COMP 7990
Principles and Practices of Data Analytics

Lecture 5: Artificial Neural Network, Support Vector Machine and k Nearest Neighbors (k-NN) Algorithm

Dr. Eric Lu Zhang

Outline for Data Preprocessing and Data Mining

- **Data Preprocessing**
- **Supervised learning**

❖Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

❖Classification

1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

Outline for Data Preprocessing and Data Mining

- Data Preprocessing
- **Supervised learning**

❖Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

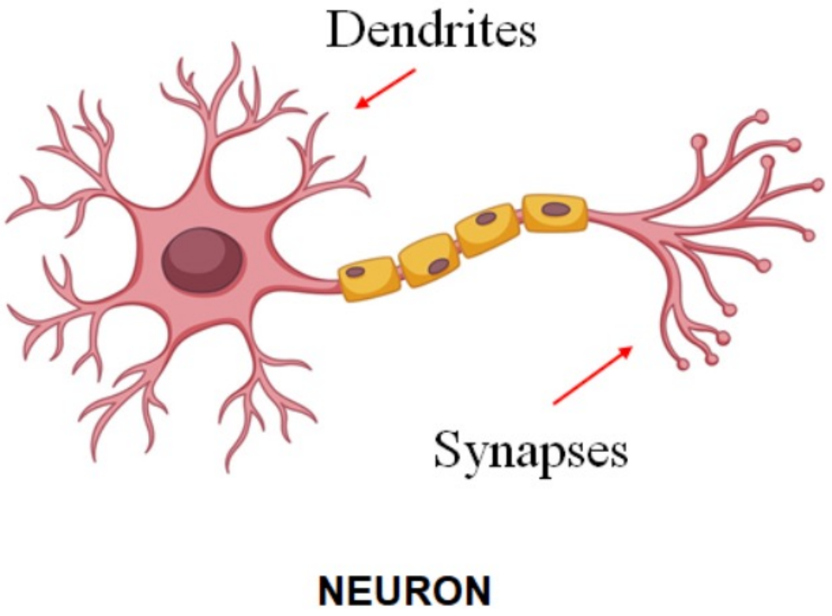
❖Classification

1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- Unsupervised learning

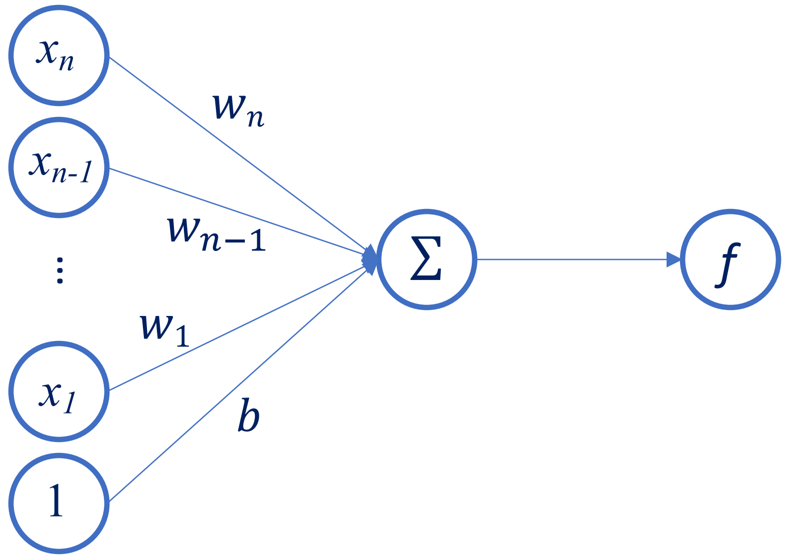
1. K-means Clustering
2. Hierarchical Clustering

