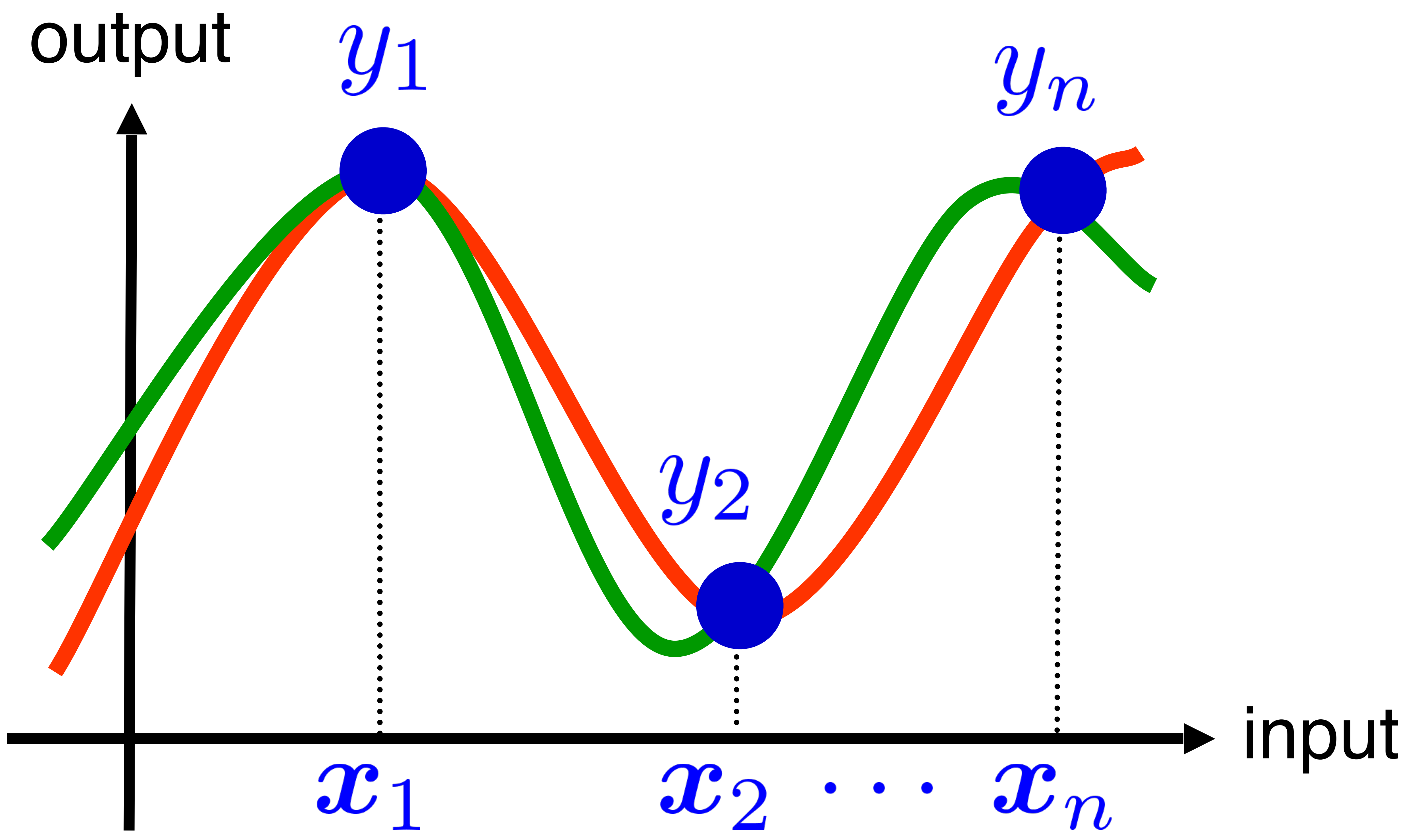
Least Squares Classification

Masashi Sugiyama, Takashi Ishida, Yivan Zhang
{sugi, ishi, yivan.zhang}@k.u-tokyo.ac.jp
http://www.ms.k.u-tokyo.ac.jp

BY NC ND

Review of regression



$f: X \to \mathbb{R}$: true function

$\hat{f}: X \to \mathbb{R}$: learned function

$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$: training samples

$y_i = f\left(\boldsymbol{x}_i\right)$(+ noise)

Aim to find a function from the training sample
that is close to the true function.

Linear-in-parameter model

- Linear model: $$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{b} \theta_j \phi_j(\boldsymbol{x})$$

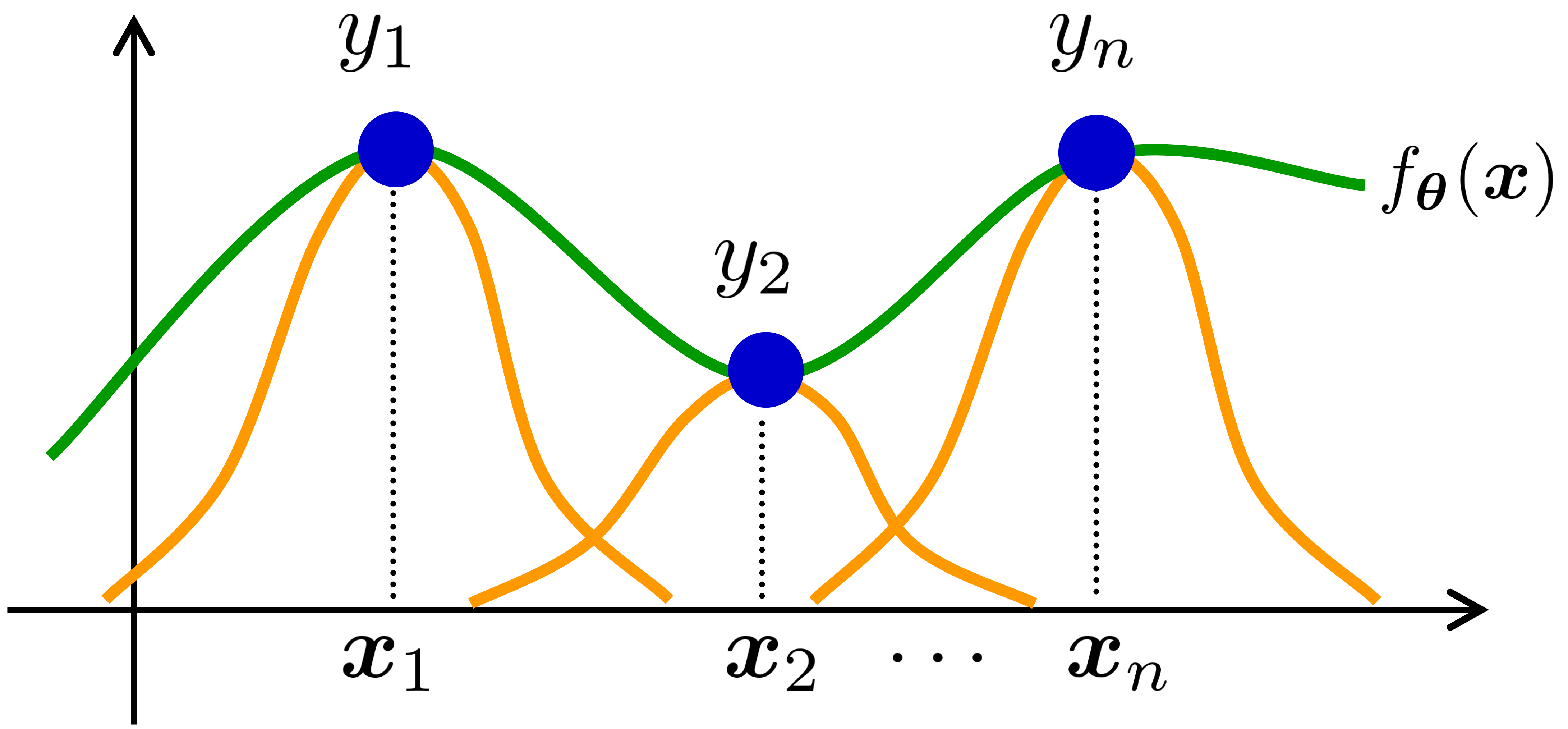
$\{\phi_j(\boldsymbol{x})\}_{j=1}^{b}$: basis functions

- Kernel model:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{n} \theta_j K(\boldsymbol{x}, \boldsymbol{x}_j)$$

Gaussian kernel:

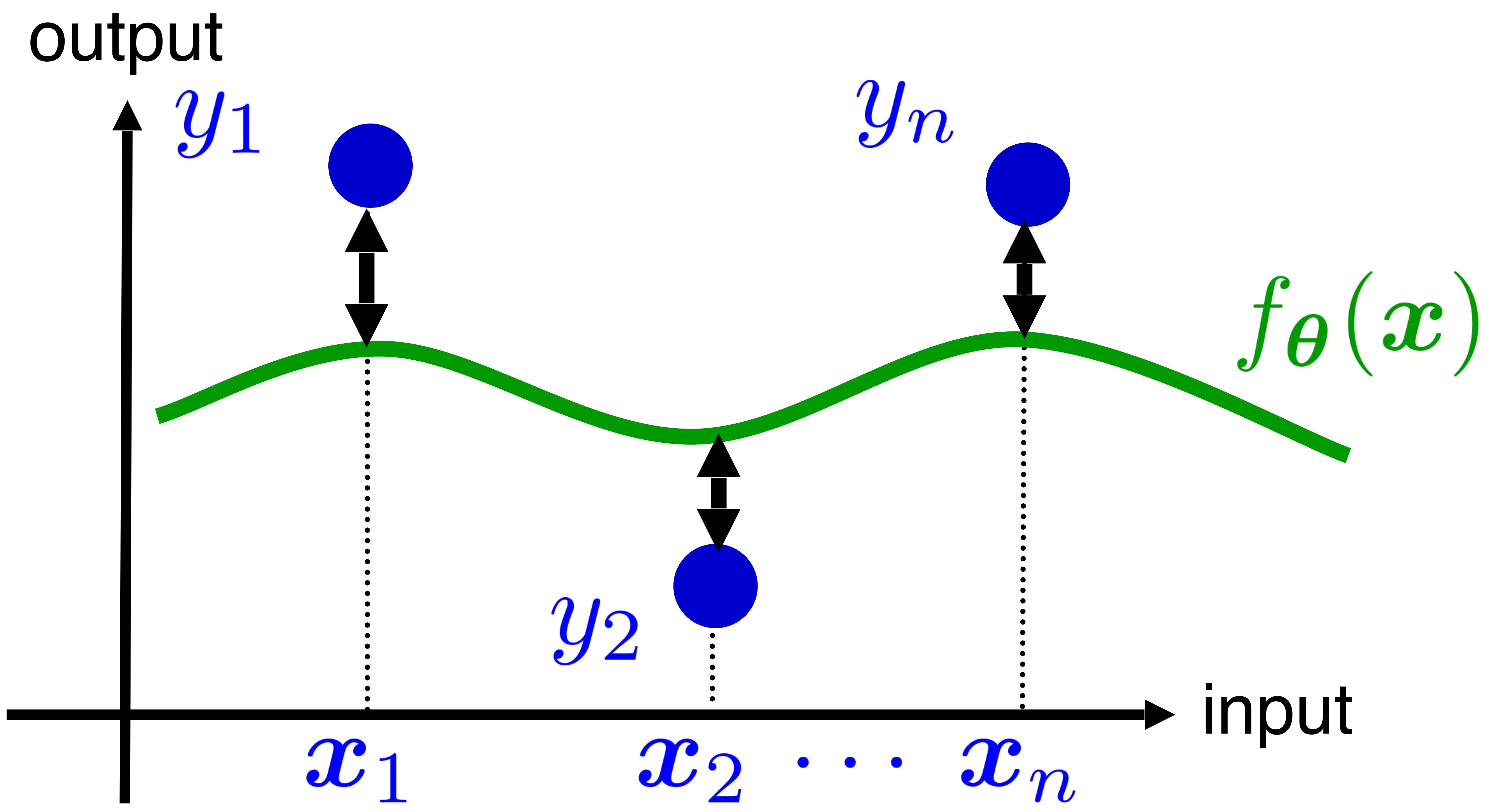
$$K(\boldsymbol{x}, \boldsymbol{c}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}\|^2}{2h^2}\right)$$



