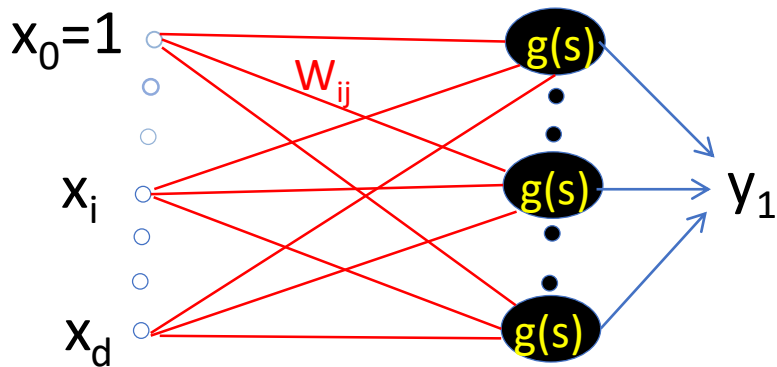


# Lecture 4

Random projections  
Support vector machines

# Random Projections

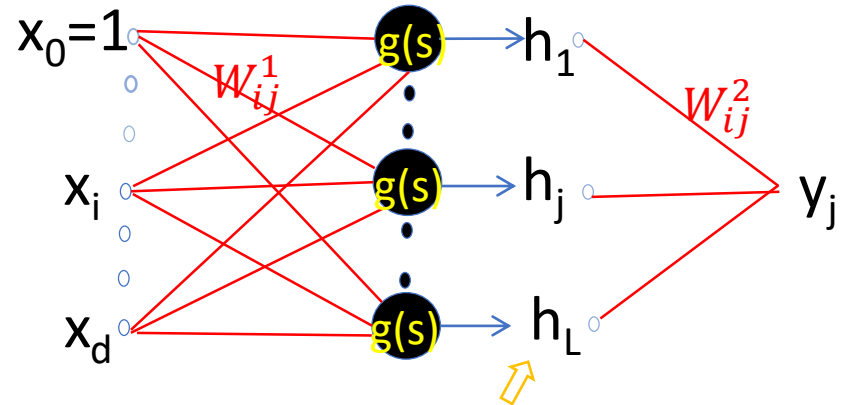
## Multiple Neurons



Training set

$$\vec{x}^{(m)} \rightarrow \vec{y}^{(m)}, m = 1 \text{ to } M$$

## Random Projection to Z-space



Hidden layer, number of hidden units:  $L$

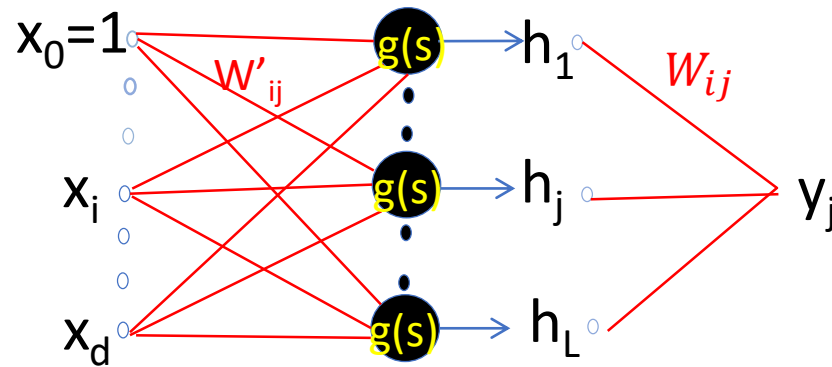
Training set:

$$X_{M \times d} \rightarrow Y_M$$

Hidden layer: A **random nonlinear** transform of input  $H_{M \times L} = g(X_{M \times d} W_{d \times L}^1)$

$W^1 = \{w_{ij}\}$  : random weights

# Random Projection



$$\hat{y}_{M \times m} = H_{N \times L} W \rightarrow \min\{\|\hat{y} - y\|^2\} \rightarrow \min\{E(W) = \|HW - y\|^2\}$$

$$\nabla E(W) = 2H^T(HW - y) = 0 \Rightarrow W_{L \times m} = (H^T H)^{-1} H^T y$$



Pseudo-inverse

Can be also solved iteratively

- One-step learning  $\rightarrow$  no iteration required
- Extremely fast
- No hyper-parameters  $\rightarrow$  no need to tune
- Nonlinearity applies with random weights  $\rightarrow$  less degrees of freedom  $\rightarrow$  better for generalization

# Direct inversion

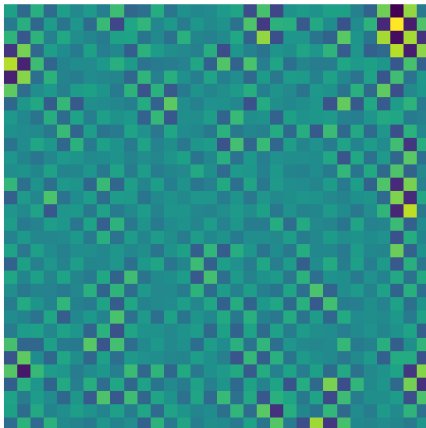
Weight calculation by matrix inversion

$$\underline{\underline{X}}\underline{\underline{w}} = \underline{\underline{t}} \Rightarrow \underline{\underline{w}} = \underline{\underline{X}}^{-1}\underline{\underline{t}}$$

$\underline{\underline{X}}$  is 1024 by 1024

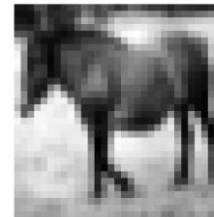
Images used for weight calculation	100%
Test 2000 new images	52%

Weights ( $\underline{\underline{w}}$ )

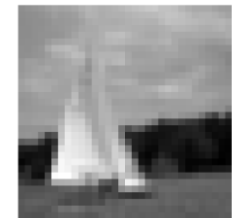


Database  
1024 training images  
2000 test images

Class 1



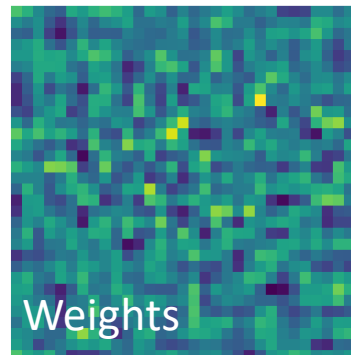
Class 2



Perceptron

Training accuracy: 51.00%  
Test accuracy: 50.50%

1000 iterations

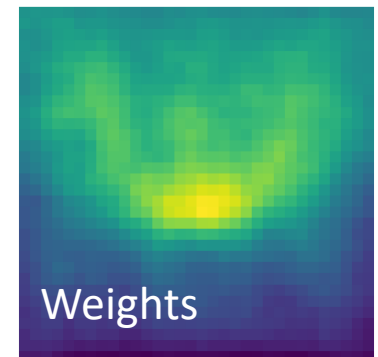


Weights

ADALINE

Training accuracy: 73.44%  
Test accuracy: 71.50%

Learning rate = 0.0001  
2000 iterations

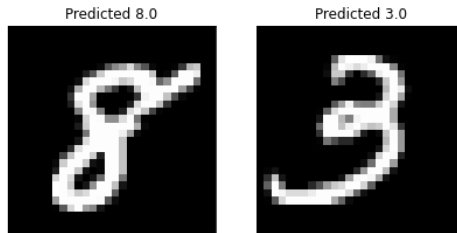


Weights

# Classification using Random Projections

Training dataset:  
12665 images, 784 pixels

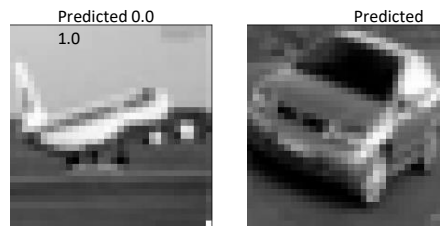
Test dataset:  
2115 images, 784 pixels



MNIST	Training acc	Test acc
Adaline (40 epochs)	98.4	98.5
random (128 units)	97.2	97.1