Artificial Neural Networks



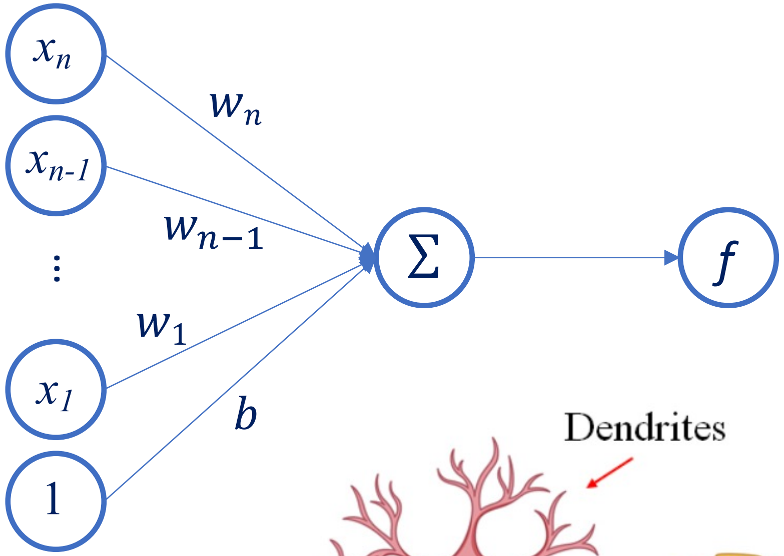
Recall on Perceptron

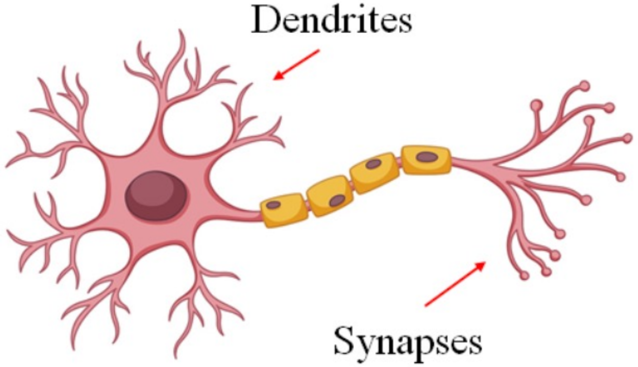
- A linear classification function $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b$
  - $y = 1$ if $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b > 0$
  - $y = -1$ if $f(\mathbf{x}) = \sum_{j=1}^{n} w_j x_j + b < 0$



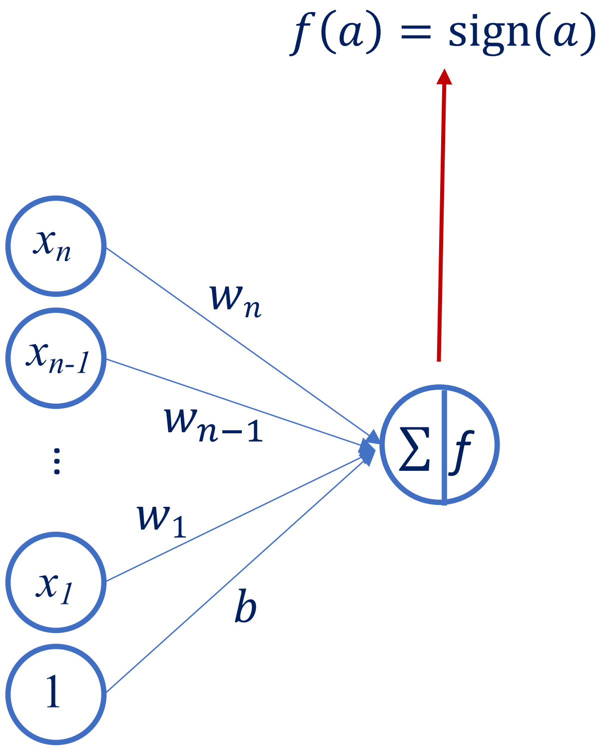
- $[x_1, \ldots, x_n]$: inputs
- $[w_1, \ldots, w_n]$: weights
- b: bias term
- $\sum$: summation
- f: activation function (sign function used)

