Multiclass Support Vector Machine loss

---

- $i^{\text{th}}$  example: image  $x_i$  and the label  $y_i$
- Score for the  $j^{\text{th}}$  class:  $s_j = f(x_i, W)_j$

# Multiclass Support Vector Machine loss

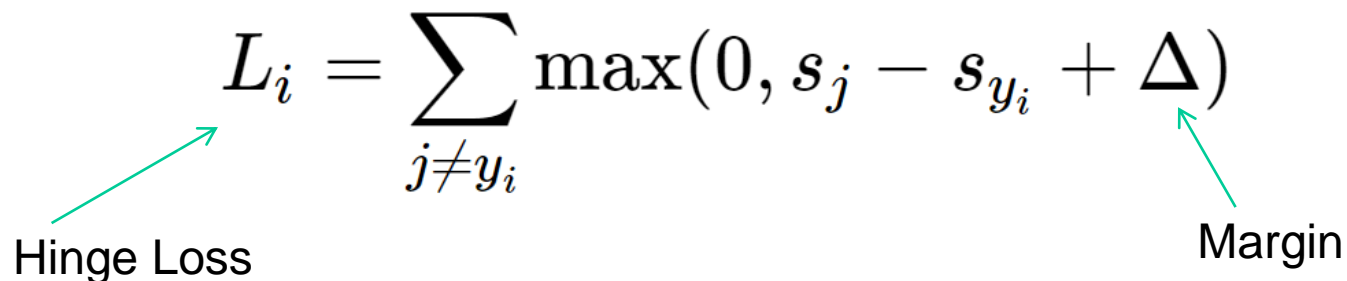
---

- $i^{\text{th}}$  example: image  $x_i$  and the label  $y_i$
- Score for the  $j^{\text{th}}$  class:  $s_j = f(x_i, W)_j$
- The Multiclass SVM loss for the  $i^{\text{th}}$  example is then formalized as follows:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Hinge Loss

Margin



# Multiclass Support Vector Machine loss

---

- $i^{th}$  example: image  $x_i$  and the label  $y_i$
- Score for the  $j^{th}$  class:  $s_j = f(x_i, W)_j$
- The Multiclass SVM loss for the  $i^{th}$  example is then formalized as follows:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Hinge Loss Margin

**Problem:  $W$  is not necessarily unique**

# Multiclass Support Vector Machine loss

---

- Regularization Penalty:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

# Multiclass Support Vector Machine loss

---

- Regularization Penalty:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

- Hinge Loss:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

# Multiclass Support Vector Machine loss

---

- Regularization Penalty:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

- Hinge Loss:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$



$$L = \frac{1}{N} \sum_i \sum_{j \neq y_i} [\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + \Delta)] + \lambda \sum_k \sum_l W_{k,l}^2$$

# Hinge Loss

matrix multiply + bias offset

0.01	-0.05	0.1	0.05
0.7	0.2	0.05	0.16
0.0	-0.45	-0.2	0.03

$W$

-15
22
-44
56

$x_i$

+

0.0
0.2
-0.3

$b$

$y_i$

2

-2.85
0.86
0.28

hinge loss (SVM)

$$\begin{aligned} &\max(0, -2.85 - 0.28 + 1) + \\ &\max(0, 0.86 - 0.28 + 1) \\ &= \\ &\mathbf{1.58} \end{aligned}$$



# Softmax Classifier

---

- Interprets the class scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

$$L_i = -\log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) = -s_{y_i} + \log \sum_j e^{s_j}$$