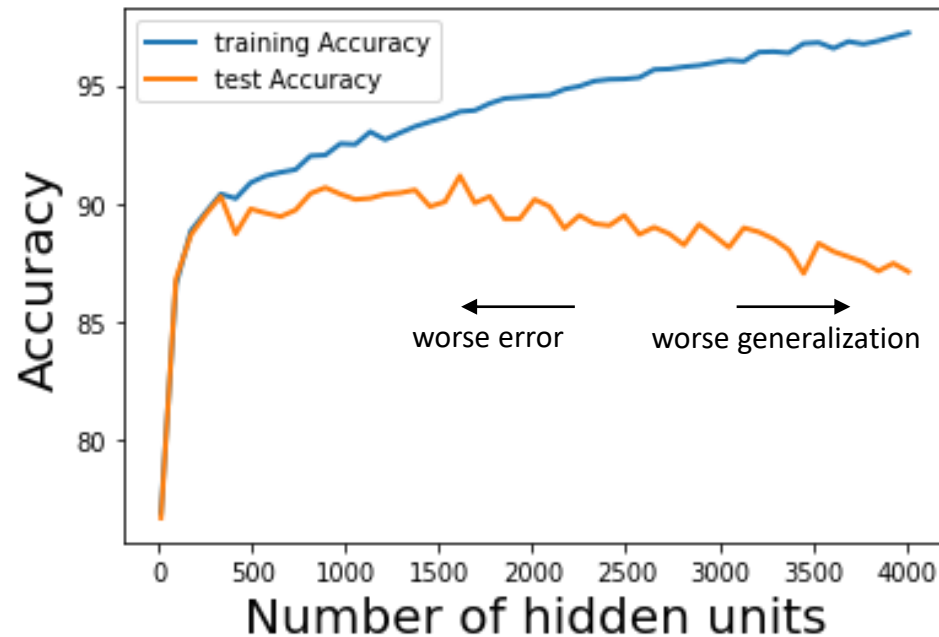
Training dataset:  
20000 images, 784 pixels

Test dataset:  
4000 images, 1024 pixels



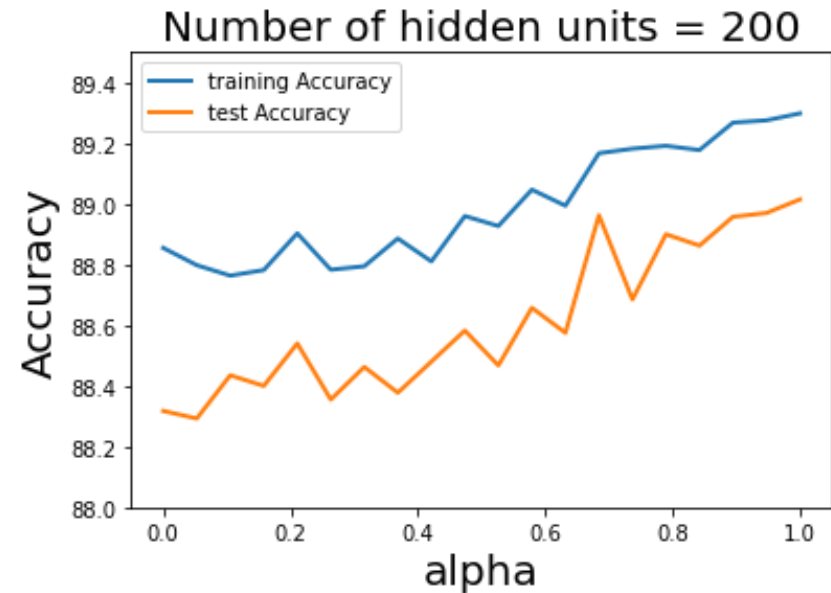
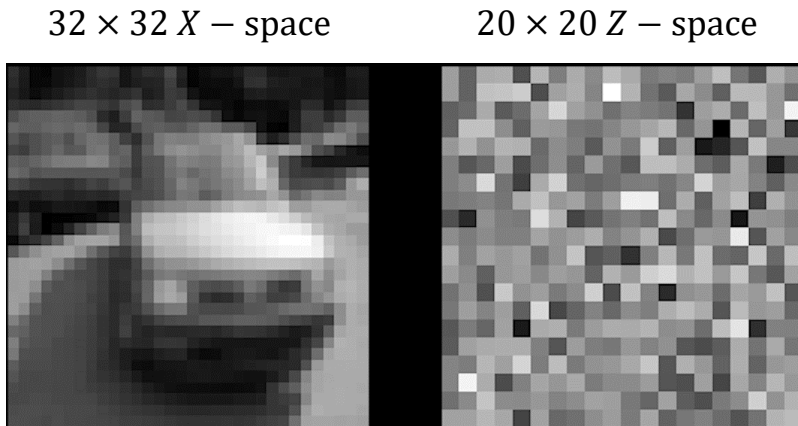
CIFAR10	Training acc	Test acc
Adaline (40 epochs)	76.3	75.6
random (512 units)	91.3	90.7

For Cifar10 data-set



# Random Projection to Z-space

Dimension Reduction:  $1024 \rightarrow 400$



Sigmoid:  $g(s) = \frac{1}{1 + e^{-\alpha s}}$

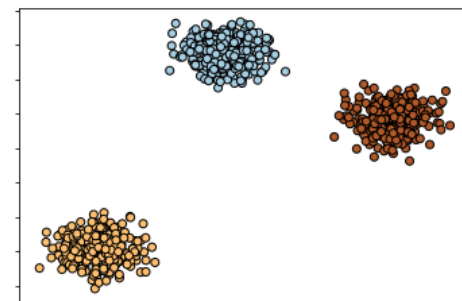
# Random Projection to Z-space

Considering  $m$  examples in  $X \in \mathbb{R}^N$ , and a given number  $0 < \varepsilon < 1$  and  $\varepsilon^2 > \frac{8\log(m)}{d}$ , there is a linear random map,  $W: X \rightarrow Z \in \mathbb{R}^d$  for which:

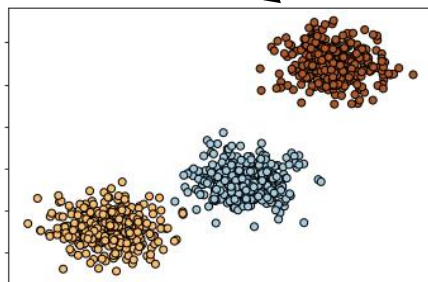
$$(1 - \varepsilon)\|X_1 - X_2\| < \|Z_1 - Z_2\| < (1 + \varepsilon)\|X_1 - X_2\|$$

For all  $X_1, X_2 \in X$

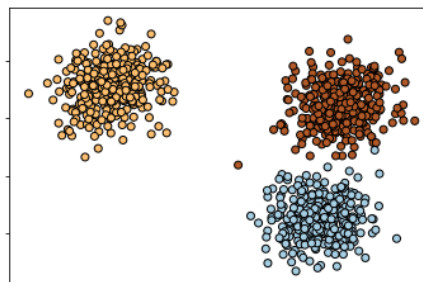
$X$ : 100 samples, 1000 features



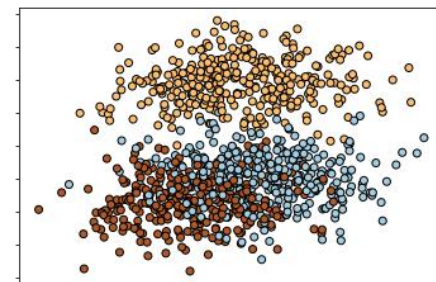
$W: 1000 \times 100$



$W: 1000 \times 60$



$W: 1000 \times 20$

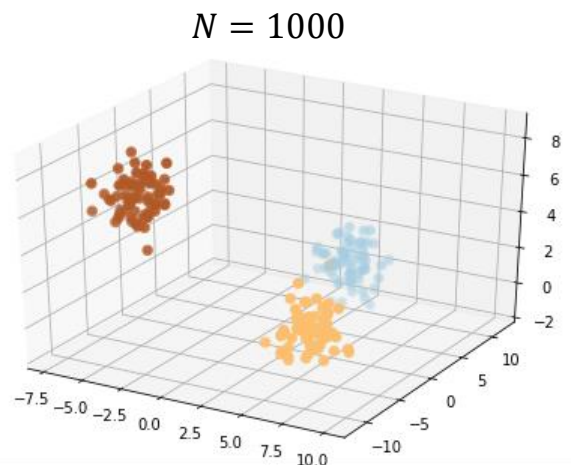


Z - Space

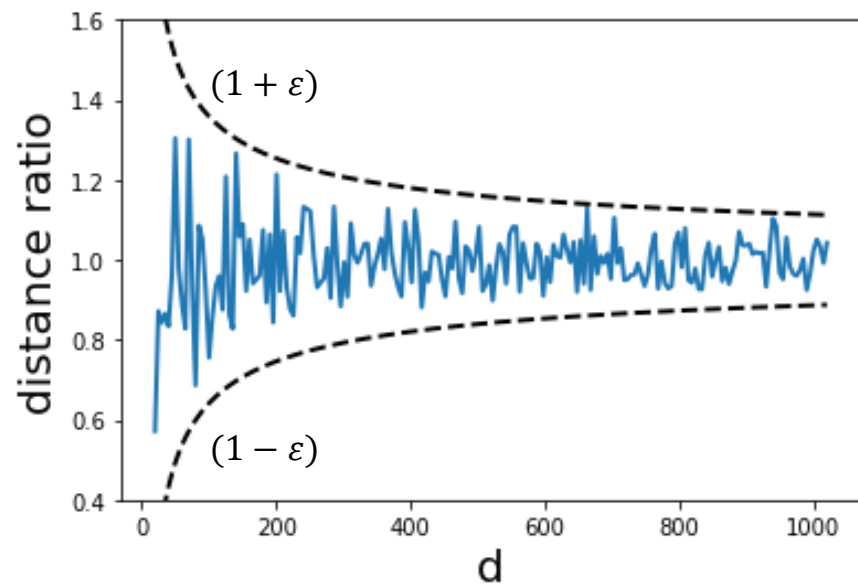


# Random Projection to Z-space

$$(1 - \varepsilon)\|X_1 - X_2\| < \|Z_1 - Z_2\| < (1 + \varepsilon)\|X_1 - X_2\|$$



