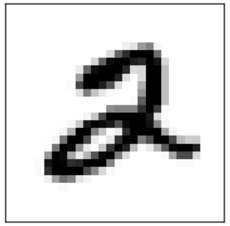
Introduction to machine learning II

- Classification
  - Linear model: logistic regression
  - Fitting of a model
  - Statistical interpretation: maximum likelihood
  - Perceptron
  - Multi-class classification: multi-class logistic regression
- Generalization errors
  - Overfitting
  - Cross-validation
- Support vector machines
  - Extension to multi-class classification
  - Kernel SVMs

Classification

- Is to identify to which of of *K* known classes the input **x** belongs



$\mathbf{x}$

0,1,2,3,4,5,6,7,8,9

- Suppose we are given N pairs of an observation and its associated true class

$$\{\mathbf{x}_n\}(n = 1, \ldots, N) \qquad \{d_n\}(n = 1, \ldots, N)$$

- Then we want to predict to which class a novel input **x** belongs

Classification

- The simplest case: two-class classification
    - Also called as binary classification

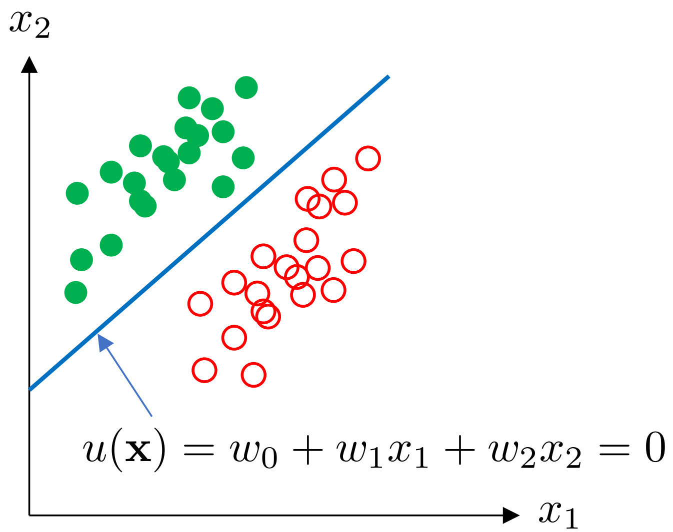
- We encode *d* by one of the following two methods
    - *d* is either 0 or 1
    - *d* is either -1 or 1

- Example: problem of estimating gender of a person from his/her weight and height

$$\mathbf{x} = [x_1, x_2]^\top \Longrightarrow d = \underset{\text{male}}{0} \ \text{or} \ \underset{\text{female}}{1}$$

Perceptron

- We choose the coding: *d* is -1 or 1
- We design *y(x)* that predicts *d* as follows:



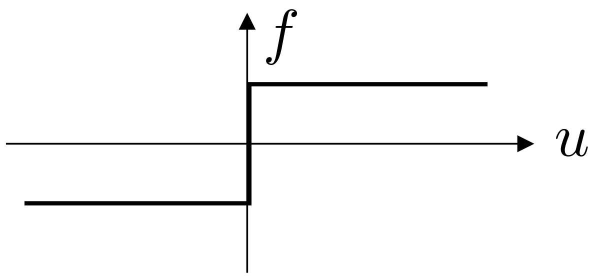
$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } u(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$u(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_I x_I$$

- Or equivalently

$$f(u) = \begin{cases} 1 & \text{if } u > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$y(\mathbf{x}, \mathbf{w}) = f(w_0 + w_1 x_1 + \cdots + w_I x_I)$$



Perceptron

- We design a function measuring errors of prediction

$$E(\mathbf{w}) = -\sum_{n \in \mathcal{M}} u(\mathbf{x}_n, \mathbf{w}) d_n = -\sum_{n \in \mathcal{M}} \mathbf{w}^\top \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} d_n$$

Set of misclassified samples

- Update the weight using one sample
- Iterate this for samples $\mathbf{x}_1, \mathbf{x}_2, \cdots$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} d_n$$