Least squares regression

- Minimize the squared error between model output and label:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i \right)^2$$

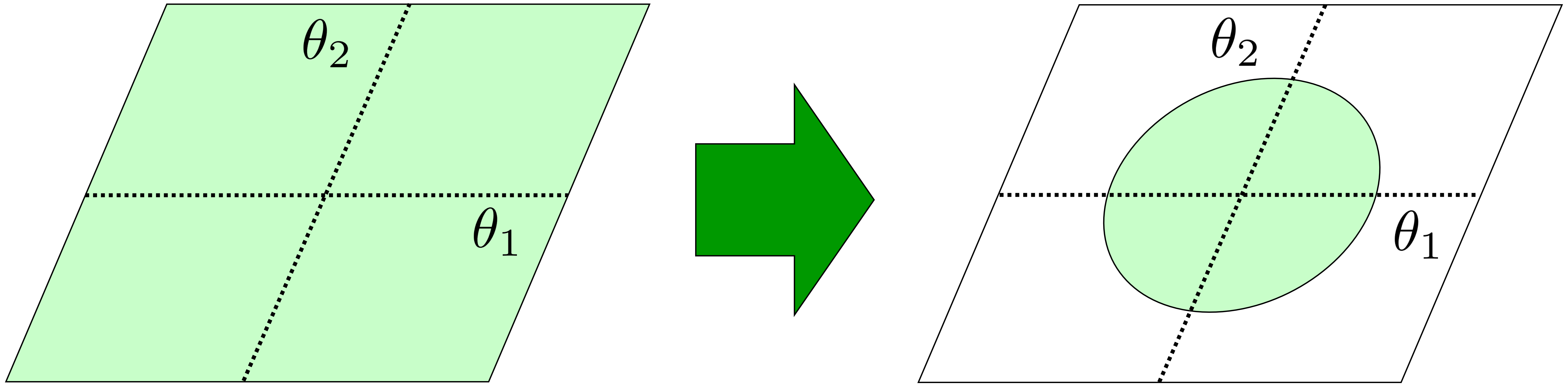


ℓ_2-constrained least squares regression

- Restrict the model to be within a hyper-cube in order to alleviate overfitting.

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i \right)^2 \quad \text{subject to } \|\boldsymbol{\theta}\|^2 \leq R$$

$$R \geq 0$$



ordinary least squares

ℓ_2-constrained least squares

Combining different losses and constraints

- Learning methods for linear and kernel models:

Loss function \ Constraint		None	ℓ_2 Regularization	ℓ_1 Regularization & sparse
ℓ_2-loss	Efficient	Analytical solution	Analytical solution	Quadratic programming
Huber loss	↕	Quadratic programming	Quadratic programming	Quadratic programming
ℓ_1-loss	Robust	Linear programming	Quadratic programming	Linear programming

- Choose the model and regularization parameters with cross-validation.

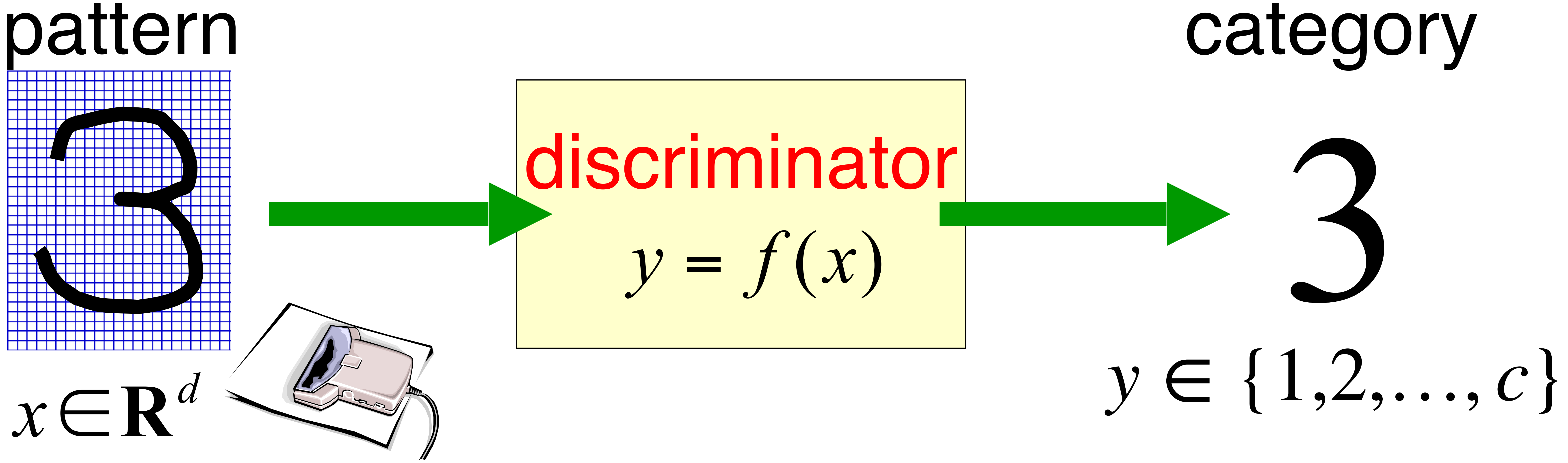
Components of a regression model

- **Model** $f : X \to \mathbb{R}$: linear model, kernel model, neural network, ...
- **Training examples**: how data is collected
- **Evaluation criteria**: how good a model is (mean absolute error (MAE), mean squared error (MSE), maximum error, ...)
- **Learning objective**: how to learn a good model
- **Optimization**: exact solution, heuristics, gradient-based optimization, derivative-free/black-box optimization

Contents

Pattern recognition

- The problem of constructing a discriminant function that assigns input patterns to categories
- Humans can design discriminant functions for each problem
- Alternatively: automatically learn discriminant functions from data



Components of a classification model

- **Number of classes** c
- **Model** $f: X \rightarrow \{1,2,\ldots,c\}$: linear model, kernel model, decision tree, neural network, …
- **Training examples**: how data is collected
- **Evaluation criteria**: how good a model is (accuracy, precision, recall, …)
- **Learning objective**: how to learn a good model
- **Optimization**: exact solution, heuristics, gradient-based optimization, derivative-free/black-box optimization

Classification with 2 classes

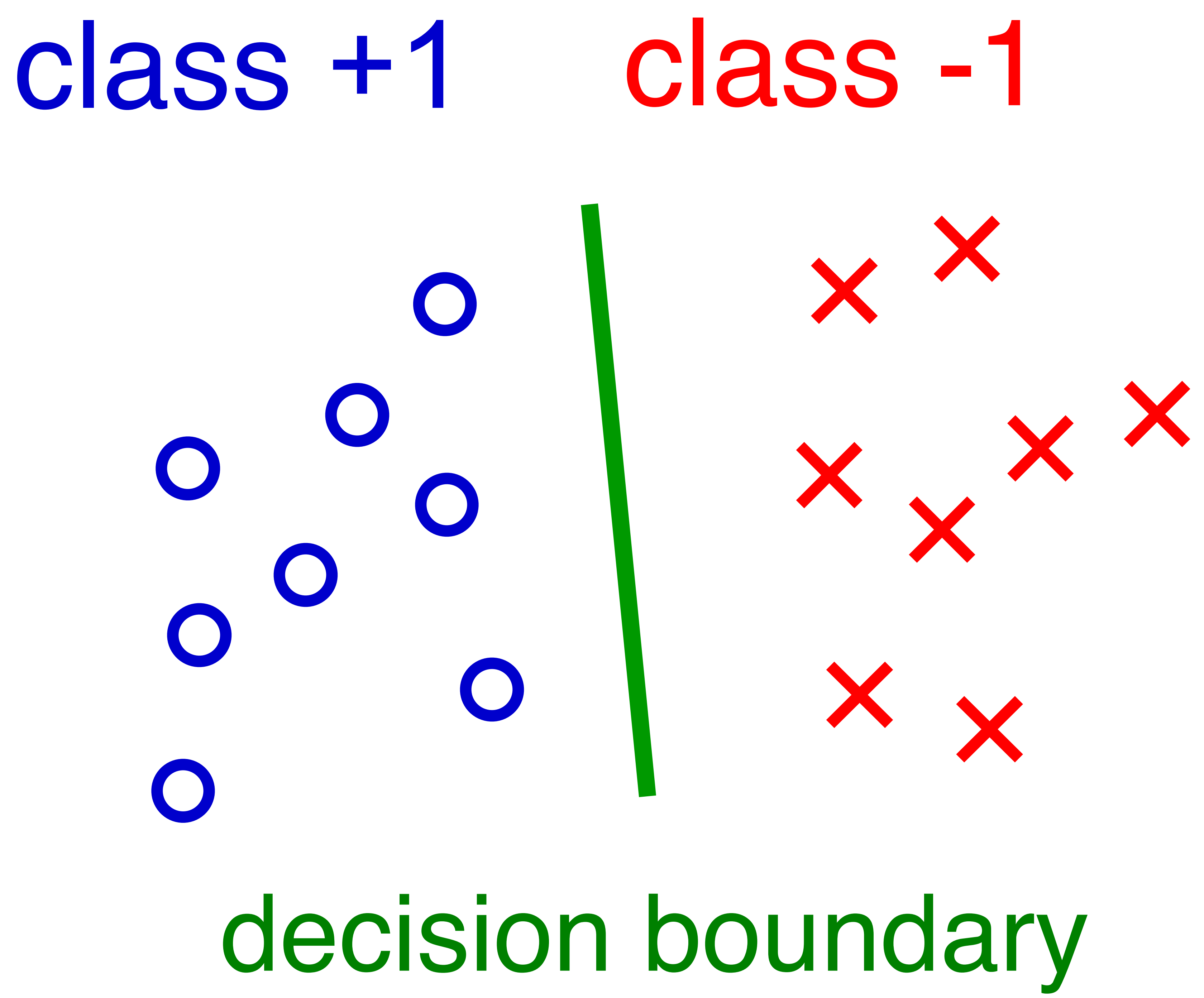
- Labeled training data: $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$
  - d-dimensional input:

$$\boldsymbol{x} \in \mathbb{R}^d$$

  - Binary output class label:

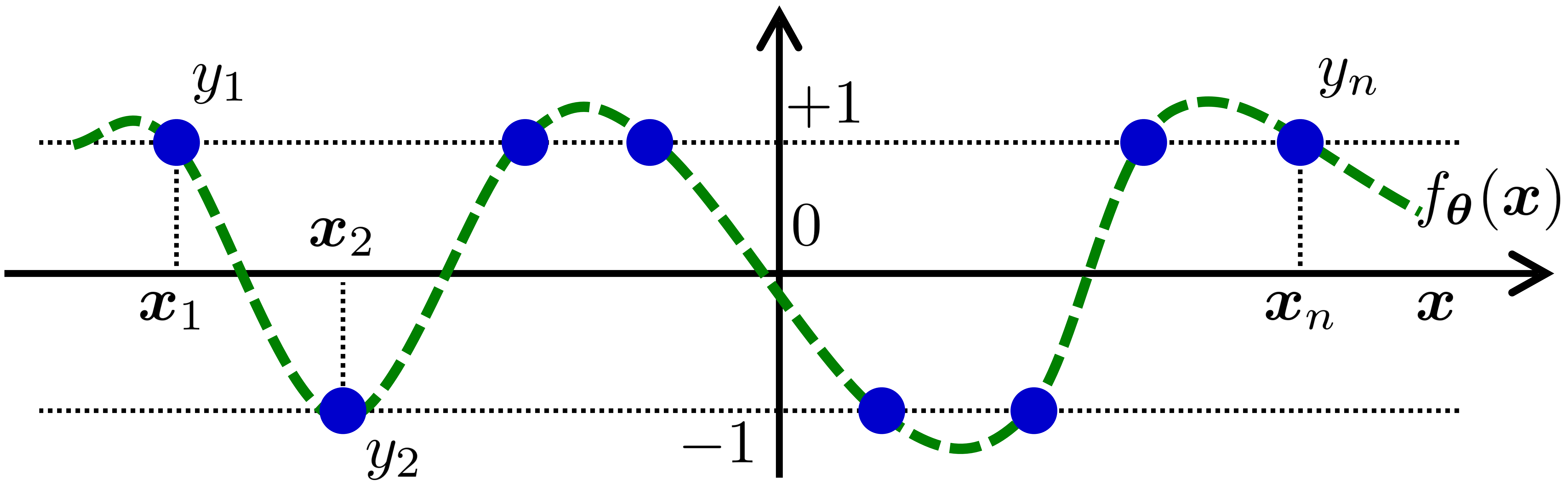
$$y \in \{+1, -1\}$$

- We want to derive the decision boundary between the two classes.