random mapping to  $\mathbb{R}^d$



$N = 1000$ : dimension of the input

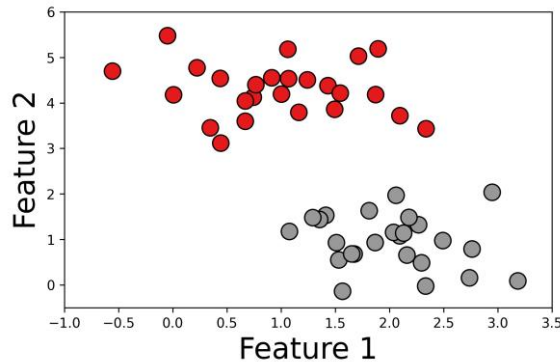
$d$ : dimension of the hidden unit

# Support Vector Machines

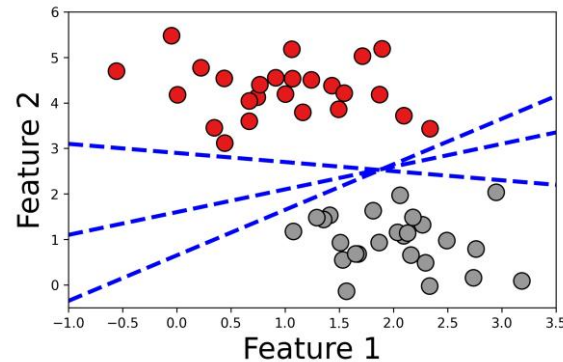
**SVM**

# Support Vector Machines

Data

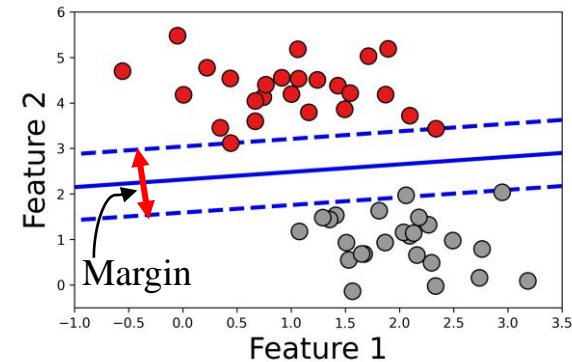


Lines that separate the classes



Many possible solutions

Line with the maximum margin



SVM finds an optimal solution by maximizing the margin

Finding a line ( $\mathbf{w}$ ,  $b$ ) such that:

$\mathbf{w} \cdot \mathbf{x} + b > 0$  For red points

$\mathbf{w} \cdot \mathbf{x} + b < 0$  For grey points

$\mathbf{w}$  is the multiplier vector,  $\mathbf{x}$  is the feature vector, and  $b$  is the bias.

Today, the vector  $\mathbf{w}$  does not include  $w_0$ . The bias  $b$  is indicated separately

$$\mathbf{w} = \sum_m^M c^{(m)} y^{(m)} \mathbf{x}^{(m)}$$

$c$  : non zero for points on the boundary

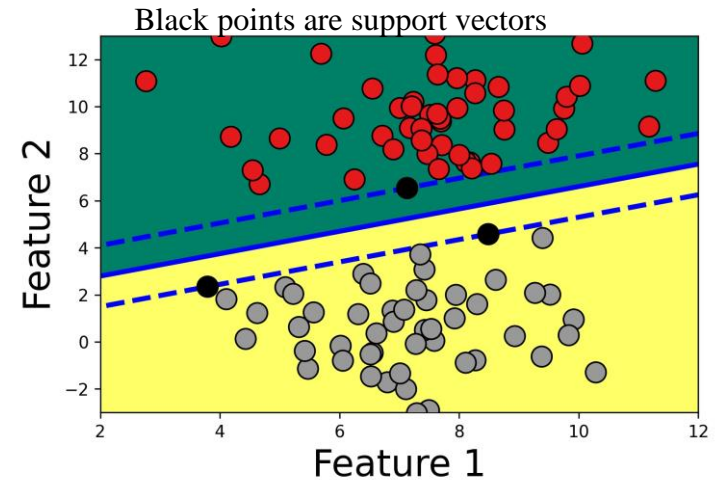
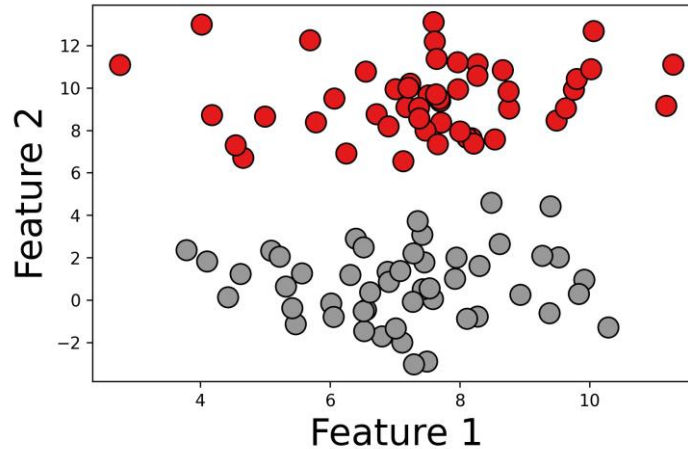
$y^{(m)} = [-1, 1]$  : labels

$\mathbf{x}^{(m)}$  : feature vector

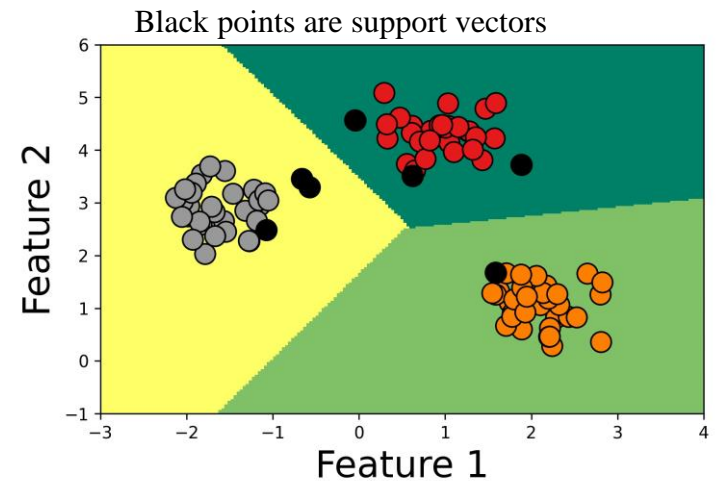
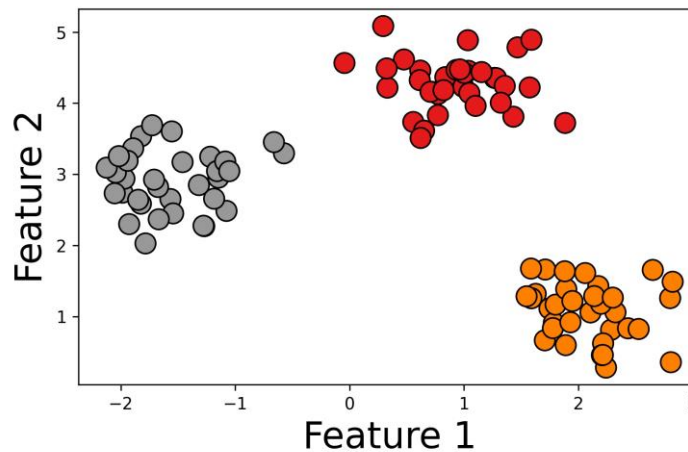
$\mathbf{w}$  : weight vector (multiplier vector)