This procedure always decreases error at least for this sample

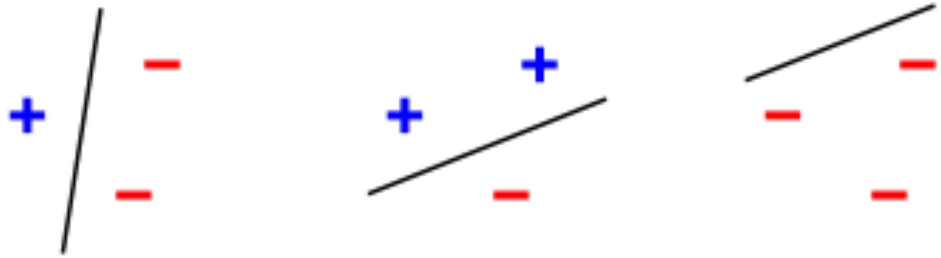
$$\mathbf{w}^{(t+1)\top} \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} d_n = \mathbf{w}^{(t)\top} \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} d_n - d_n \begin{bmatrix} 1 & \mathbf{x}_n \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} d_n$$

The total sum of errors is not guaranteed to decrease for each update; nevertheless, this iteration will converge for a finite number of iterations in *linearly separable* cases

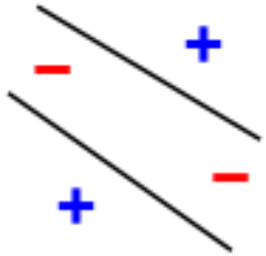
Linearly separable

- A problem is called linearly separable if all the data points are correctly classified by a single hyperplane

Linearly separable



Linearly inseparable



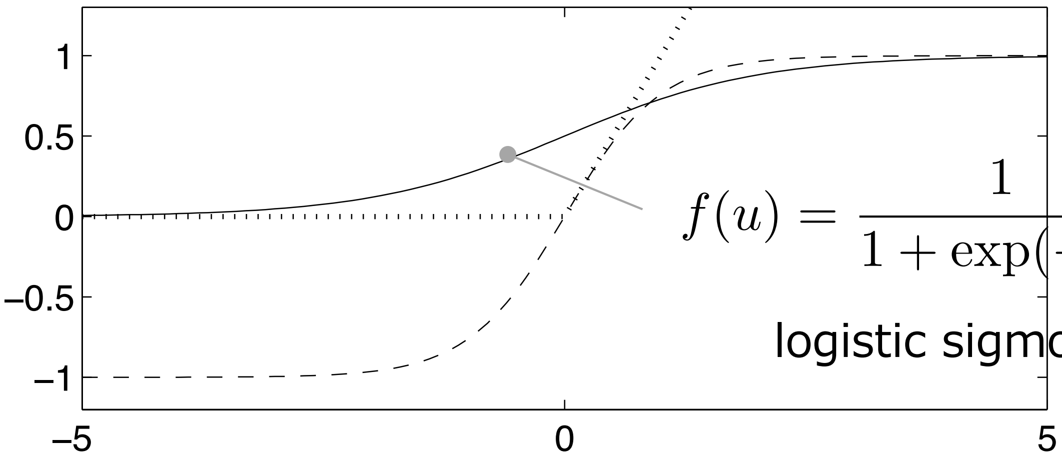
Logistic regression

- The posterior probability of d=1 is modeled by y(x) :

$$p(d=1\,|\,\mathbf{x}) \approx y(\mathbf{x}) \qquad \left(\quad p(d=0\,|\,\mathbf{x}) \approx 1 - y(\mathbf{x}) \quad \right)$$

- We choose integration of a *logistic function* with a linear function *for y*(x)

$$y(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-u(\mathbf{x}, \mathbf{w}))} \qquad u(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_I x_I$$



$$f(u) = \frac{1}{1 + \exp(-u)}$$

logistic sigmoid

Fitting to data

- We are given *N* pairs of an observation and its true class

$$\{\mathbf{x}_n\}(n = 1, \ldots, N) \qquad \{d_n\}(n = 1, \ldots, N)$$

- We wish to determine **w** that agrees well with these samples
- Toward this goal, we employ maximum likelihood estimation

- We choose the value of **w** that maximize the likelihood

$$l(\mathbf{w}) \equiv p(d_1, \ldots, d_N \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_N \,;\, \mathbf{w})$$

$$p(d_1, \ldots, d_N \,|\, \mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(d_n \,|\, \mathbf{x}_n)$$