Simplified Illustration of a Neuron



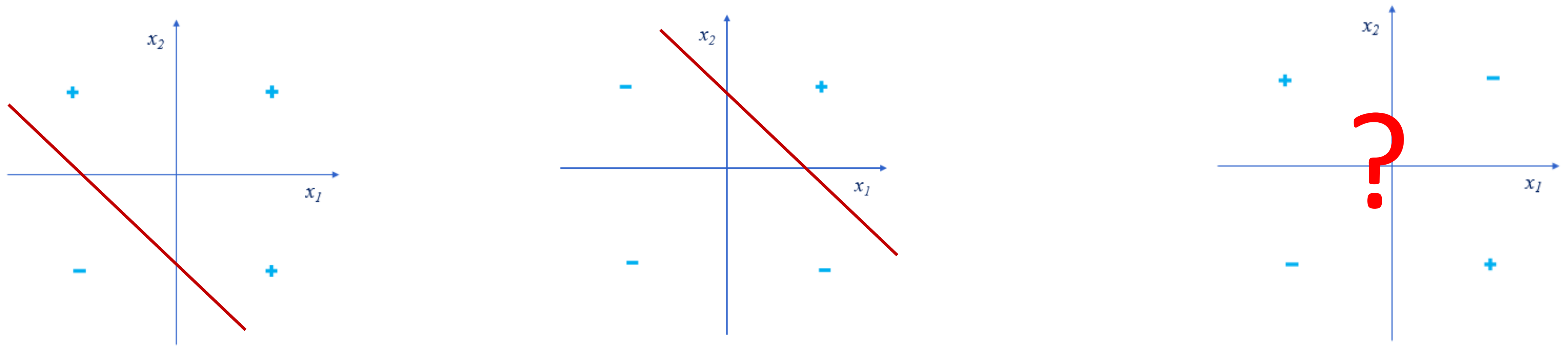


NEURON

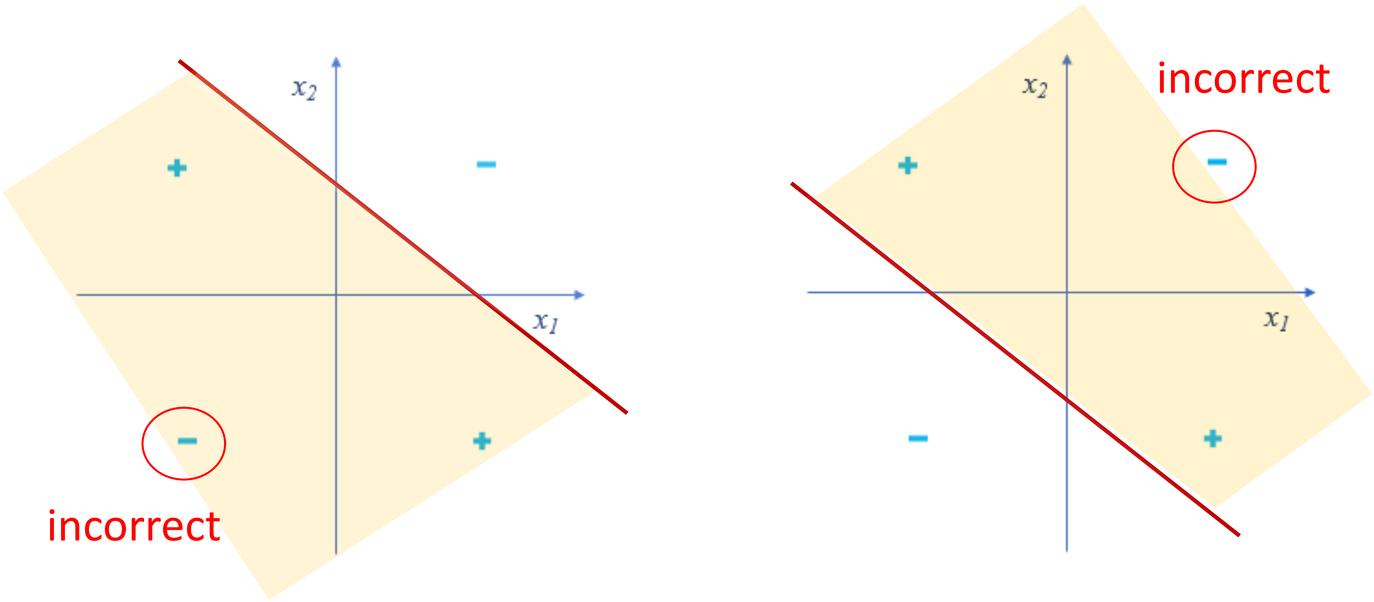


Limitation of Perceptron

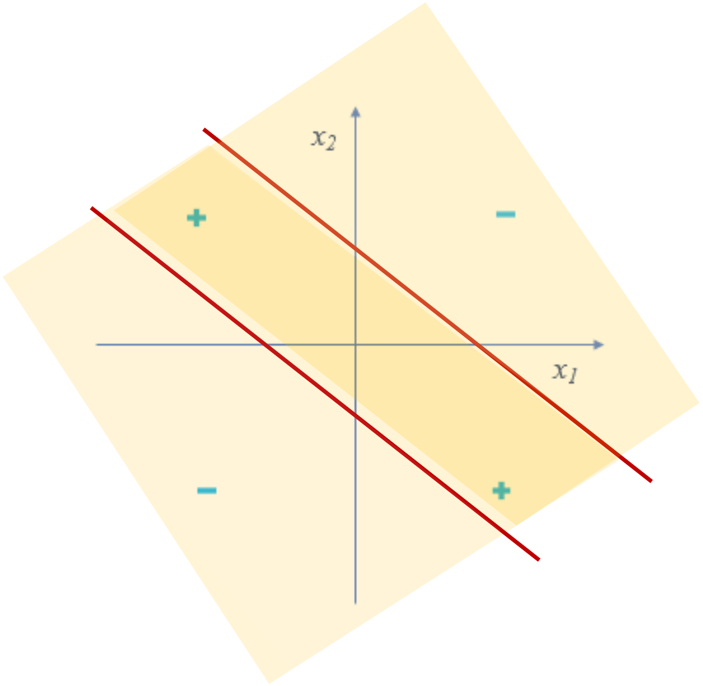