# Support Vector Machines: Linearly separable training set

2 classes

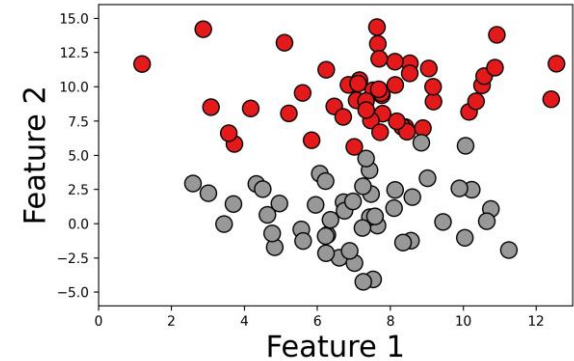


3 classes

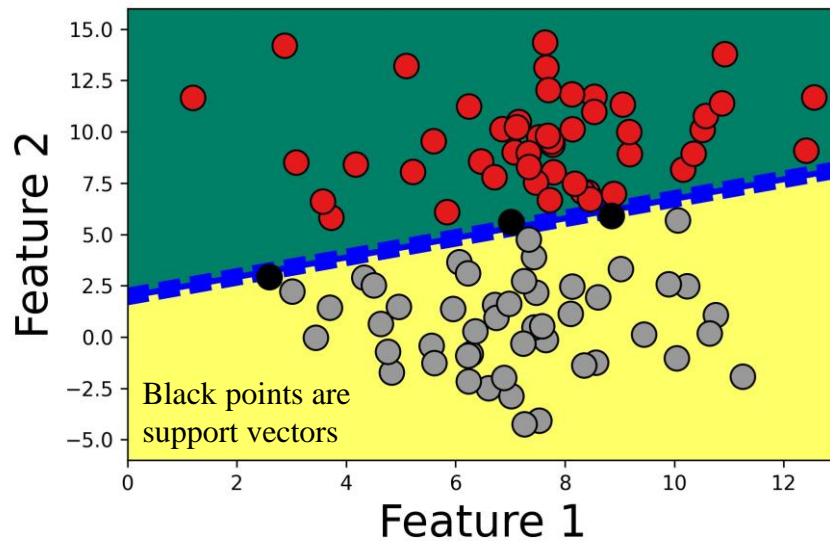


# Support Vector Machines: Margin

Data

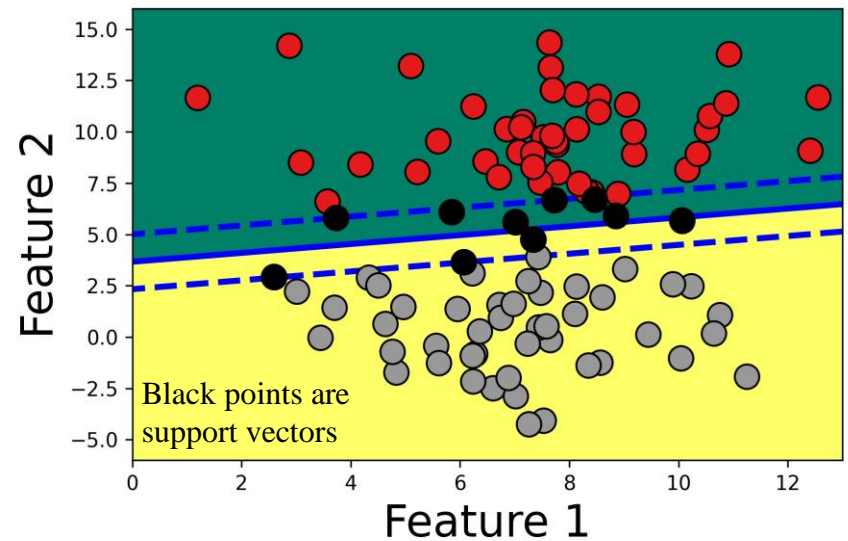


Hard margin



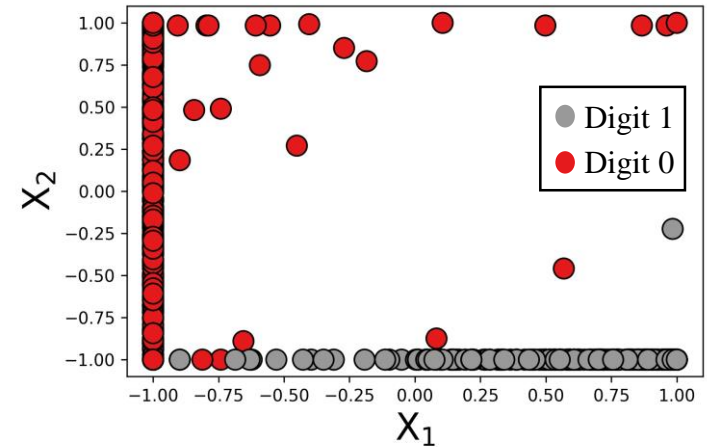
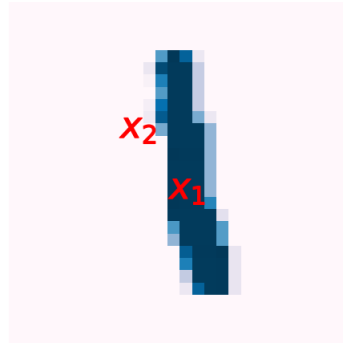
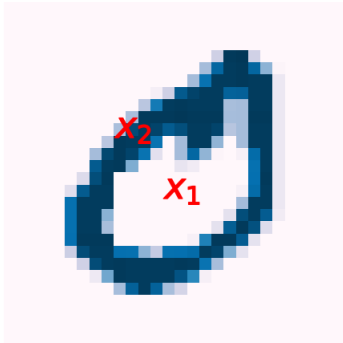
$$y^{(m)}(\mathbf{w} \cdot \mathbf{x}^{(m)} + b) > 1$$