- Or equivalently, we minimize more convenient negative log-likelihood:

$$E(\mathbf{w}) \equiv -\log l(\mathbf{w}) = -\sum_{n=1}^{N} \log p(d_n \,|\, \mathbf{x}_n)$$

Fitting to data

- We model the posterior probability of d=1 with y(x)

$$p(d=1\,|\,\mathbf{x}) \approx y(\mathbf{x})$$

- Using one of standard tricks, we may represent p(d|x) as

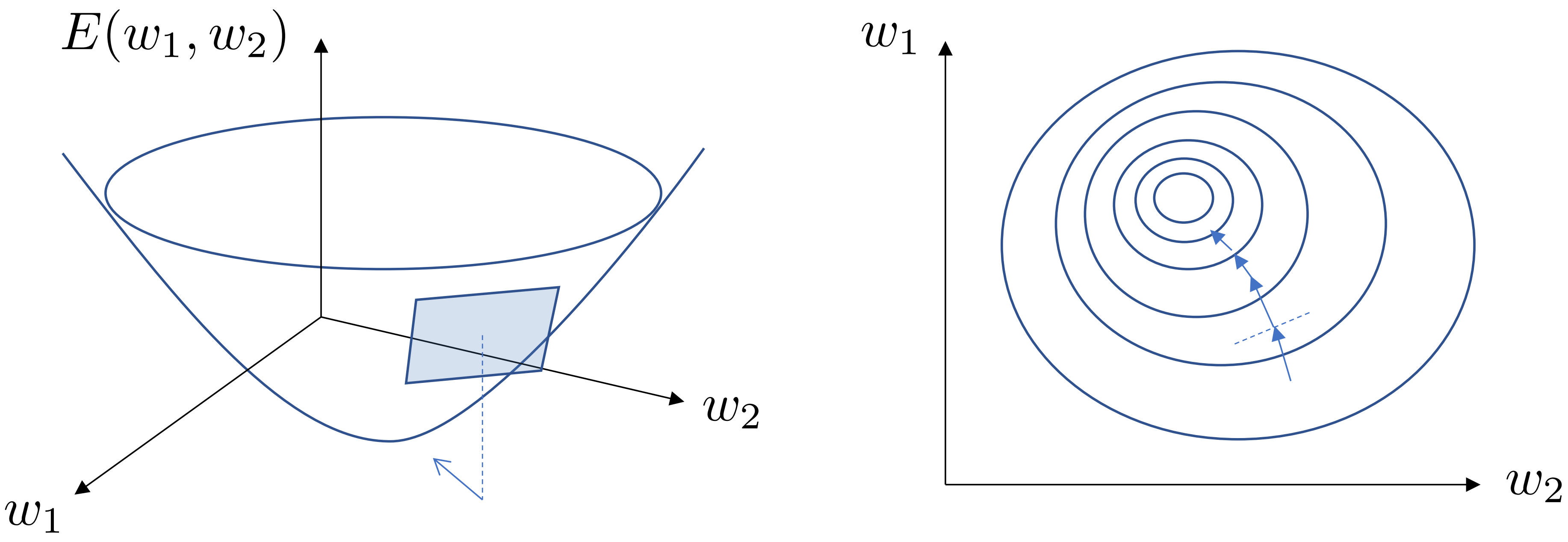
$$\begin{aligned} p(d\,|\,\mathbf{x}) &= \{p(d=1\,|\,\mathbf{x})\}^{d}\{p(d=0\,|\,\mathbf{x})\}^{1-d} \\ &= \{y(\mathbf{x},\mathbf{w})\}^{d}\{(1-y(\mathbf{x},\mathbf{w}))\}^{1-d} \end{aligned}$$

- The negative log-likelihood yields “cross-entropy loss function”

$$\begin{aligned} E(\mathbf{w}) &= -\sum_{n=1}^{N} \log p(d_n\,|\,\mathbf{x}_n) \\ &= -\sum_{n=1}^{N} \{d_n \log y(\mathbf{x}_n,\mathbf{w}) + (1-d_n)\log(1-y(\mathbf{x}_n,\mathbf{w}))\} \end{aligned}$$

Computing optimal solutions

- The optimal solution cannot be determined uniquely unlike the case of linear regression
- We then resort to iterative methods such as
  - Gradient descent
  - Newton’s methods