Classification with 2 classes

- The two-class classification problem can be expressed as an approximation for a binary function.



- We can use the methods we learned in regression!

Classification by regression

- Learn parameters with regularized least squares:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum_{i=1}^{n} \Big(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i \Big)^2 + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2 \right]$$

$\lambda(\geq 0)$: regularization hyper-parameter

- Classifying test patterns (samples):

$$\widehat{y} = \operatorname{sign}\left(f_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{x}) \right) = \begin{cases} +1 & (f_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{x}) > 0) \\ 0 & (f_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{x}) = 0) \\ -1 & (f_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{x}) < 0) \end{cases}$$

Example of implementation

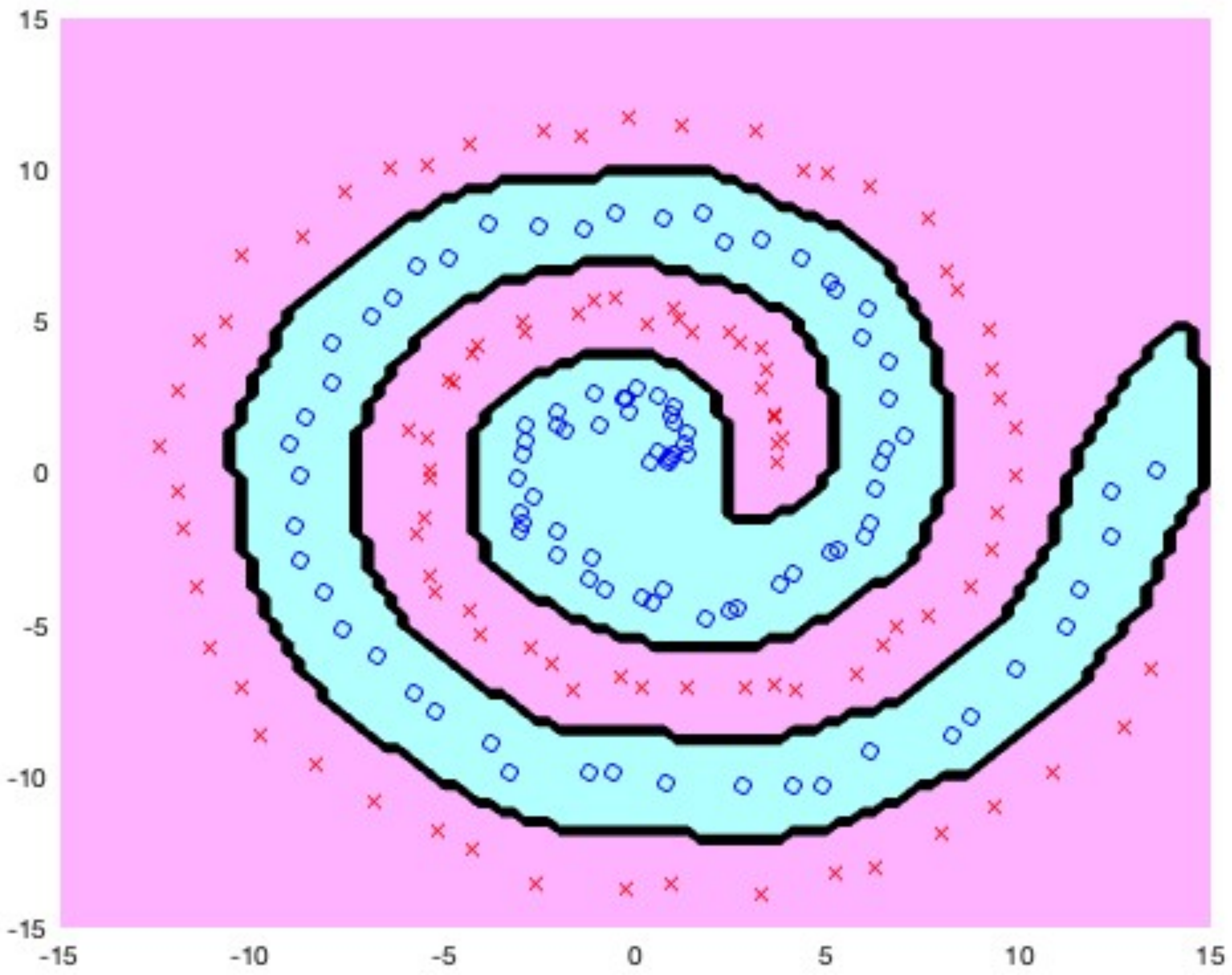
- For the Gaussian kernel model, let's try implementing a classification algorithm based on regularized least squares regression.

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{n} \theta_j K(\boldsymbol{x}, \boldsymbol{x}_j)$$

$$K(\boldsymbol{x}, \boldsymbol{c}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}\|^2}{2h^2}\right)$$

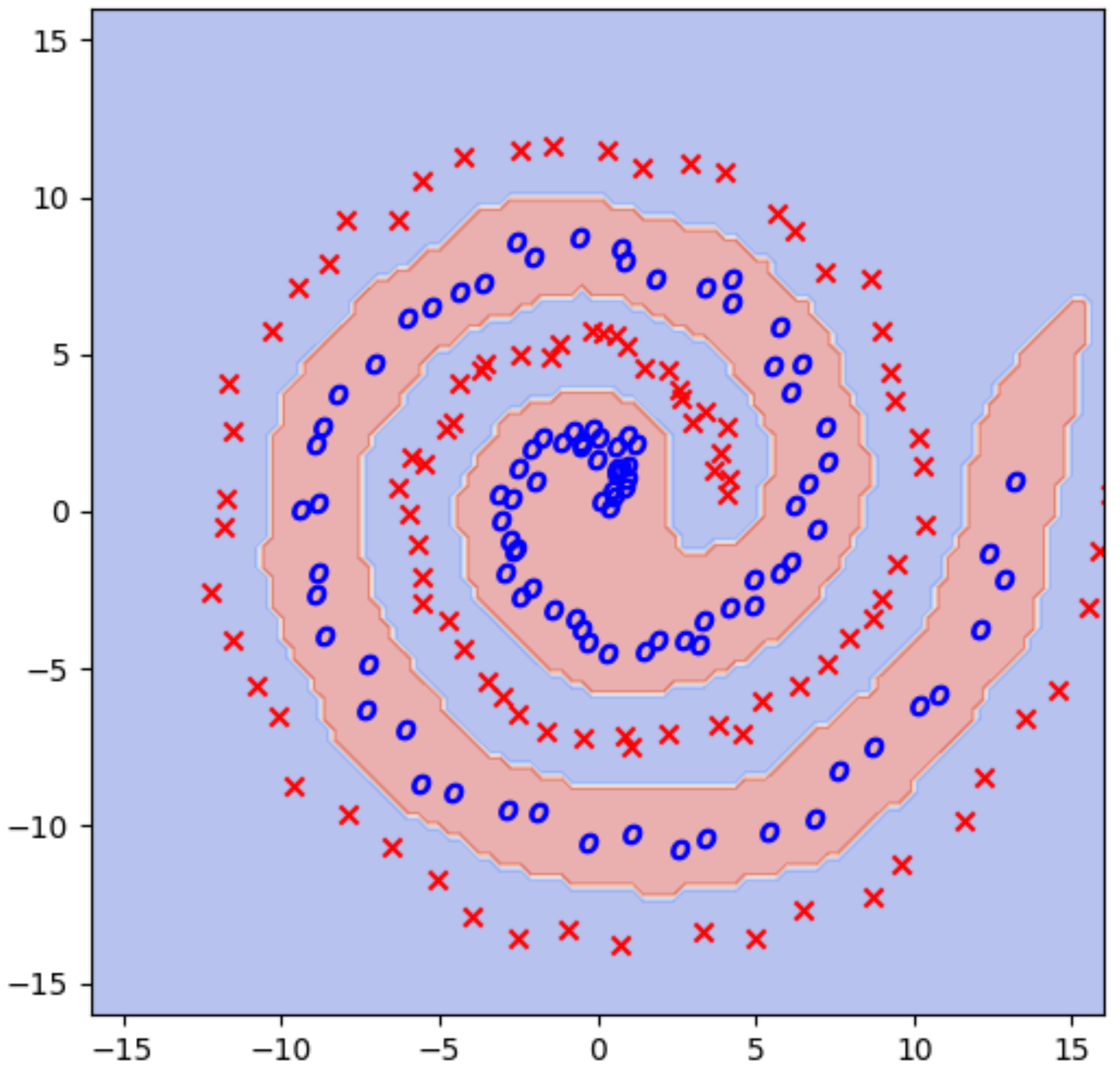
MATLAB code

```
clear all; rand('state',0); randn('state',0);
n=200; a=linspace(0,4*pi,n/2);
u=[a.*cos(a) (a+pi).*cos(a)]'+1*rand(n,1);
v=[a.*sin(a) (a+pi).*sin(a)]'+1*rand(n,1);
x=[u v]; y=[ones(1,n/2) -ones(1,n/2)]';
x2=sum(x.^2,2); hh=2*1^2; l=0.01;
k=exp(-(repmat(x2,1,n)+repmat(x2',n,1)-2*x*x')/hh);
t=(k^2+l*eye(n))\(k*y);

m=100; X=linspace(-15,15,m)'; X2=X.^2;
U=exp(-(repmat(u.^2,1,m)+repmat(X2',n,1)-2*u*X')/hh);
V=exp(-(repmat(v.^2,1,m)+repmat(X2',n,1)-2*v*X')/hh);
figure(1); clf; hold on; axis([-15 15 -15 15]);
contourf(X,X,sign(V'*(U.*repmat(t,1,m))));
plot(x(y==1,1),x(y==1,2),'bo');
plot(x(y==-1,1),x(y==-1,2),'rx');
colormap([1 0.7 1; 0.7 1 1]);
```