Soft margin



$$y^{(m)}(\mathbf{w} \cdot \mathbf{x}^{(m)} + b) > 1 - \zeta^{(m)}$$

# Support Vector Machines: classification – two features



Training: 2000 samples

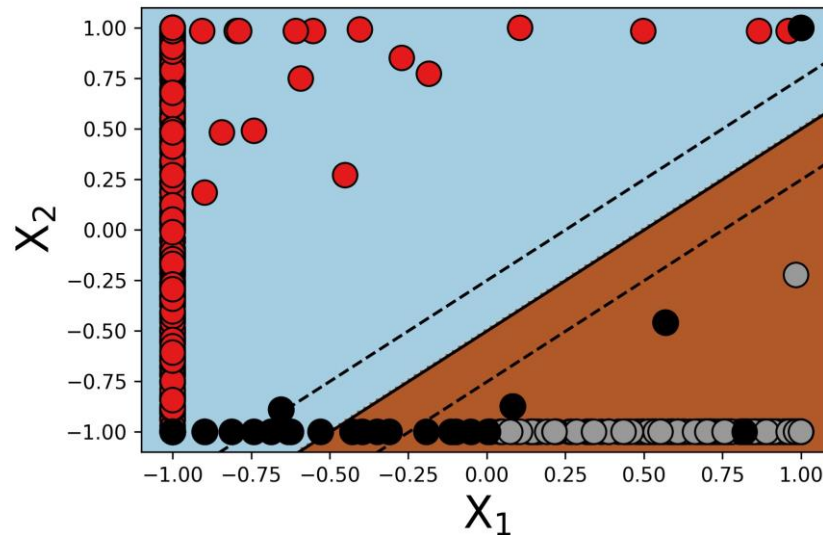
Test: 2115 samples

Pixel size: 28x28

Test accuracy: 99.20 %

Number of support vectors: 39

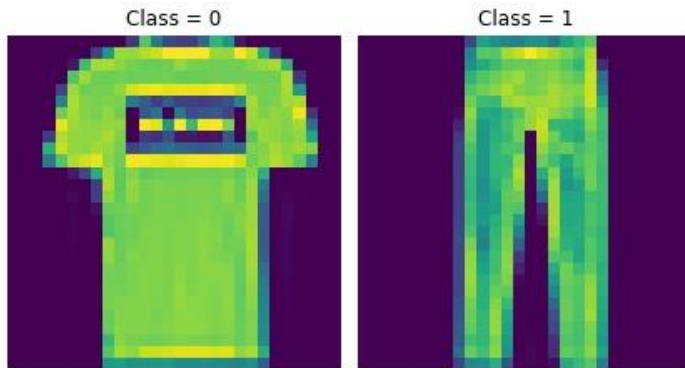
SVM Results



Black points are support vectors

# Support Vector Machines: classification

MNIST fashion: 2 classes



Training: M=12000 samples

Test: 2000 samples

Pixel size: 28x28

$$E[E_{out}] \leq E\left[\frac{\# c}{M-1}\right]$$

$E[]$  is the expected value

# c is the number of support vectors

M is the number of samples

$E_{out}$  is the classification error

ADALINE

Train accuracy	98.02 %
Test accuracy	97.7 %
Training time	9.99 s

Softmax activation function

Cross entropy cost function

Learning rate = 0.01

10 epochs

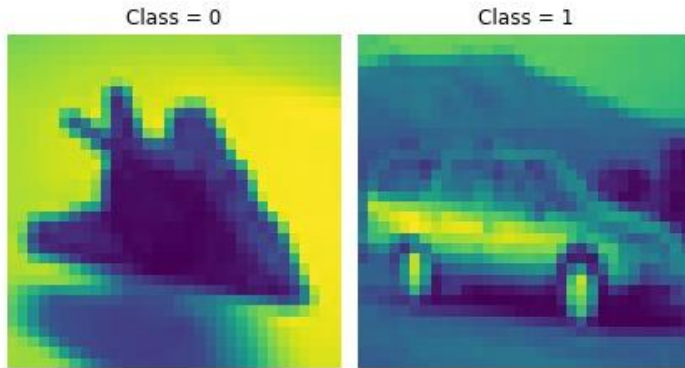
SVM

Train accuracy	99.84 %
Test accuracy	98.25 %
Training time	6.6 s

C=number of support vectors: 383

# Support Vector Machines: classification

CIFAR 10: 2 classes



Training: **M=10000** samples

Test: 2000 samples

**Pixel size: 32x32**

$$E[E_{out}] \leq E\left[\frac{\# c}{M-1}\right]$$

ADALINE

Train accuracy	74.77 %
Test accuracy	74.5 %
Training time	8.28 s

Softmax activation function

Cross entropy cost function

Learning rate = 0.01

10 epochs

SVM

Train accuracy	82.36 %
Test accuracy	78.10 %
Training time	122.48 s

**C=Number of support vectors: 4940**

M~2C



# Support Vector Machines: classification

MNIST fashion: 10 classes



Training: **M=60000** samples

Test: 10000 samples

Pixel size: **28x28**

ADALINE

Train accuracy	82.712 %
Test accuracy	81.31 %
Training time	48.91 s

Softmax activation function

Cross entropy cost function

Learning rate = 0.01

10 epochs

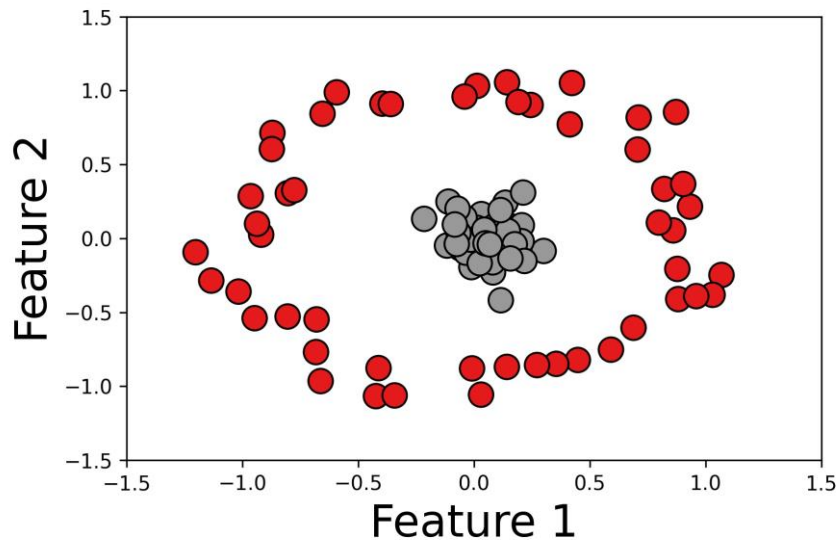
SVM

Train accuracy	90.23 %
Test accuracy	85.04 %
Training time	671 s

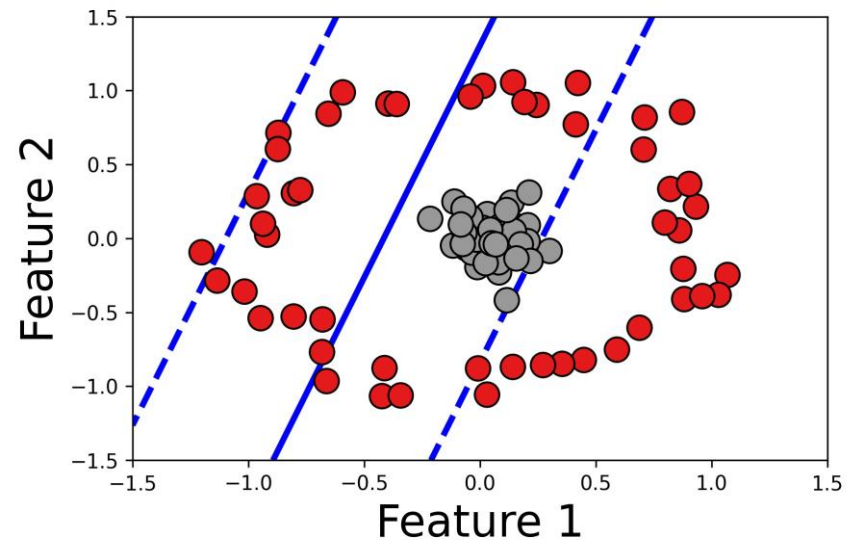
Number of support vectors: **C=18265**

# Support Vector Machines: non-separable clusters

Data not linearly separable



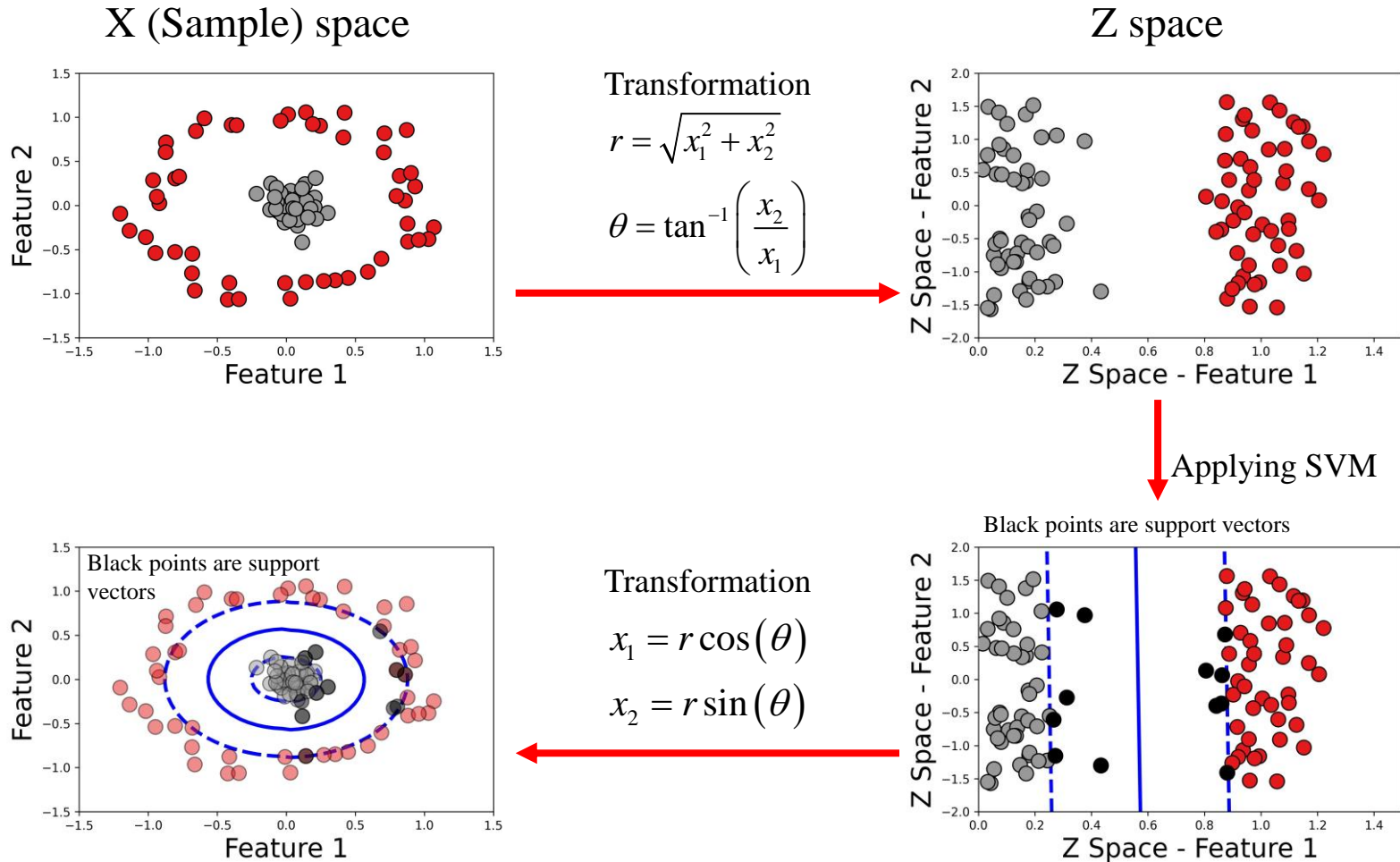
There is no line to separate the clusters



SVM accuracy = 68 %

The goal is to map the data into a new space where they are linearly separable.