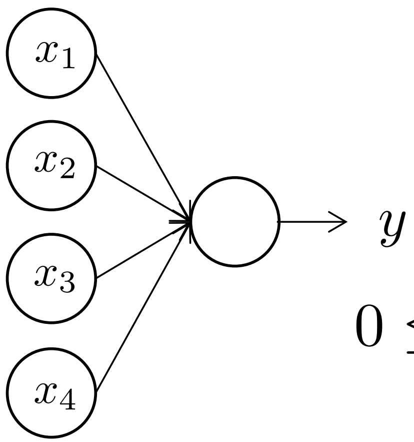
Multi-class classification

- 1-of-K coding is used for multi-class classification
  - E.g., the third class of K=10 classes

$$\mathbf{d} = [0,0,1,0,0,0,0,0,0,0]^\top$$

Two-class classification

$$d = 0 \ \text{ or } \ 1$$

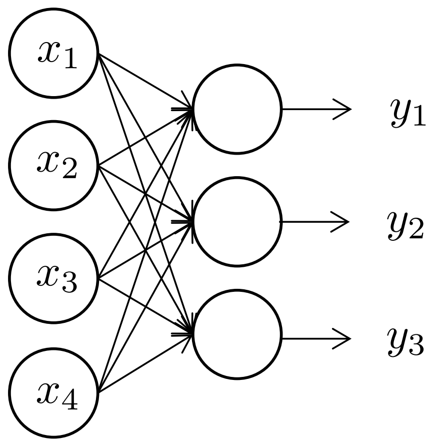


$$0 \leq y \leq 1$$

Logistic function

Three-class classification

$$\mathbf{d} = [0,0,1] \ \text{ or } \ [0,1,0] \ \text{ or } \ [1,0,0]$$



$$\sum_k y_k = 1$$

$$0 \leq y_k \leq 1$$

Softmax

Multi-class logistic regression

- We model the posterior of d_k=1 with y_k(x) :

$$p(d_k = 1 \,|\, \mathbf{x}) \approx y_k(\mathbf{x})$$

- We choose for y(x) linear function + softmax as follows:

$$y_k(\mathbf{x}) = \frac{\exp(u_k)}{\sum_{j=1}^{K} \exp(u_j)} \qquad \leftarrow \text{softmax関数}$$

$$u_k \equiv u(\mathbf{x}, \mathbf{w}_k) = w_{k0} + w_{k1}x_1 + \cdots + w_{kI}x_I$$

- Note that the outputs can viewed as probabilities

$$\sum_{k=1}^{K} y_k(\mathbf{x}) = 1 \qquad \Leftrightarrow \qquad \sum_{k=1}^{K} p(d_k = 1 \,|\, \mathbf{x}) = 1$$

Loss function of multi-class logistic regression

- How to derive a loss function?
- The joint probability may be written as

$$p(\mathbf{d}|\mathbf{x}) = \prod_{k=1}^{K} p(\mathcal{C}_k|\mathbf{x})^{d_k} = \prod_{k=1}^{K} p(d_k = 1|\mathbf{x})^{d_k}$$

- We employ maximum likelihood estimation:

$$L(\mathbf{w}) = \prod_{n=1}^{N} p(\mathbf{d}_n|\mathbf{x}_n; \mathbf{w}) = \prod_{n=1}^{N} \prod_{k=1}^{K} p(\mathcal{C}_k|\mathbf{x}_n)^{d_{nk}} = \prod_{n=1}^{N} \prod_{k=1}^{K} (y_k(\mathbf{x}; \mathbf{w}))^{d_{nk}}$$