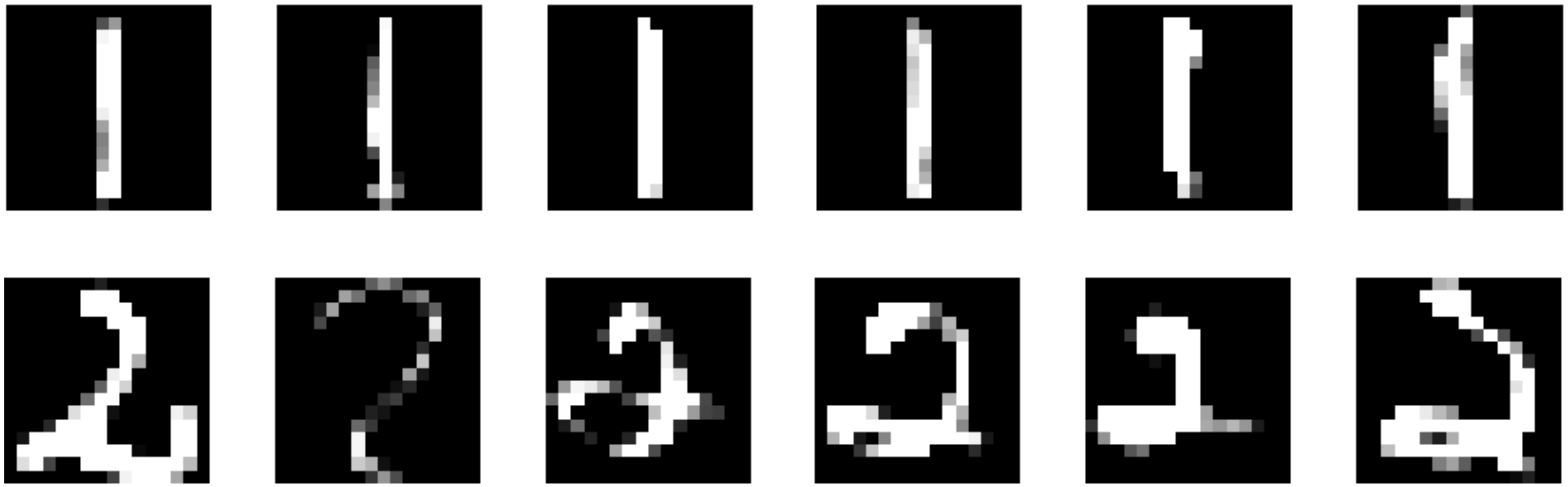
Python code


```
import numpy as np; import matplotlib; matplotlib.use('TkAgg'); import matplotlib.pyplot as plt; np.random.seed(1)

def generate_data(sample_size):
    a = np.linspace(0, 4 * np.pi, num=sample_size // 2)
    x = np.concatenate(
        [np.stack([a * np.cos(a), a * np.sin(a)], axis=1),
         np.stack([(a + np.pi) * np.cos(a), (a + np.pi) * np.sin(a)], axis=1)])
    x += np.random.random(size=x.shape)
    y = np.concatenate([np.ones(sample_size // 2), -np.ones(sample_size // 2)])
    return x, y

def build_design_mat(x1, x2, bandwidth):
    return np.exp(
        -np.sum((x1[:, None] - x2[None]) ** 2, axis=-1) / (2 * bandwidth ** 2))

def optimize_param(design_mat, y, regularizer):
    return np.linalg.solve(
        design_mat.T.dot(design_mat) + regularizer * np.identity(len(y)),
        design_mat.T.dot(y))

def visualize(theta, x, y, grid_size=100, x_min=-16, x_max=16):
    grid = np.linspace(x_min, x_max, grid_size)
    X, Y = np.meshgrid(grid, grid)
    mesh_grid = np.stack([np.ravel(X), np.ravel(Y)], axis=1)
    design_mat = build_design_mat(x, mesh_grid, bandwidth=1.)
    plt.clf()
    plt.figure(figsize=(6, 6))
    plt.xlim(x_min, x_max)
    plt.ylim(x_min, x_max)
    plt.contourf(X, Y, np.reshape(np.sign(design_mat.T.dot(theta)), (grid_size, grid_size)), alpha=.4, cmap=plt.cm.coolwarm)
    plt.scatter(x[y == 1][:, 0], x[y == 1][:, 1], marker='$O$', c='blue')
    plt.scatter(x[y == -1][:, 0], x[y == -1][:, 1], marker='x', c='red')
    plt.savefig('l5-p15.png')

x, y = generate_data(sample_size=200)
design_mat = build_design_mat(x, x, bandwidth=1.)
theta = optimize_param(design_mat, y, regularizer=0.01)
visualize(theta, x, y)
```



Handwritten digit recognition by Gaussian kernel least-squares classification

- 16 x 16 pixels, density of each pixel from 0 to 255
- Training sample: 1,000 characters, 500 for each of 1 and 2
- Test sample: 400 characters, 200 each of 1 and 2



Importing Handwritten Digits

- Download the file from:

  http://www.ms.k.u-tokyo.ac.jp/sugi/software/SML.zip

- Load the handwritten digits by:

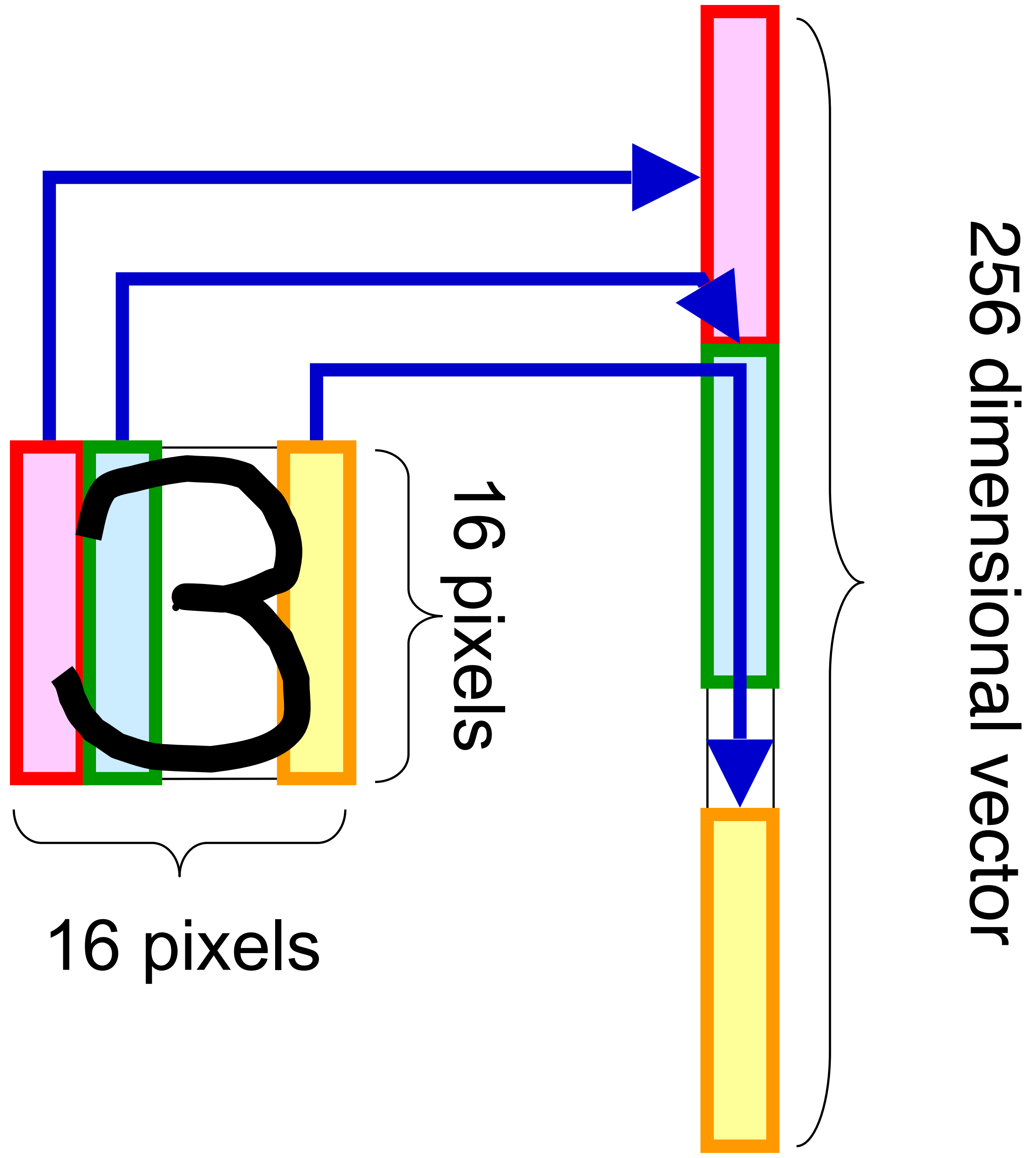
```
load
digit.mat
```

- Then, we will have X (training) and T (test).
- Use the `whos` command:

```
> whos
  Name          Size             Bytes  Class
  T             256x200x10     4096000  double
  X             256x500x10    10240000  double
```

Details of the handwritten digits

- X and T are 3-dimensional arrays
- A single handwritten digit is a 256-dimensional vector.
- It is a vector of 16x16 pixel image data, each element of which is a real number between -1 and 1.



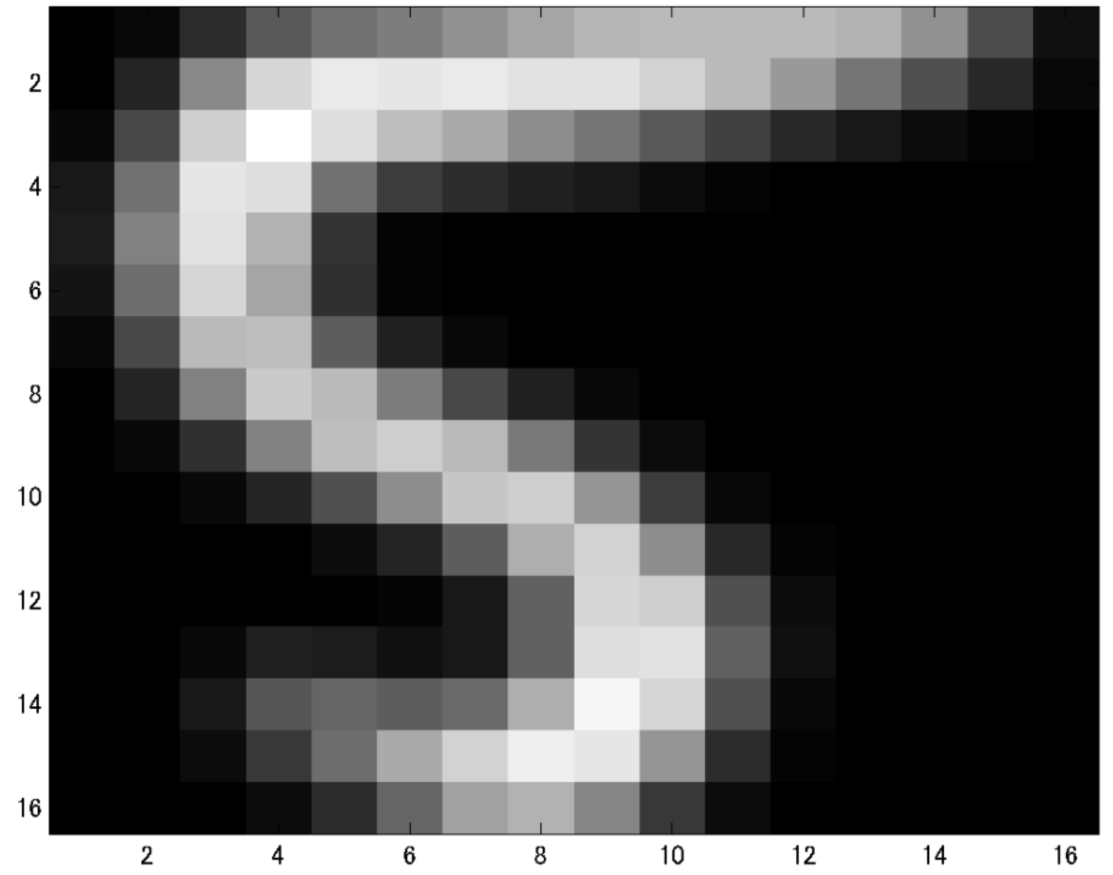
Details of the handwritten digits

- When the value is -1: pixel is black
- When the value is 1: pixel is white
- X: 500 characters for each number from 0 to 9
- T: 200 characters for each number from 0 to 9
- For example, to extract the data of the 23rd training handwritten digit 5 into the variable x, use x=X(:,23,5);.
- Note that the data for handwritten digit 0 corresponds to the case where the third argument is 10.