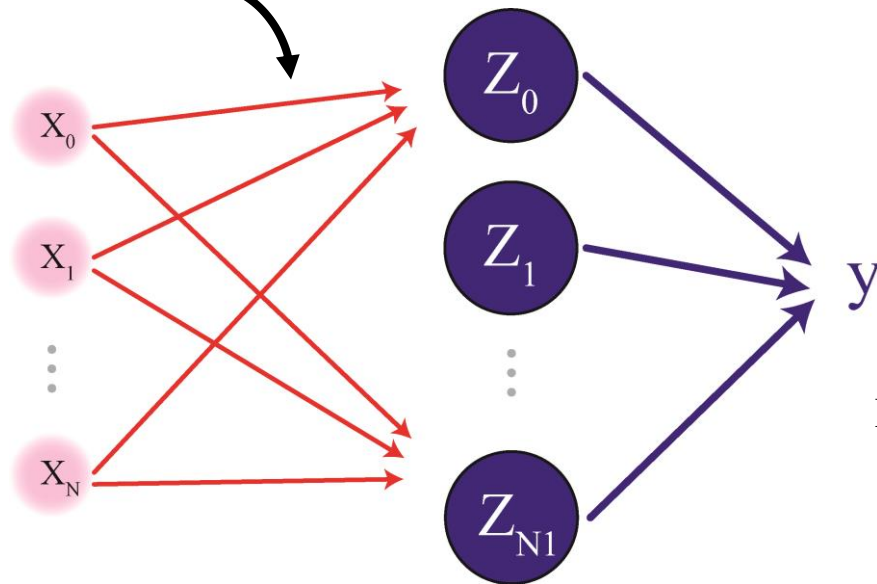
# Support Vector Machines: non-linear transformation



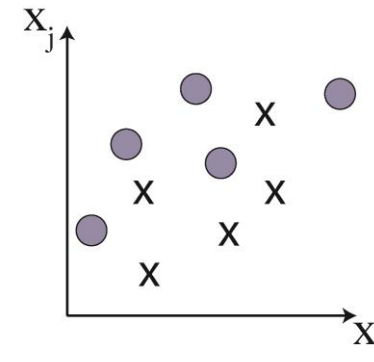
SVM accuracy = 100 %

# Support Vector Machines: non-linear transformation

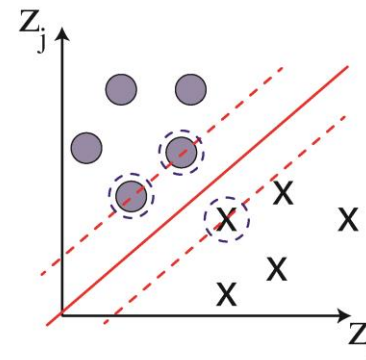
Fixed weights



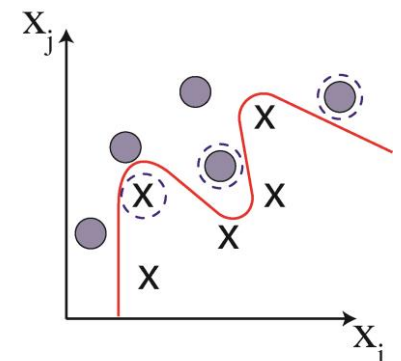
Initial data distribution



Data distribution in Z space



Decision line in X space



Support vectors are shown by dashed blue circles

# Support Vector Machines: kernel trick

The multiplier vector or the weights are:

$$\mathbf{w} = \sum_m^M c^{(m)} y^{(m)} Z(\mathbf{x}^{(m)})$$

How do we calculate the weights if we do not know the transformation?

We do not calculate the weights. We calculate the labels directly:

$$\text{prediction} = \mathbf{w} \cdot Z(\mathbf{x}^{(m_1)}) + b = \sum_m^M c^{(m)} y^{(m)} Z(\mathbf{x}^{(m)}) \cdot Z(\mathbf{x}^{(m_1)}) + b$$

$$\rightarrow \text{prediction} = \sum_m^M c^{(m)} y^{(m)} K(\mathbf{x}^{(m)}, \mathbf{x}^{(m_1)}) + b$$

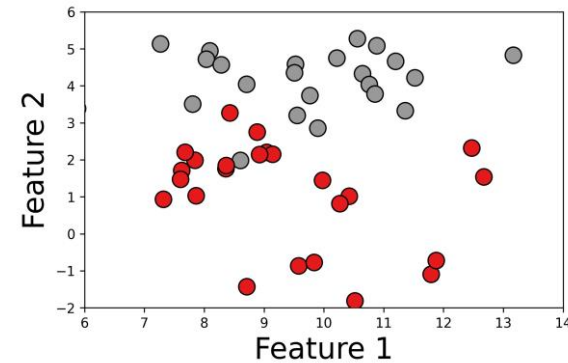
We define K as the kernel. We can calculate the prediction value using the kernel function.

# Support Vector Machines: Radial basis function (RBF) kernel

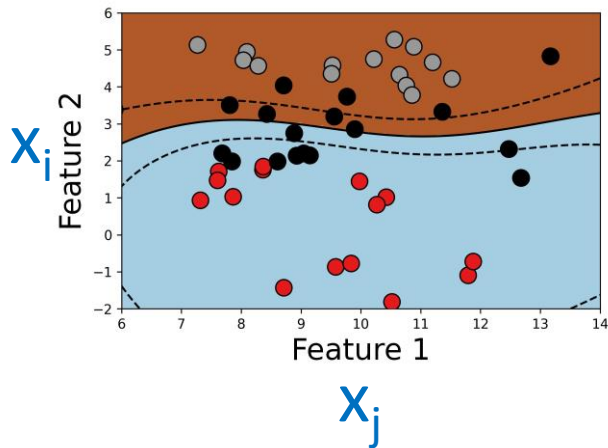
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

Effect of  $\gamma$

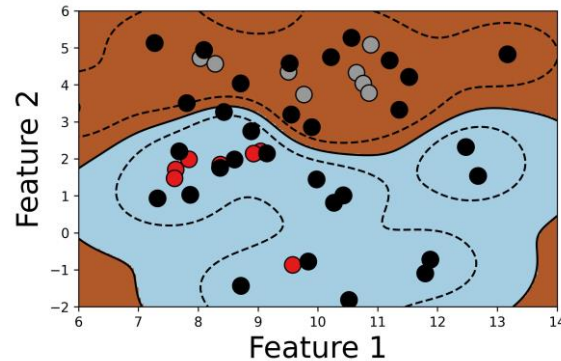
Data



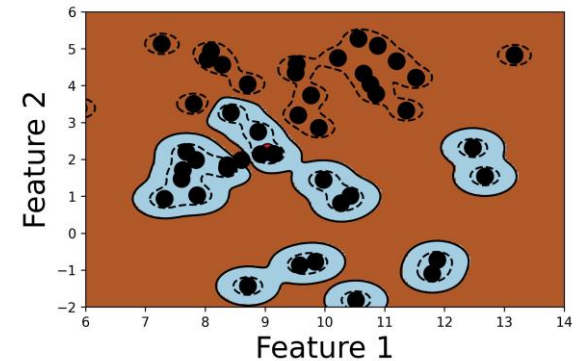
$\gamma = 0.1$



$\gamma = 1$



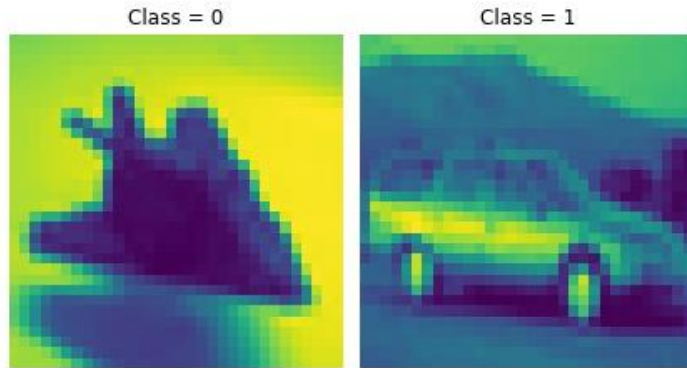
$\gamma = 10$



Black points are support vectors

# Support Vector Machines: classification – RBF kernel

CIFAR 10: 2 classes



Training: 10000 samples

Test: 2000 samples

Pixel size: 32x32

## ADALINE

Train accuracy	74.77 %
Test accuracy	74.5 %
Training time	8.28 s

Softmax activation function

Cross entropy cost function

Learning rate = 0.01

10 epochs

## SVM without kernel

Train accuracy	82.36 %
Test accuracy	78.10 %
Training time	122.48 s

Number of support vectors: 4940

## SVM with RBF kernel ( $\gamma=0.02$ )

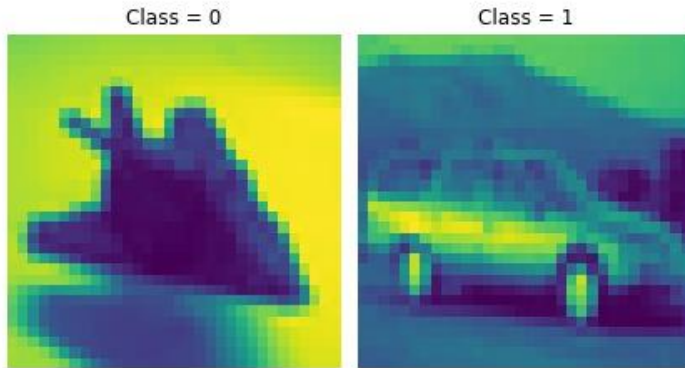
Train accuracy	94.19 %
Test accuracy	87.45 %
Training time	70.6 s