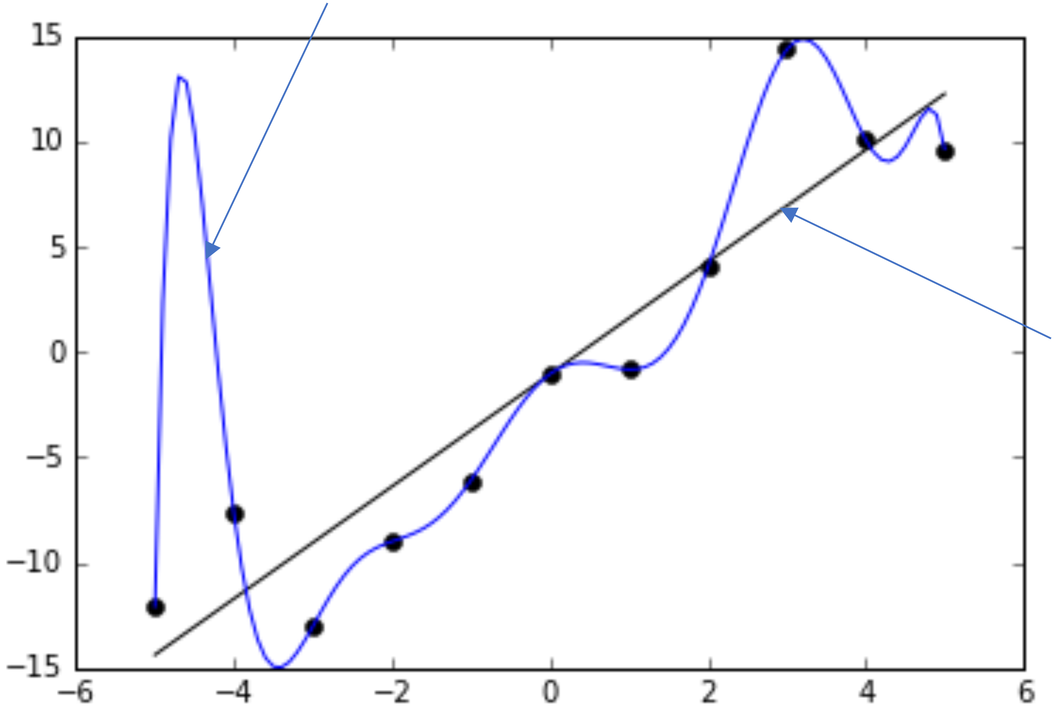
- What we do is to minimize the following function:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} d_{nk} \log y_k(\mathbf{x}_n; \mathbf{w})$$

Overfitting

- It is to fit a model to training samples excessively
  - Also called overtraining
- E.g., Suppose fitting a linear func. And 10^{th}-order polynomial func. to the same sample points:

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_{10} x^{10}$$



$$y = a_0 + a_1 x$$

https://commons.wikimedia.org/wiki/File:Overfitted_Data.png

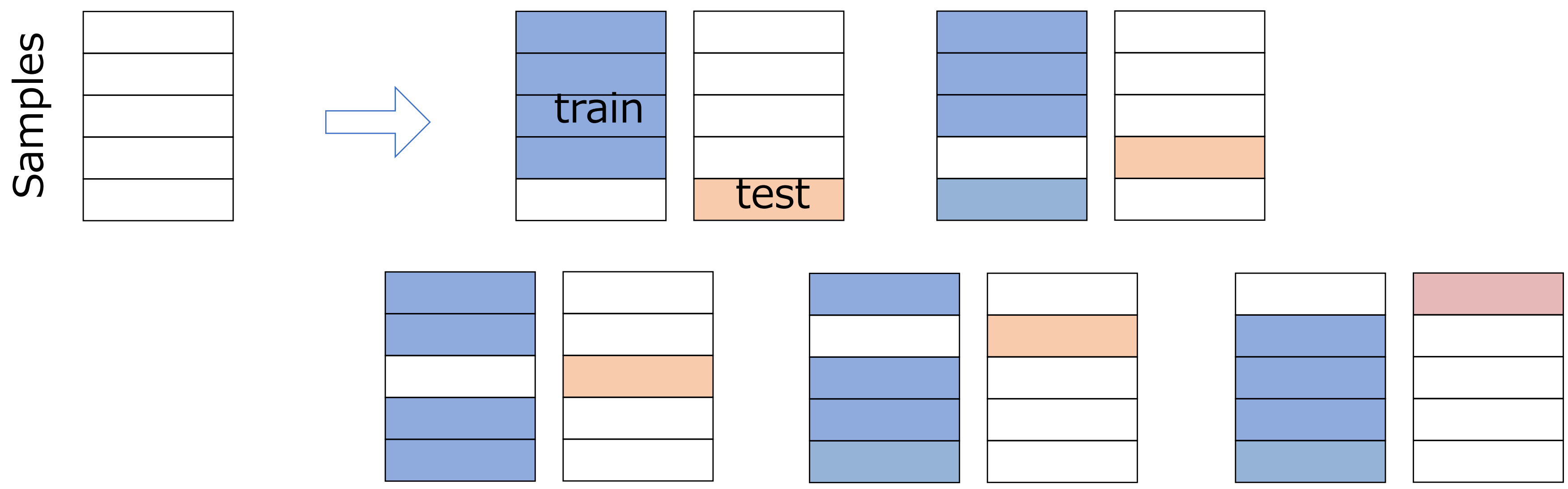
Training error and generalization error

- Training error
  - Sum (or average) of errors (loss values) for training samples
- Generalization error
  - Expected error (loss value) for a novel sample
- Test error (or validation error)
  - Sum (or average) of errors (loss values) for the samples at hand that were not used for training