Visualizing the chosen data

- Images of the retrieved handwritten text data can be displayed as follows:

```
imagesc(reshape(x,[16 16])')
colormap(gray)
```



Example


```
clear all; rand('state',0); randn('state',0);
load digit.mat
x=[X(:,:,1) X(:,:,2)]; y=[ones(500,1); -ones(500,1)];
n=length(y); x2=sum(x.^2,1); hh=2*10^2; l=1;
k=exp(-(repmat(x2,n,1)+repmat(x2',1,n)-2*x'*x)/hh);
t=(k^2+l*eye(n))\(k*y);

u=T(:,:,1); % Test patterns 1
v=exp(-
(repmat(x2,200,1)+repmat(sum(u.^2,1)',1,n)-2*u'*x)/hh)*t;
C(1,1)=sum(sign(v)>=0); C(1,2)=sum(sign(v)<0);
u=T(:,:,2); % Test patterns 2
v=exp(-
(repmat(x2,200,1)+repmat(sum(u.^2,1)',1,n)-2*u'*x)/hh)*t;
C(2,1)=sum(sign(v)>=0); C(2,2)=sum(sign(v)<0);

C
```

Results

- Accuracy is: 3 9 9 / 4 0 0 = 9 9 . 7 5 %

Predicted category

True category

	1	2
1	199	1
2	0	200

Contents

1. Classification by least squares regression
2. Multi-class classification problems
3. Fisher discriminant analysis
4. 0/1-loss and the margin

Multiclass classification

- Classification with 3 or more classes
  - Digit classification: 10 classes
  - Alphabet classification: 26 classes
  - Kanji classification: around 3k classes
- Least squares classification was designed for binary classification 👉 cannot be used for multiclass classification directly.
- Idea: decompose multiclass classification into several binary classification problems.
  - One-versus-all(other), one-versus-one

One-versus-other (all)

- Consider binary classification:
  - one particular class vs. all other class
- Classify into the class that gives the highest score

+1		-1	
class 1	vs.	classes other than 1	$\widehat{f}_1$
class 2	vs.	classes other than 2	$\widehat{f}_2$
	$\vdots$		$\vdots$
class c	vs.	classes other than c	$\widehat{f}_c$

$$\widehat{y} = \underset{y=1,\ldots,c}{\operatorname{argmax}}\ \widehat{f}_y(\boldsymbol{x})$$

One-versus-one

- Decompose into binary problems of one class and another class
- Classify samples into the class with the most votes

$$\mathrm{sign}\left(\widehat{f}_{y,y'}(\boldsymbol{x})\right) = \begin{cases} +1 & \text{vote for class } y \\ 0 & \text{no vote} \\ -1 & \text{vote for class } y' \end{cases}$$

	Class 1	Class 2	Class 3	$\cdots$	Class c
Class 1		$\widehat{f}_{1,2}$	$\widehat{f}_{1,3}$	$\cdots$	$\widehat{f}_{1,c}$
Class 2			$\widehat{f}_{2,3}$	$\cdots$	$\widehat{f}_{2,c}$
Class 3				$\cdots$	$\widehat{f}_{3,c}$
$\vdots$					$\vdots$
Class c					

Summary of multiclass classification

- Solving multi-class classification problems directly
  - May not be possible depending on the method used
- One-vs-all method:
  - We can break down a multi-class problem with "c" classes into "c" binary classification problems.
  - Each binary problem involves a single class against all other classes. All training samples are included in one of the binary problems, which may result in imbalanced training sample sizes for the two classes.
- One-vs-one method:
  - We break down the multi-class problem into c(c+1)/2 binary classification problems, where each problem involves a pair of classes. This results in fewer training samples per binary problem compared to the one-vs-all method. However, there may be subjectivity in the voting mechanism to determine the final class.
- Which method is better depends on the specific situation and dataset being used.

Contents

Statistical pattern recognition

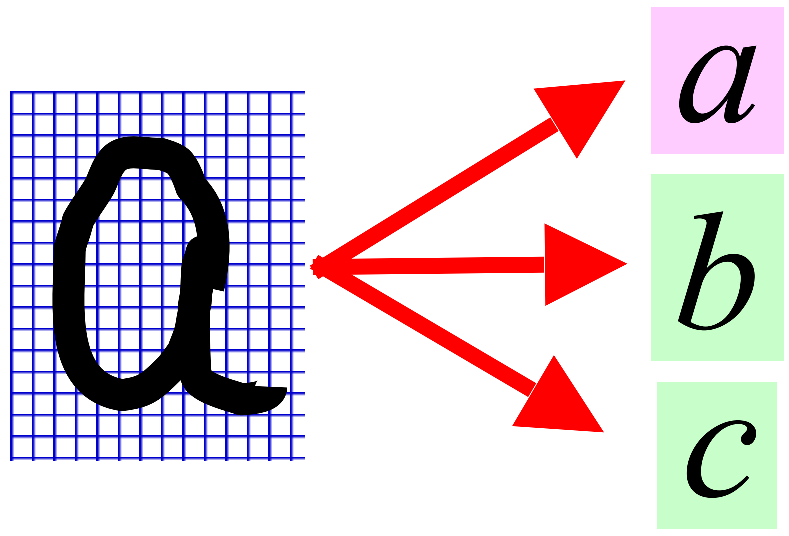
- Training samples: the patterns with category information $\{(x_i, y_i)\}_{i=1}^{n}$
  - inputs are $\boldsymbol{x}_i \in \mathbb{R}^d$, outputs are $y_i \in \{1,2,\ldots,c\}$.
- Statistical pattern recognition: learn discriminative function by using the statistical properties of the training samples
- Assumption:

$$(x_i, y_i) \overset{\text{iid}}{\sim} p(x, y)$$

  - i.i.d.: independent and identically distributed

An ideal case of pattern recognition

- Class-posterior probability $p(y\,|\,\boldsymbol{x})$:
  - The probability that $\boldsymbol{x}$ belongs to class y.



- Classifying patterns into categories that maximize the posterior probability minimizes the rate of pattern misidentification:

$$f(x) = \arg\max_{y} p(y \mid x)$$

- In practice, the posterior probabilities are unknown and must be estimated from the training sample.

Classification based on generative models

- Classify to maximize class posterior probability: $\text{argmax}_y \; p(y \mid x)$
- With the Bayes theorem, discriminative models to be expressed in terms of generative models:

$$\underbrace{p(y|x)}_{\text{discriminative model}} = \frac{p(x,y)}{p(x)} \propto \underbrace{p(x,y)}_{\text{generative model}} = \underbrace{p(x|y)}_{\text{conditional probability}}\underbrace{p(y)}_{\text{class prior}}$$

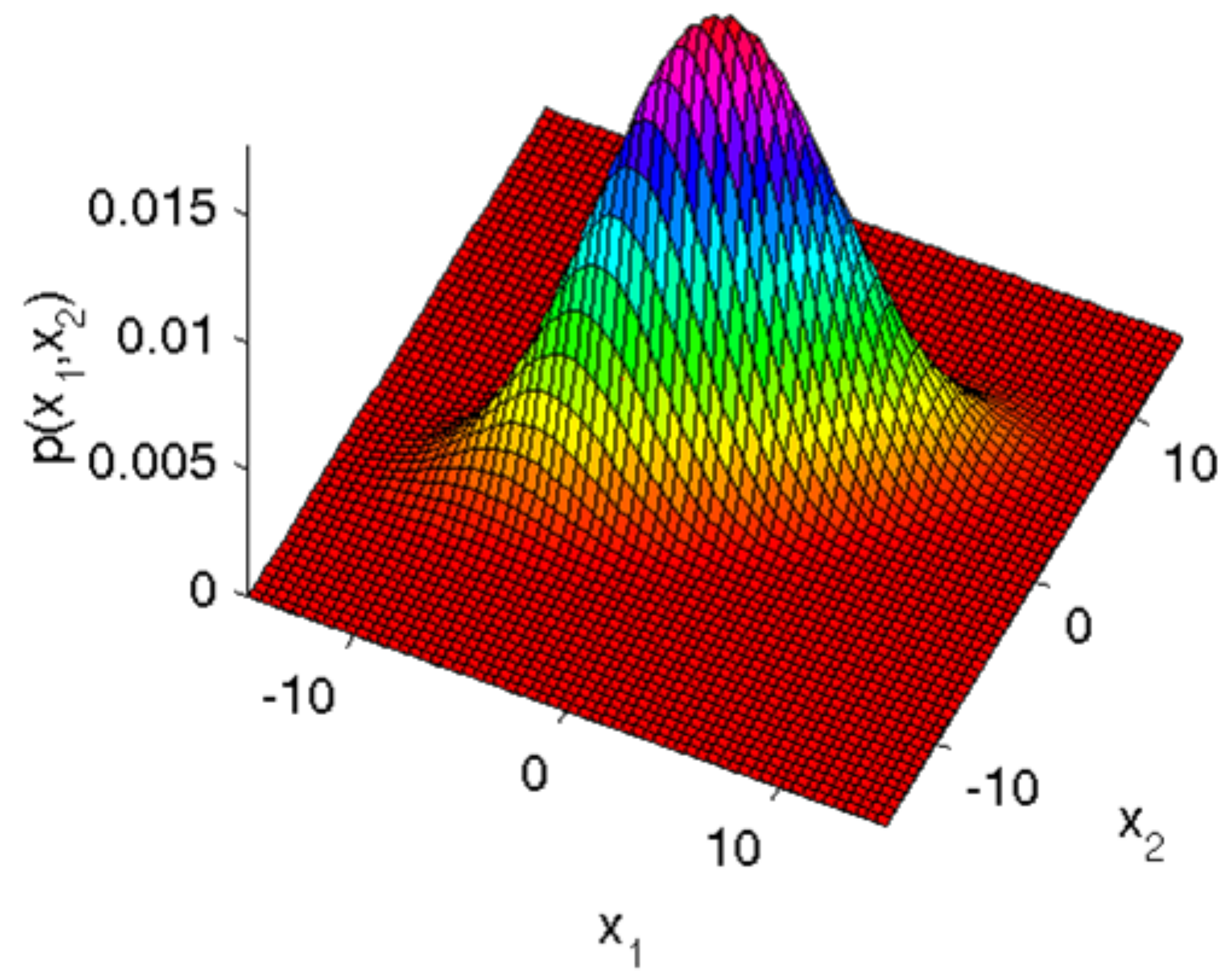
- Taking the log:

$$\log p(y \mid x) = \log p(x \mid y) + \log p(y) + C$$

$C = -\log p(x)$:constant

Estimation of prior and conditional probabilities

- Class prior $p(y)$: estimate with the proportion of training samples that belong to class y.
  - $\hat{p}(y) = \frac{n_y}{n}$, where n_y is the number of samples that belong to class y.
- Conditional probability $p(\boldsymbol{x} \mid y)$:
  - Assume a Gaussian model
  - Maximum likelihood of the expectation and variance-covariance matrix.



$$q(x;\mu_y,\Sigma_y) = \frac{1}{(2\pi)^{d/2}\det(\Sigma_y)^{1/2}}\exp\left(-\frac{1}{2}(x-\mu_y)^T\Sigma_y^{-1}(x-\mu_y)\right)$$

Maximum likelihood estimation

- Maximum likelihood estimation: determine parameter values so that the training sample at hand is most likely to occur
- The probability that training samples $\{x_i\}_{i=1}^{n}$ is sampled from $q(\boldsymbol{x};\theta)$:

$$p\left(x_1, x_2, \ldots, x_n\right) = \prod_{i=1}^{n} q\left(x_i;\theta\right)$$

- Likelihood: the function when we see this as a function of θ

$$L(\theta) = \prod_{i=1}^{n} q\left(x_i;\theta\right)$$

Maximum likelihood estimation (continued)