Number of support vectors: 4834

# Support Vector Machines: classification – RBF kernel

## Reducing the value of Gamma

### CIFAR 10: 2 classes



Training: 10000 samples

Test: 2000 samples

Pixel size: 32x32

### ADALINE

Train accuracy	74.77 %
Test accuracy	74.5 %
Training time	8.28 s

Softmax activation function

Cross entropy cost function

Learning rate = 0.01

10 epochs

### SVM without kernel

Train accuracy	82.36 %
Test accuracy	78.10 %
Training time	122.48 s

Number of support vectors: 4940

### SVM with RBF kernel ( $\gamma=0.005$ )

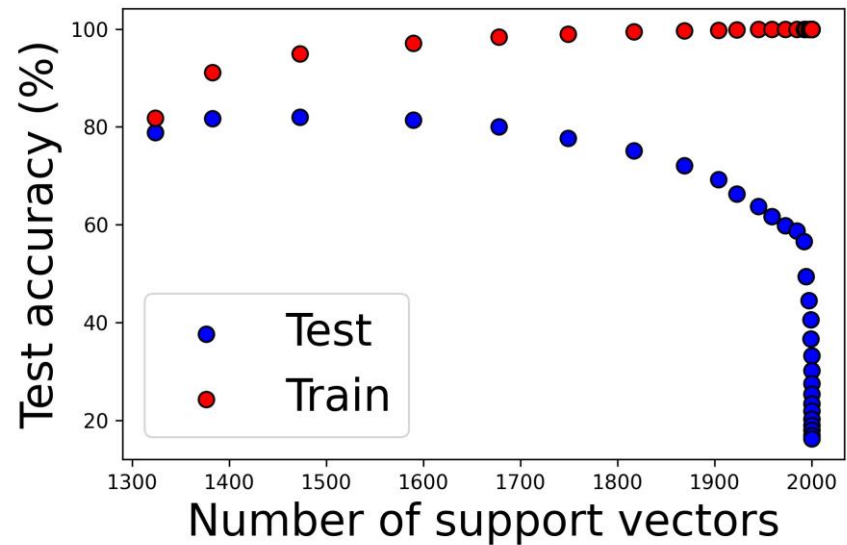
Train accuracy	86.73 %
Test accuracy	85.15 %
Training time	73.37 s

Number of support vectors: 4747



# Support Vector Machines: number of support vectors

MNIST fashion dataset



Training: 2000 samples

Test: 10000 samples

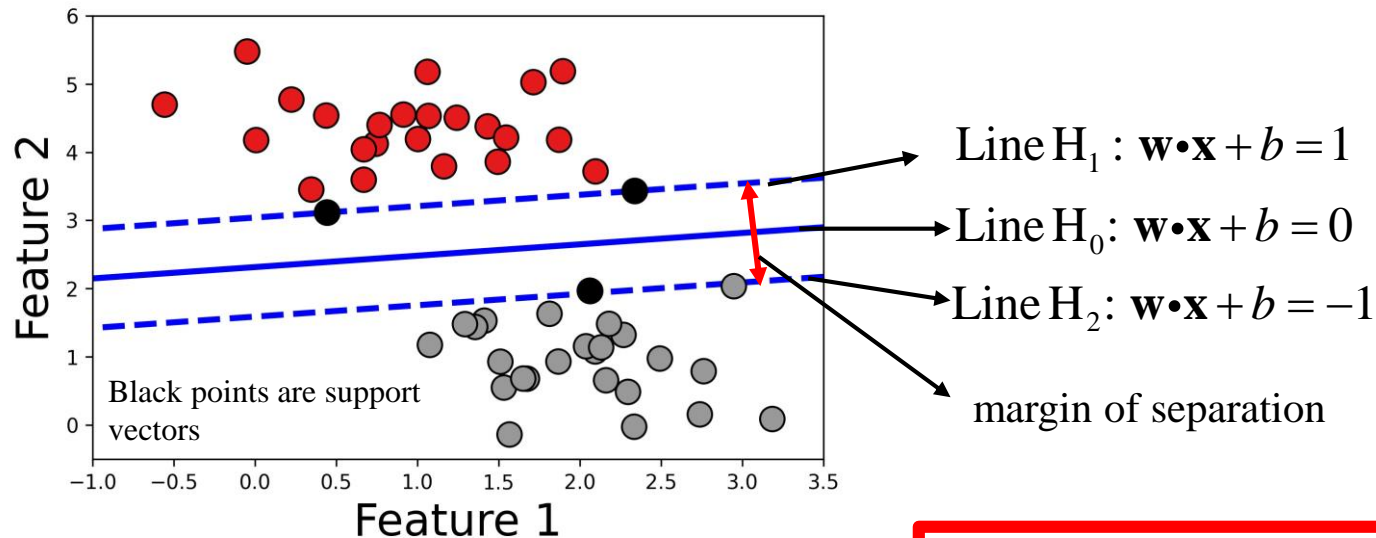
Pixel size: 28x28

Kernel: RBF

Gamma changed from 0.001 to 0.1

$$E[E_{out}] \leq E\left[\frac{\# c}{M-1}\right]$$

# Support Vector Machines: How does it work??



Select  $\mathbf{w}$  to do 2 things:

1. Maximize the margin
2. Minimize the number of samples on the boundaries

SVM finds the points close to the decision boundary in order to draw the decision line.

These points are called support vectors.

The goal is to find a line (hyperplane) with the maximum margin.

## Finding $\mathbf{w}$ with the maximum margin

$\mathbf{x}^{(m1)}$  is the nearest point to the decision plane. We are going to find the distance between  $\mathbf{x}^{(m1)}$  and the plane  $\mathbf{w} \cdot \mathbf{x} + b = 0$ .

For this point:

$$|\mathbf{w} \cdot \mathbf{x}^{(m1)} + b| > 0$$

We normalize the dataset in way that

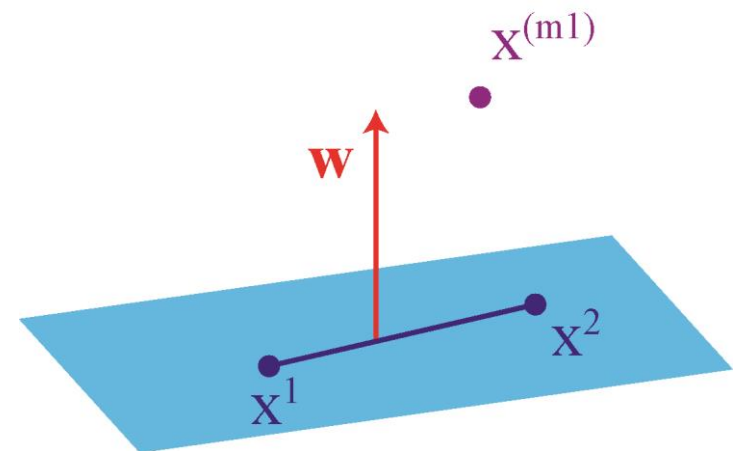
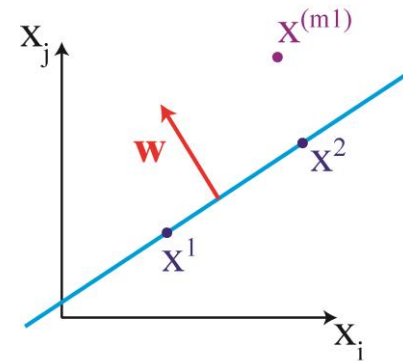
$$|\mathbf{w} \cdot \mathbf{x}^{(m1)} + b| = 1$$

$\mathbf{x}^1$  and  $\mathbf{x}^2$  are points on the decision plane.

$$\mathbf{w} \cdot \mathbf{x}^1 + b = 0 \quad \text{and} \quad \mathbf{w} \cdot \mathbf{x}^2 + b = 0$$

$$\rightarrow \mathbf{w} \cdot (\mathbf{x}^1 - \mathbf{x}^2) = 0$$

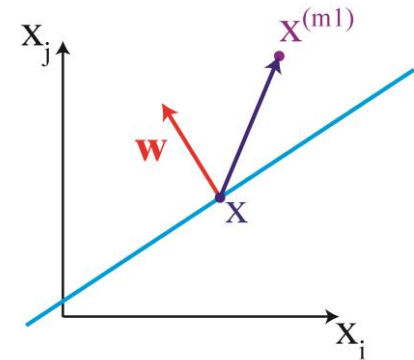
Therefore,  $\mathbf{w}$  is perpendicular to the decision plane.