$$E(\mathbf{w}) = \sum_{n=1}^{N} (d_n - y(\mathbf{x}_n, \mathbf{w}))^2$$

Cross validation

- We usually split the set of samples into two, one for training and the other for test (or validation)
- Accuracy will vary depending on how we split
- Cross validation is a method to evaluate accuracy by calculating the average accuracy for multiple different splits
  - E.g., 5-fold cross validation



Support vector machines (SVMs)

- Consider two class classification:

$$d_n = 1 \ \text{ or } \ -1$$

- Training samples: $(\boldsymbol{x}_1, d_1), (\boldsymbol{x}_2, d_2), \cdots, (\boldsymbol{x}_N, d_N)$

- Classification:

$$y(\mathbf{x}) = \begin{cases} 1 & \text{if } u(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

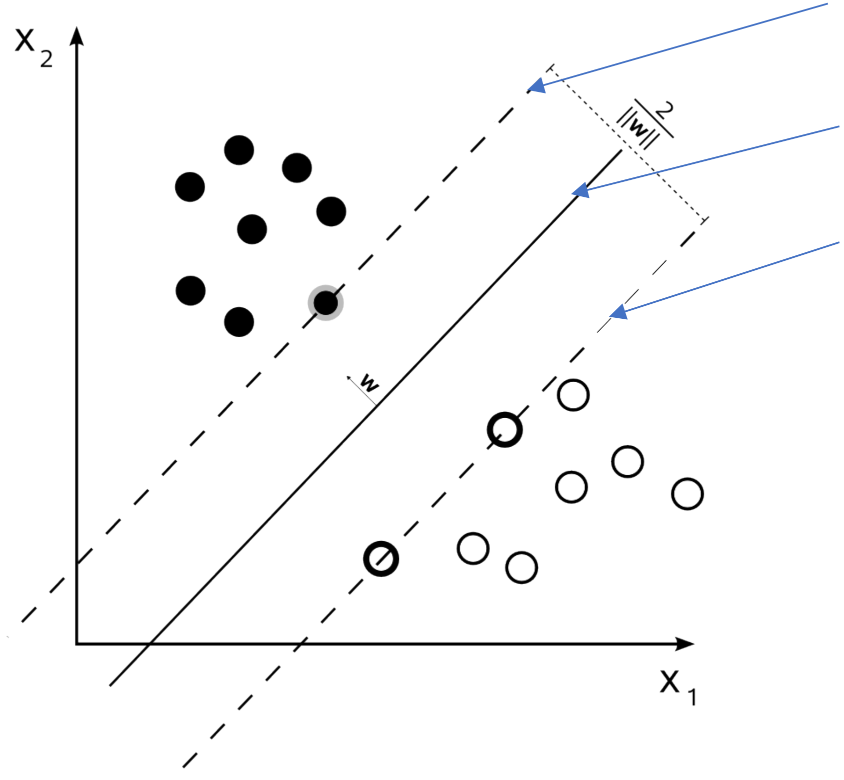
$$u(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \cdots + w_I x_I \ = w_0 + \mathbf{w}^\top \mathbf{x}$$

- Consider minimization of

$$\|\mathbf{w}\| \quad \text{subject to} \quad d_n(w_0 + \mathbf{w}^\top \mathbf{x}) \geq 1$$

Support vector machines (SVMs)

- We assume data are linearly separable
- We wish to find the two parallel hyperplanes that have the maximum distance between them and each of which separates the samples
- We then choose the parallel hyperplane that are equally distant to them



$$w_0 + \mathbf{w}^\top \mathbf{x} = 1$$

$$w_0 + \mathbf{w}^\top \mathbf{x} = 0$$

$$w_0 + \mathbf{w}^\top \mathbf{x} = -1$$

Support vector machines (SVMs)