- Determine parameter values to maximize likelihood

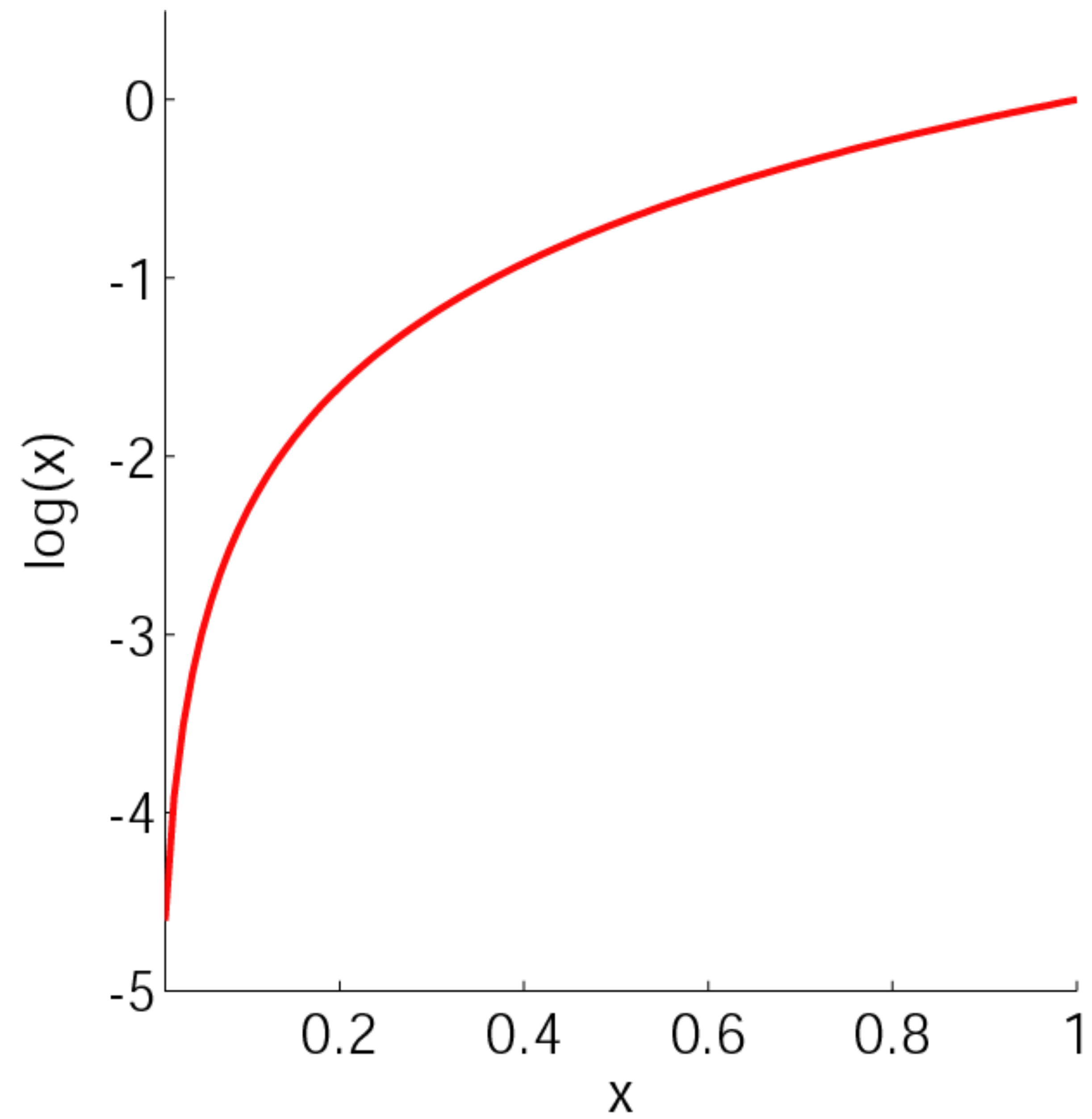
$$\hat{\theta}_{ML} = \underset{\theta \in \Theta}{\arg\max}\, L(\theta)$$

$$L(\theta) = \prod_{i=1}^{n} q(x_i;\theta)$$

- It is often convenient to use log-likelihood:

$$\hat{\theta}_{ML} = \underset{\theta \in \Theta}{\arg\max}\, \log L(\theta)$$

$$\log L(\theta) = \sum_{i=1}^{n} \log q(x_i;\theta)$$



Math exercise

- Gaussian model: $$q(x;\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}\det(\Sigma)^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

- Prove that the maximum likelihood estimators with training samples $\{x_i\}_{i=1}^n$ are:

$$\hat{\mu}_{ML} = \frac{1}{n}\sum_{i=1}^n x_i \qquad \hat{\Sigma}_{ML} = \frac{1}{n}\sum_{i=1}^n (x_i - \hat{\mu}_{ML})(x_i - \hat{\mu}_{ML})^T$$

- Helpful info: you may use the below.

$$\frac{\partial}{\partial\mu}\mu^T\Sigma\mu = 2\Sigma\mu \qquad \frac{\partial}{\partial\mu}\mu^T\Sigma x = \Sigma x \qquad \frac{\partial}{\partial\Sigma}x^T\Sigma^{-1}x = -\Sigma^{-1}xx^T\Sigma^{-1} \qquad \frac{\partial}{\partial\Sigma}\log\det(\Sigma) = \Sigma^{-1}$$

Solution

Log-likelihood:

$$\log L(\mu,\Sigma) = \sum_{i=1}^{n}\left(-\frac{d}{2}\log 2\pi - \frac{1}{2}\log \det(\Sigma) - \frac{1}{2}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)\right)$$

$$= -\frac{nd}{2}\log 2\pi - \frac{n}{2}\log \det(\Sigma) - \frac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)$$

Likelihood equation:

$$\frac{\partial}{\partial\mu}\log L(\mu,\Sigma) = -n\Sigma^{-1}\mu + \Sigma^{-1}\sum_{i=1}^{n}x_i = 0$$

$$\frac{\partial}{\partial\Sigma}\log L(\mu,\Sigma) = -\frac{n}{2}\Sigma^{-1} + \frac{1}{2}\sum_{i=1}^{n}\Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\Sigma^{-1} = 0$$

Solve the above and we are done.

Estimation of prior and conditional probabilities

- So far, we only considered $\{x_i\}_{i=1}^{n}$ without class labels. However, we are interested in applying maximum likelihood for each class because we are interested in $p(x \mid y)$ and we assume it is Gaussian.
- Maximum likelihood estimator of the expected value vector of the class y:

$$\hat{\mu}_y = \frac{1}{n_y} \sum_{i:y_i=y} x_i$$

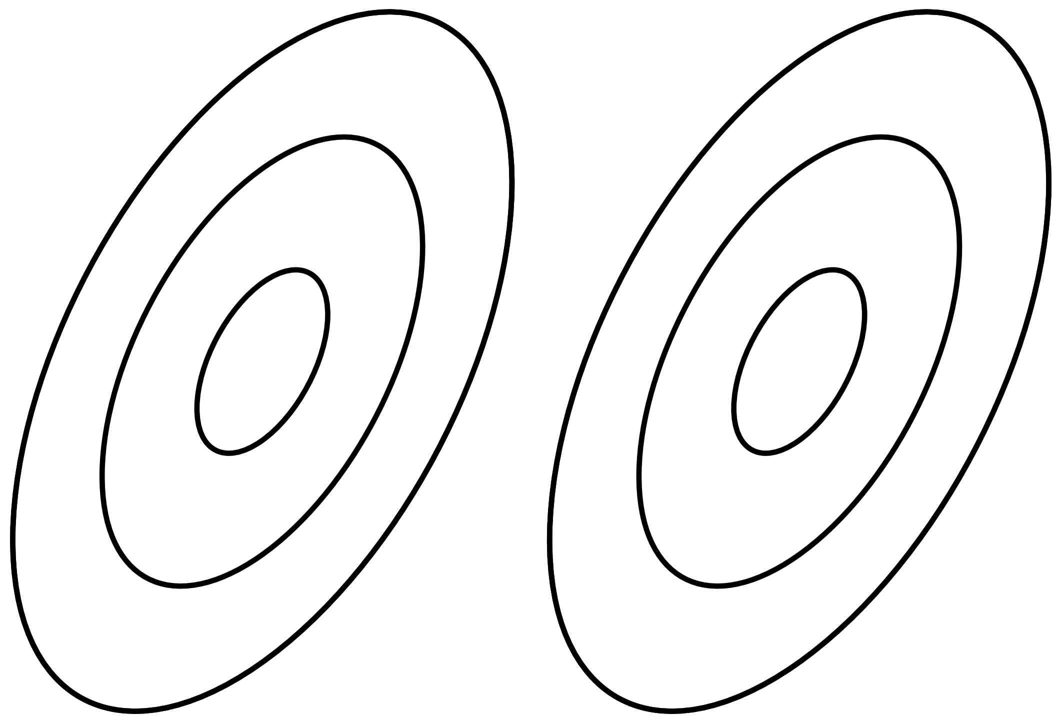
$\sum_{i:y_i=y}$ is the sum for is that satisfies $y_i = y$.

- Assuming that the covariance matrices of each class are equal, the maximum likelihood estimator of the common covariance matrix Σ is:

$$\widehat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{y=-1,+1} \sum_{i:y_i=y} \left(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_y\right)\left(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_y\right)^{\top}$$

$$= \frac{1}{n}\left(n_+ \widehat{\boldsymbol{\Sigma}}_+ + n_- \widehat{\boldsymbol{\Sigma}}_-\right)$$

$\Sigma_+ = \Sigma_- = \Sigma$



Fisher discriminant analysis

$$\log p(y \mid x) = \log p(x \mid y) + \log p(y) + C$$

- Log posterior probability:

$$\log \hat{p}(y \mid x) = \underbrace{-\frac{1}{2} x^T \hat{\Sigma}^{-1} x + \hat{\mu}_y^T \hat{\Sigma}^{-1} x - \frac{1}{2} \hat{\mu}_y^T \hat{\Sigma}^{-1} \hat{\mu}_y - \frac{1}{2} \log\det(\hat{\Sigma})}_{\log p(x|y)} + \underbrace{\log n_y}_{\log p(y)} + C'$$

$$= \hat{\mu}_y^T \hat{\Sigma}^{-1} x - \frac{1}{2} \hat{\mu}_y^T \hat{\Sigma}^{-1} \hat{\mu}_y + \log n_y + C''$$

- The decision boundary $\hat{p}(y = +1 \mid x) = \hat{p}(y = -1 \mid x)$ is linear form: $a^T x + b = 0$

$$a = \hat{\Sigma}^{-1}(\hat{\mu}_{+1} - \hat{\mu}_{-1})$$

$$b = -\tfrac{1}{2}(\hat{\mu}_{+1}^T \hat{\Sigma}^{-1} \hat{\mu}_{+1} - \hat{\mu}_{-1}^T \hat{\Sigma}^{-1} \hat{\mu}_{-1}) + \log(n_{+1}/n_{-1})$$

Relationship between least squares classification and Fisher discriminant analysis

- Setup
  - The average of training samples is zero: $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = \boldsymbol{0}$
  - Use linear-in-input ($\boldsymbol{x}$) model: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^\top \boldsymbol{x}$

Relationship between least squares classification and Fisher discriminant analysis

- Based on the setup on the previous slide, least squares classification is:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i \right)^2$$

- The direction of the decision boundary: $\widehat{\boldsymbol{\Sigma}}^{-1}(\widehat{\boldsymbol{\mu}}_{+} - \widehat{\boldsymbol{\mu}}_{-})$

- This is the same as Fisher discriminant analysis!
  - Normal vector of FDA is: $a = \hat{\Sigma}^{-1}(\hat{\mu}_{+1} - \hat{\mu}_{-1})$

Proving this is homework.

Contents

1. Classification by least squares regression
2. Multi-class classification problems
3. Fisher discriminant analysis
4. 0/1-loss and the margin

0/1 loss function and the margin

- In classification, we only need the sign of the learned function: $\widehat{y} = \mathrm{sign}\left(f_{\widehat{\boldsymbol{\theta}}}(\boldsymbol{x})\right)$