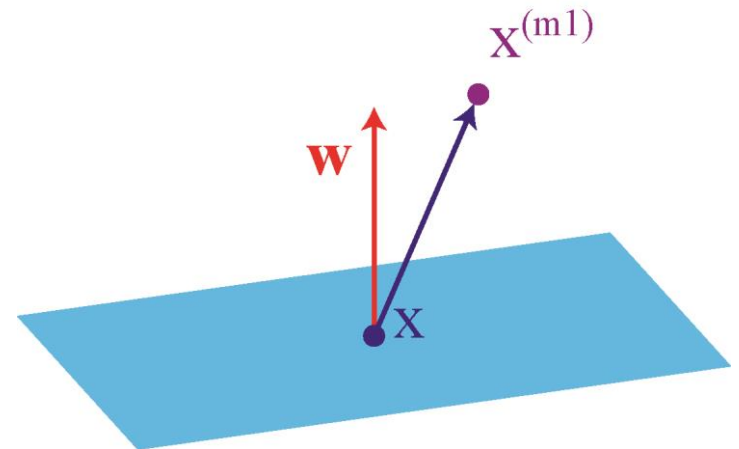
## Finding $\mathbf{w}$ with the maximum margin

The distance between  $\mathbf{x}^{(n)}$  and the decision plane can be calculated by projecting  $(\mathbf{x}^{(n)} - \mathbf{x})$  on  $\mathbf{w}$ .  $\mathbf{x}$  is any point on the decision plane.

$$\mathbf{w} = \frac{\mathbf{w}}{\|\mathbf{w}\|} \rightarrow \text{distance} = \left| \mathbf{w} \cdot (\mathbf{x}^{(m1)} - \mathbf{x}) \right|$$



$$\text{distance} = \frac{|\mathbf{w} \cdot \mathbf{x}^{(m1)} - \mathbf{w} \cdot \mathbf{x}|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \left| \underbrace{\mathbf{w} \cdot \mathbf{x}^{(m1)}}_{=1} + \underbrace{b - \mathbf{w} \cdot \mathbf{x} - b}_{=0} \right| = \frac{1}{\|\mathbf{w}\|}$$



# Support Vector Machines: optimization problem

Select  $\mathbf{w}$  to maximize the distance between the decision plane and the nearest point

$$\text{Maximize } \frac{1}{\|\mathbf{w}\|} \longrightarrow \text{Minimize } \frac{1}{2} \|\mathbf{w}\|^2$$

With the constraint that there are no datapoints in the margin region

$$y^{(m)}(\mathbf{x}^{(m)} \bullet \mathbf{w} + b) \geq 1 \text{ for } m = 1, 2, \dots, M$$

$$\text{Labels: } y^{(m)} = [-1, 1]$$

$\frac{1}{2} \|\mathbf{w}\|^2$  is a quadratic function in  $w_i$ . Therefore, the surface is a paraboloid and it has only one m

# Lagrange multipliers

We use Lagrange multipliers methods to solve this optimization problem.

Optimum points of a function (f) subject to a given constraint (g) is where the function is tangent to the constraint function. This means:

$$\nabla f = c \nabla g$$

c is the Lagrange multiplier. The above equation should be solved along with the constraints to find the minimum or maximum of the function f. The complete set of equations is:

$$\begin{cases} \nabla f(\mathbf{x}) = c \nabla g(\mathbf{x}) \\ g(\mathbf{x}) = 0 \end{cases}$$

We can combine these equations into one equation as Lagrangian L:

$$L(\mathbf{x}, c) = f(\mathbf{x}) - c g(\mathbf{x})$$

# Support Vector Machines: optimization problem

Optimizing the problem using Lagrange multipliers:

$$f = \frac{1}{2} \|\mathbf{w}\|^2$$

$$g^{(m)} = y^{(m)} (\mathbf{x}^{(m)} \cdot \mathbf{w} + b) - 1$$

$$\text{Lagrangian: } L = f - \sum_m c^{(m)} g^{(m)} = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_m c^{(m)} y^{(m)} (\mathbf{x}^{(m)} \cdot \mathbf{w} + b) + \sum_m c^{(m)}$$

$c$  is the Lagrange multiplier. The optimal line can be found by **minimizing  $L$  with respect to  $\mathbf{w}$  and  $b$** . Therefore:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_m c^{(m)} y^{(m)} \mathbf{x}^{(m)} = 0 \rightarrow \mathbf{w} = \sum_m c^{(m)} y^{(m)} \mathbf{x}^{(m)}$$

$$\frac{\partial L}{\partial b} = \sum_m c^{(m)} y^{(m)} = 0$$


Substituting these equations in Lagrangian leads to Lagrangian dual problem where we **maximize  $L$  with respect to  $a$** .

$$L = \sum_m c^{(m)} - \frac{1}{2} \sum_{m'} \sum_m c^{(m)} c^{(m')} y^{(m)} y^{(m')} (\mathbf{x}^{(m)} \cdot \mathbf{x}^{(m')})$$

Select  $\mathbf{w}$  to do 2 things:

1. Maximize the margin
2. Minimize the number of samples on the boundaries

## Support Vector Machines: dot product

$$L = \sum_m c^{(m)} - \frac{1}{2} \sum_{m'} \sum_m c^{(m)} c^{(m')} y^{(m)} y^{(m')} (\mathbf{x}^{(m)} \cdot \mathbf{x}^{(m')})$$


Inner products describe the similarity between patterns

The goal is to maximize  $L$  with respect to  $c$ .  $c$  is a positive number.

- If two patterns ( $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$ ) are completely dissimilar, then their dot product is zero and they do not contribute to  $L$ .
- If two patterns ( $\mathbf{x}^{(1)}$  and  $\mathbf{x}^{(2)}$ ) are alike, then:
  - If they have the same label, then  $c^{(1)} c^{(2)} y^{(1)} y^{(2)} \mathbf{x}^{(1)} \cdot \mathbf{x}^{(2)}$  is positive. This decreases the value of  $L$ . The algorithm does not consider this case since it wants to maximize the value of  $L$ .
  - If they have opposite labels, then  $c^{(1)} c^{(2)} y^{(1)} y^{(2)} \mathbf{x}^{(1)} \cdot \mathbf{x}^{(2)}$  is negative. The value of  $L$  increases in this case. We are looking for these critical patterns since they can be used to distinguish different categories (classes).



# Support Vector Machines: kernel trick

The function that we want to optimize is:

$$L = \sum_m c^{(m)} - \frac{1}{2} \sum_{m'} \sum_m c^{(m)} c^{(m')} y^{(m)} y^{(m')} (\mathbf{x}^{(m)} \bullet \mathbf{x}^{(m')})$$

If we transform the data into a new space with transformation  $Z$ , then the function is:

$$L = \sum_m c^{(m)} - \frac{1}{2} \sum_{m'} \sum_m c^{(m)} c^{(m')} y^{(m)} y^{(m')} (Z(\mathbf{x}^{(m)}) \bullet Z(\mathbf{x}^{(m')}))$$

We need to calculate the dot product of the transformed feature vectors  $(Z(\mathbf{x}^{(m)}) \bullet Z(\mathbf{x}^{(n)}))$ . Calculation of this dot product is computationally expensive.

We define a kernel function  $K$  such that:

$$K(\mathbf{x}^{(m)}, \mathbf{x}^{(m')}) = Z(\mathbf{x}^{(m)}) \bullet Z(\mathbf{x}^{(m')})$$

We do not need to know  $Z$  transformation. This kernel defines the dot product in  $Z$  space.

# Support Vector Machines: kernel trick

The multiplier vector or the weights are:

$$\mathbf{w} = \sum_m^M c^{(m)} y^{(m)} Z(\mathbf{x}^{(m)})$$

How do we calculate the weights if we do not know the  $Z(\mathbf{x}^{(m)})$  transformation?

We do not calculate the weights. We calculate the labels directly:

$$\text{prediction} = \mathbf{w} \bullet Z(\mathbf{x}) + b = \sum_m^M c^{(m)} y^{(m)} Z(\mathbf{x}^{(m)}) \bullet Z(\mathbf{x}) + b$$

$$\rightarrow \text{prediction} = \sum_m^M c^{(m)} y^{(m)} K(\mathbf{x}^{(m)}, \mathbf{x}) + b$$

We can calculate the prediction value using the kernel function.