- Many classification problems are NOT linearly separable.

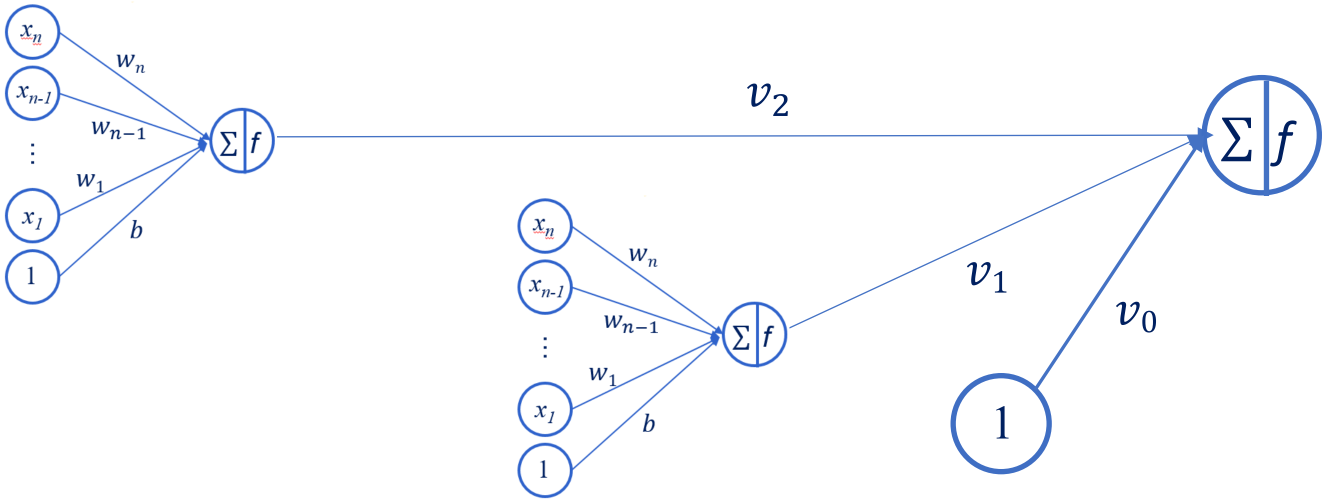


- Possible solution for nonlinear classification problem: Composition
  - Multiple Layer Perceptron (also called Feedforward Neural Network)