# Softmax Classifier

---

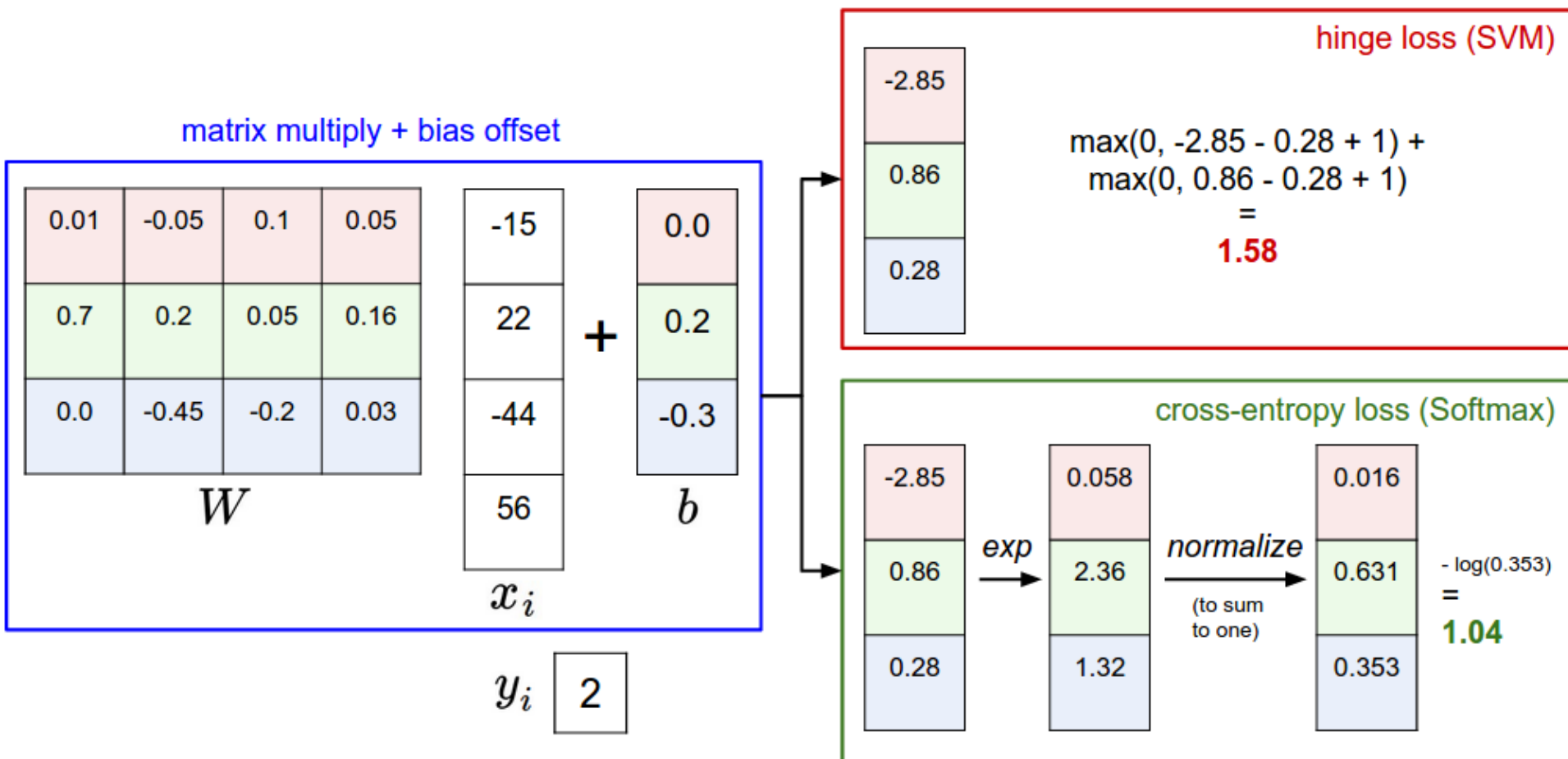
- Interprets the class scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

$$L_i = -\log \left( \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \right) = -s_{y_i} + \log \sum_j e^{s_j}$$

- Softmax Loss:

$$L = \underbrace{\frac{1}{N} \sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

# Hinge vs Cross-entropy Loss



# Acknowledgements

---

Thanks to the following researchers for making their teaching/research material online

- Forsyth
- Steve Seitz
- Noah Snavely
- J.B. Huang
- Derek Hoiem
- D. Lowe
- A. Bobick
- S. Lazebnik
- K. Grauman
- R. Zaleski
- Antonio Torralba
- Rob Fergus
- Leibe
- And many more .....

# Next Lecture

---

## Neural Networks