- Instead of the least squares, more natural to use 0/1 loss:

$$l_{0/1}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i) = \begin{cases} 0 & \text{if } \mathrm{sign}\left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right) = y_i \\ 1 & \text{otherwise} \end{cases}$$

$$= \frac{1 - \mathrm{sign}\left(m_i\right)}{2}$$

margin: $m_i = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) y_i$

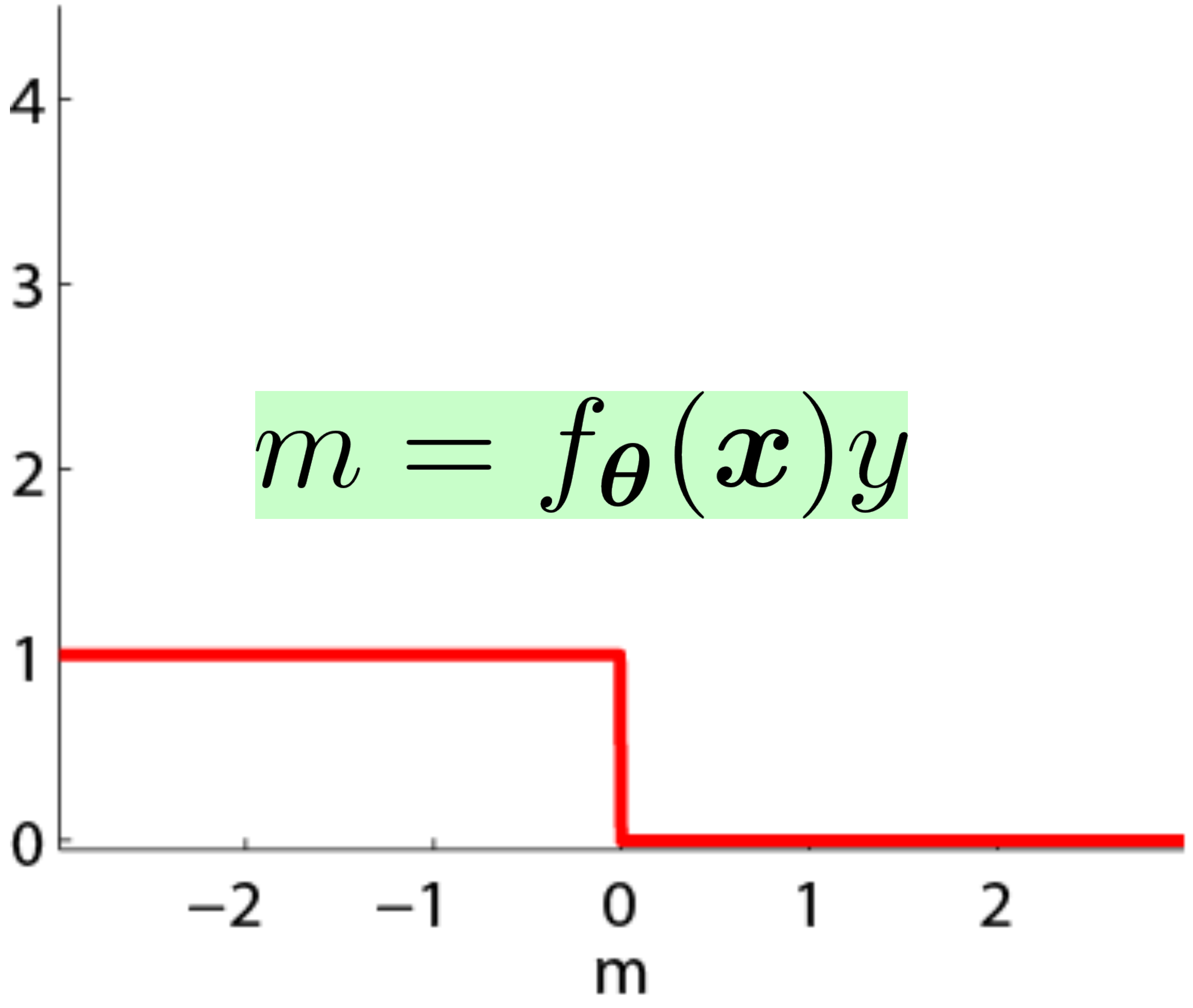
- $J_{0/1}(\boldsymbol{\theta}) = \sum_{i=1}^{n} l_{0/1}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$ is the number of misclassified samples.

0/1 loss function and the margin

- The 0/1-loss corresponds to the number of misclassified samples.
  - If the margin is positive, the error is 0
  - If the margin is negative, the error is 1
- As a classification loss, the 0/1-loss is ideal!
  - It directly evaluates misclassification
  - However, the 0/1-loss is a discrete function without gradients, which makes it difficult to optimize
- Minimizing the 0/1-loss is an NP-hard problem
  - It cannot be solved in a realistic time frame using conventional algorithms

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{i=1}^{n}\left(1 - \mathrm{sign}\left(m_i\right)\right)$$

$$m = f_{\boldsymbol{\theta}}(\boldsymbol{x})y$$



Math exercise

- Express the ℓ_2 loss function with the margin.

$$\frac{1}{2}\Big(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\Big)^2 \qquad m_i = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)y_i$$

- Useful info : $y_i = \pm 1$

Solution

$$\frac{1}{2}\Big(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i\Big)^2 = \frac{1}{2}\left(\frac{m_i}{y_i} - y_i\right)^2$$

$$m_i = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)y_i$$

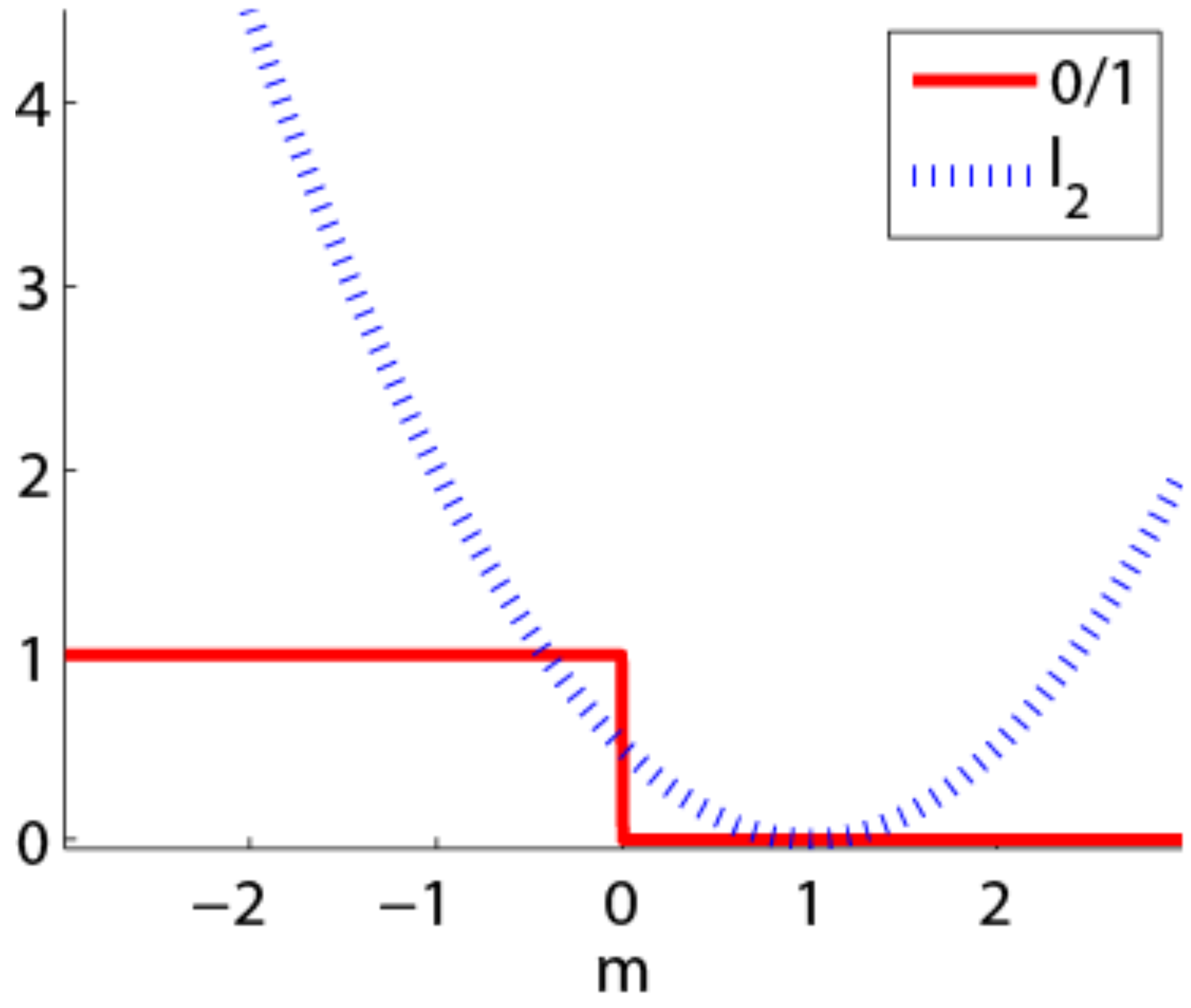
$$= \frac{1}{2}\left(\frac{m_i^2}{y_i^2} - 2m_i + y_i^2\right)$$

$$= \frac{1}{2}\left(m_i^2 - 2m_i + 1\right)$$

$$y_i^2 = 1$$

$$= \frac{1}{2}\left(m_i - 1\right)^2$$

ℓ_2-loss and the margin



$$m = f_{\boldsymbol{\theta}}(\boldsymbol{x})y$$

- The ℓ_2-loss function is a continuous function and easy to handle.
- It tries to turn a negative margin into a positive value.
- It also aims to reduce a large positive margin to +1.

Issue of ℓ_2-loss

- By trying to reduce a large positive margin to +1, the following data might not be correctly separated:

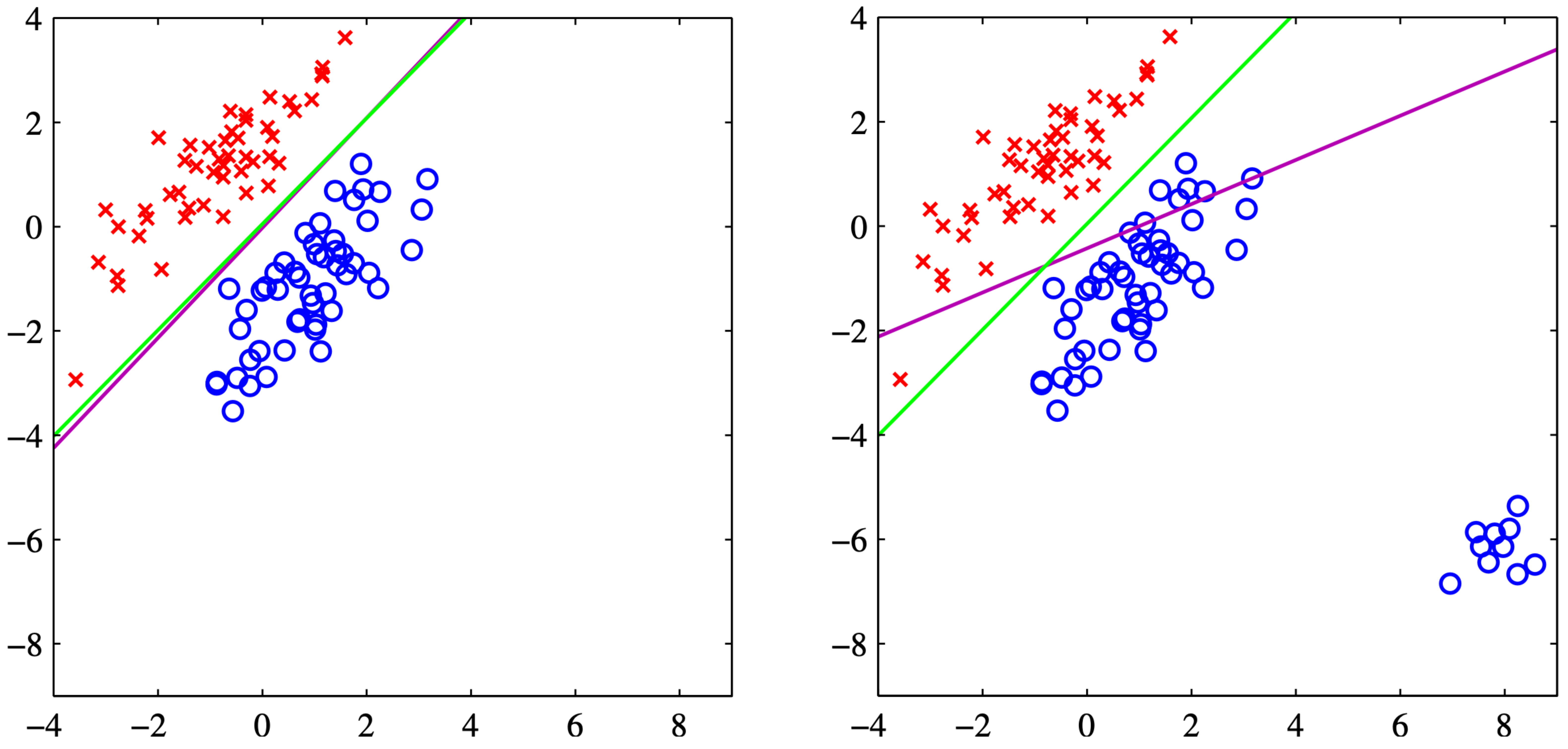
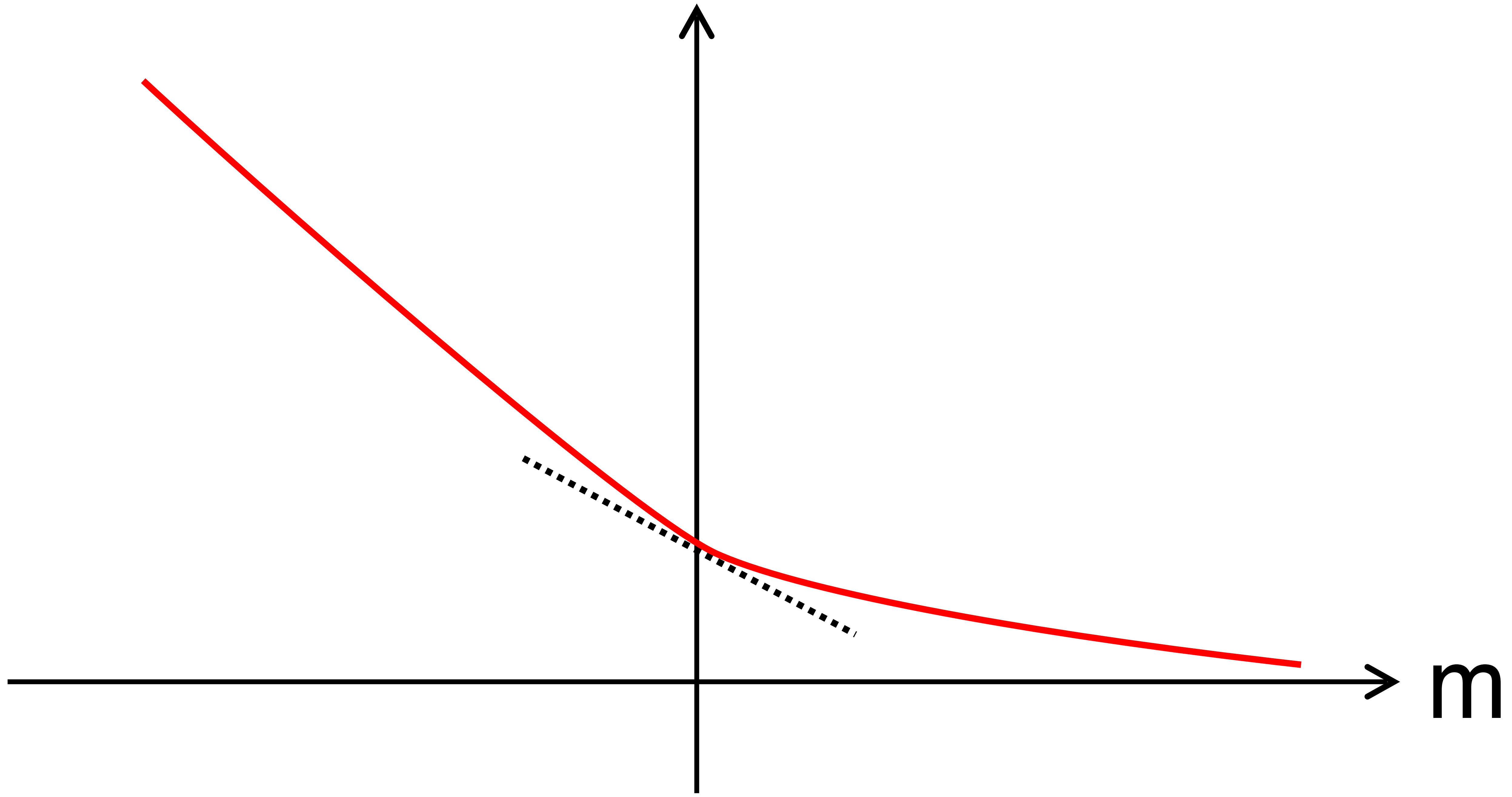