- For linearly non-separable samples, we consider the soft-margin

$$E(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{n}^{N} \left(\underbrace{\max}_{\substack{\text{Avoid excessive high evaluation} \\ \text{of correctly classified samples}}} \left(0, 1 - \mathbf{w}^\top \begin{bmatrix} 1 \\ \mathbf{x}_n \end{bmatrix} d_n \right) \right)^2$$

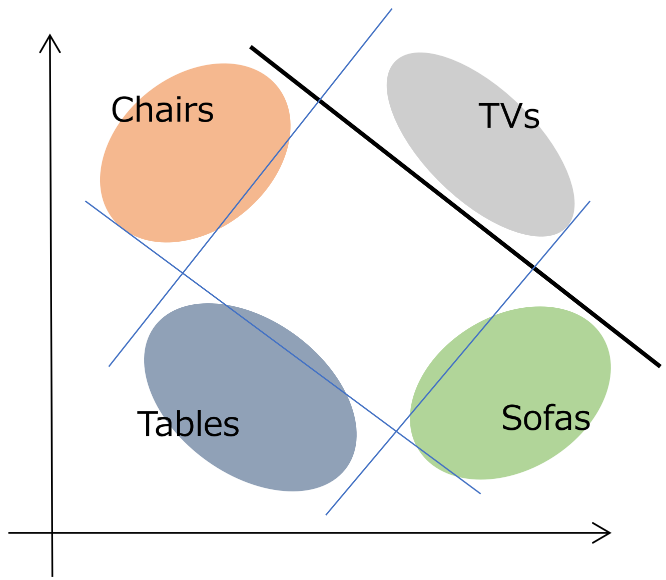
- We can always obtain globally optimal solution for the problem

Multi-class classification with two-class classifier

- One-versus-the-rest classifier is the most popular

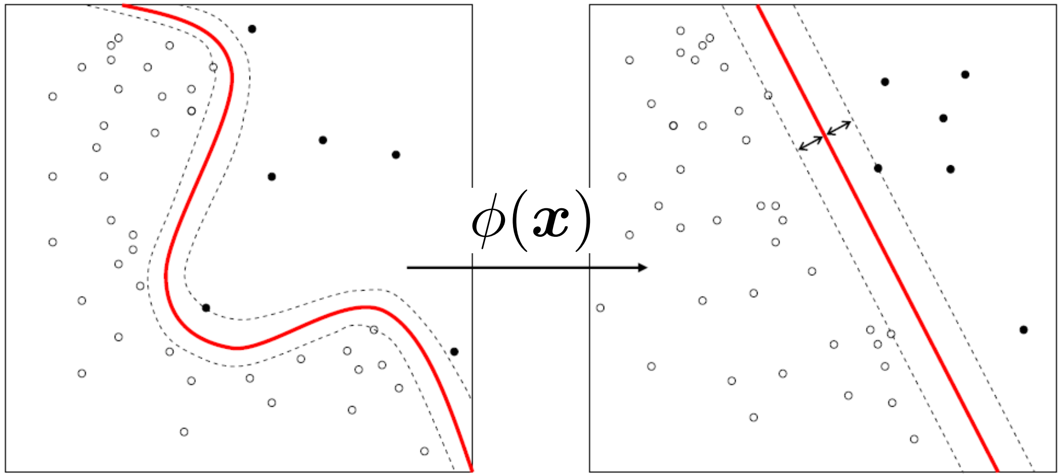
1. Train k two-class classifiers $y(x)$ that separates the class k and all other classes (the rest)
2. Regarding the output of the model y(x) as "score" of the class, classify the input into the class with the highest score

$$\operatorname*{argmax}_{k} y_k(\mathbf{x})$$



Kernel SVMs (nonlinear SVMs)

- Project the feature space with a nonlinear transformation *Φ*; all the samples are projected to the new space
- Train a linear SVM in the new space using the projected samples
- We do not need explicitly specify Φ; instead specify the inner-product of two projected samples



$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$$

- linear: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$.
- polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i{}^T \mathbf{x}_j + r)^d$, $\gamma > 0$.
- radial basis function (RBF): $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$.
- sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i{}^T \mathbf{x}_j + r)$.