Figure from p186 of “Pattern Recognition and Machine Learning”.
https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

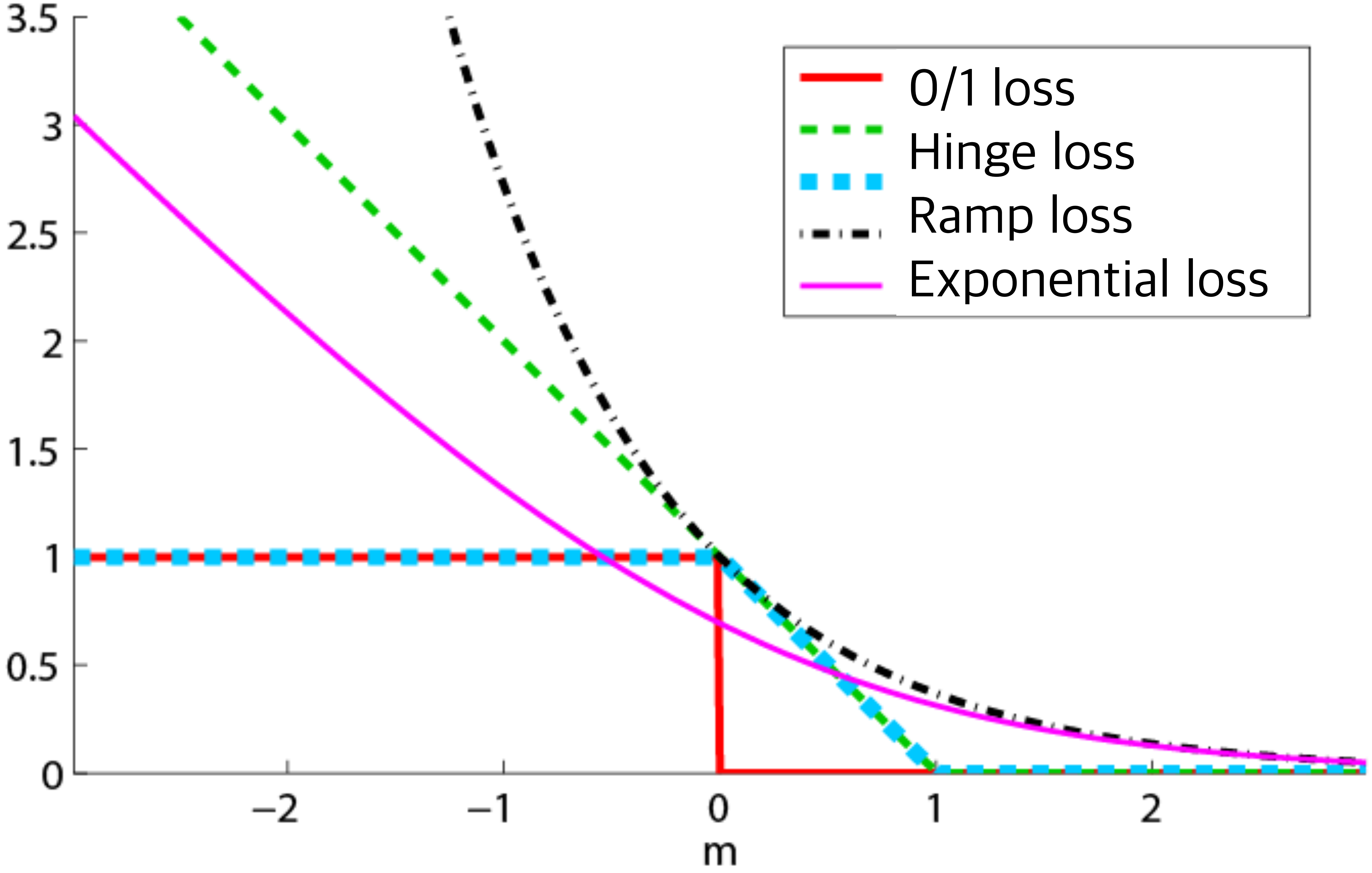
Surrogate loss

- A good loss function to use as a surrogate for the 0/1-loss should be monotonically non-increasing, with a negative slope at $m = 0$.
- This kind of loss function attempts to turn a negative margin into a positive one, while not reducing the positive margin.



Various surrogate losses

- In machine learning, various surrogate losses are used.

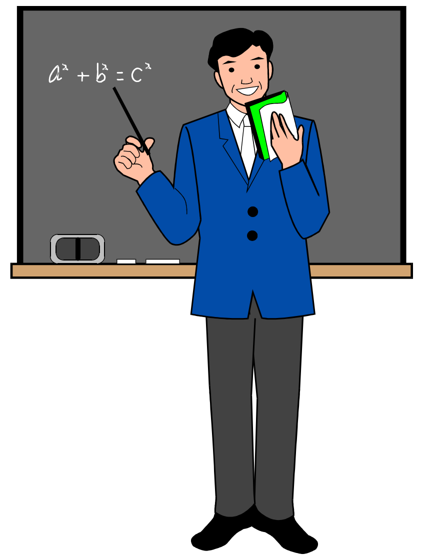


- Do they lead to the same classifier?
- Can they recover the posterior probability?

Contents

Summary

- **Reduce classification to regression**: Classification problems can be solved using least squares regression.
- **Reduce multi-class classification to binary classification**: Multi-class problems can be decomposed into multiple binary problems using either one-vs-all or one-vs-one strategies.
- **Probabilistic model**: Least squares classification is equivalent to Fisher discriminant analysis for linear-in-input models.
- **Margin-based surrogate loss**: The ℓ_2-loss is not a very good surrogate for the 0/1-loss.



Next lecture

- Support vector classification

Homework 1

- Show that $\hat{\boldsymbol{\theta}}$ (or the normal vector of the decision boundary) of the least squares classification is the same direction as $\widehat{\boldsymbol{\Sigma}}^{-1}\left(\widehat{\boldsymbol{\mu}}_{+} - \widehat{\boldsymbol{\mu}}_{-}\right)$ (or the normal vector of the decision boundary) obtained by Fisher discriminant analysis.

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\mathrm{argmin}} \frac{1}{2} \sum_{i=1}^{n} \Big(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i \Big)^2$$

- Mean zero inputs: $\dfrac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i = \mathbf{0}$
- Linear-in-input model: $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^{\top} \boldsymbol{x}$

Useful info for Homework 1

- $(\boldsymbol{X}^\top \boldsymbol{X})\widehat{\boldsymbol{\theta}} = \boldsymbol{X}^\top \boldsymbol{y}$ $\quad \boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)^\top$ $\quad \boldsymbol{y} = (y_1, \ldots, y_n)^\top$

- Consider the following steps:

1. Use the relation $\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = 0$ to express $\boldsymbol{\mu}$_and $\boldsymbol{X}^\top \boldsymbol{y}$ in terms of $\boldsymbol{\mu}_+, n_-, n_+$.
2. Express $\boldsymbol{X}^\top \boldsymbol{X}$ in terms of $\widehat{\Sigma}$, $\boldsymbol{\mu}+$, $n-$, n_+.
3. Use the following fact: For any vectors $\boldsymbol{v}, \boldsymbol{\theta}$, there exists a constant $c = c_{\boldsymbol{v},\boldsymbol{\theta}}$ such that $\boldsymbol{v}\boldsymbol{v}^\top \boldsymbol{\theta} = c \cdot \boldsymbol{v}$.

Homework 2

- Training samples: 5000 samples (500 for each digit)
- Test samples: 2000 samples (200 for each digit)
- Perform classification based on least squares regression with a Gaussian kernel.

Homework 2: Importing Handwritten Digits

- Download the file from:

  http://www.ms.k.u-tokyo.ac.jp/sugi/software/SML.zip

- Load the handwritten digits by:

```
load
digit.mat
```

- Then, we will have X (training) and T (test).
- Use the `whos` command:

```
> whos
  Name          Size              Bytes  Class
  T             256x200x10      4096000  double
  X             256x500x10     10240000  double
```

Homework 2: Importing Handwritten Digits

- See the below for loading the data in Python:

```
from scipy.io import loadmat
data = loadmat('digit.mat')
train = data['X']
test = data['T']
```

Homework 2 (Continued)

- Example: accuracy is 1908/2000=95.4%

One-vs-All

Predicted category

True category

	1	2	3	4	5	6	7	8	9	0
1	199	1	0	0	0	0	0	0	0	0
2	0	191	0	6	0	0	2	1	0	0
3	0	0	189	0	5	0	2	4	0	0
4	1	0	0	185	0	4	0	1	9	0
5	0	1	4	2	187	0	0	0	4	2
6	0	2	0	1	1	195	0	0	0	1
7	1	1	0	4	0	0	188	0	6	0
8	1	1	6	1	3	0	0	185	2	1
9	1	0	0	1	0	0	3	2	193	0
0	0	1	0	0	0	3	0	0	0	196