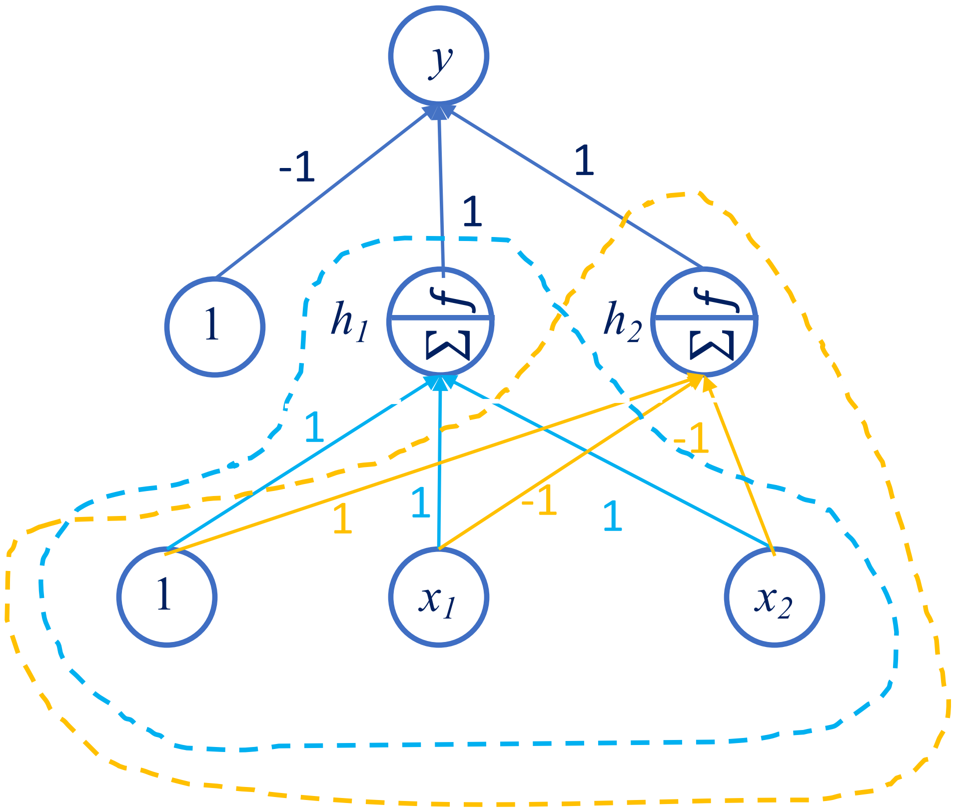


Compose them together

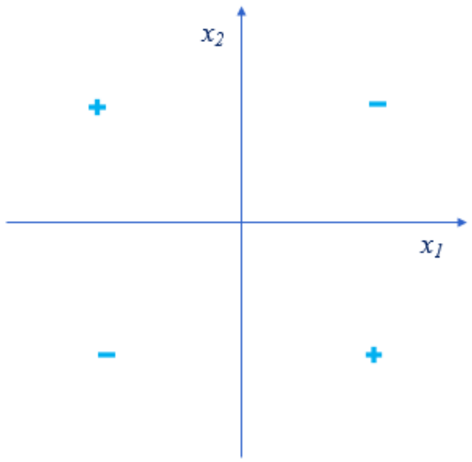




Multi-Layer Perceptron for Nonlinear Classification



x_1	x_2	y
1	-1	1
1	1	-1
-1	1	1
-1	-1	-1

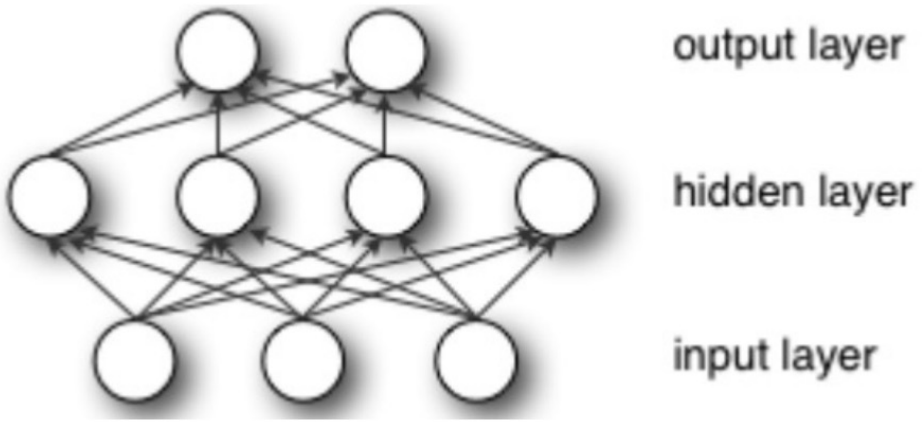


This multi-layer perceptron can solve this nonlinear classification problem.

How to learn the model parameter (i.e., weights on the edge) from data?

Multi-Layer Perceptron

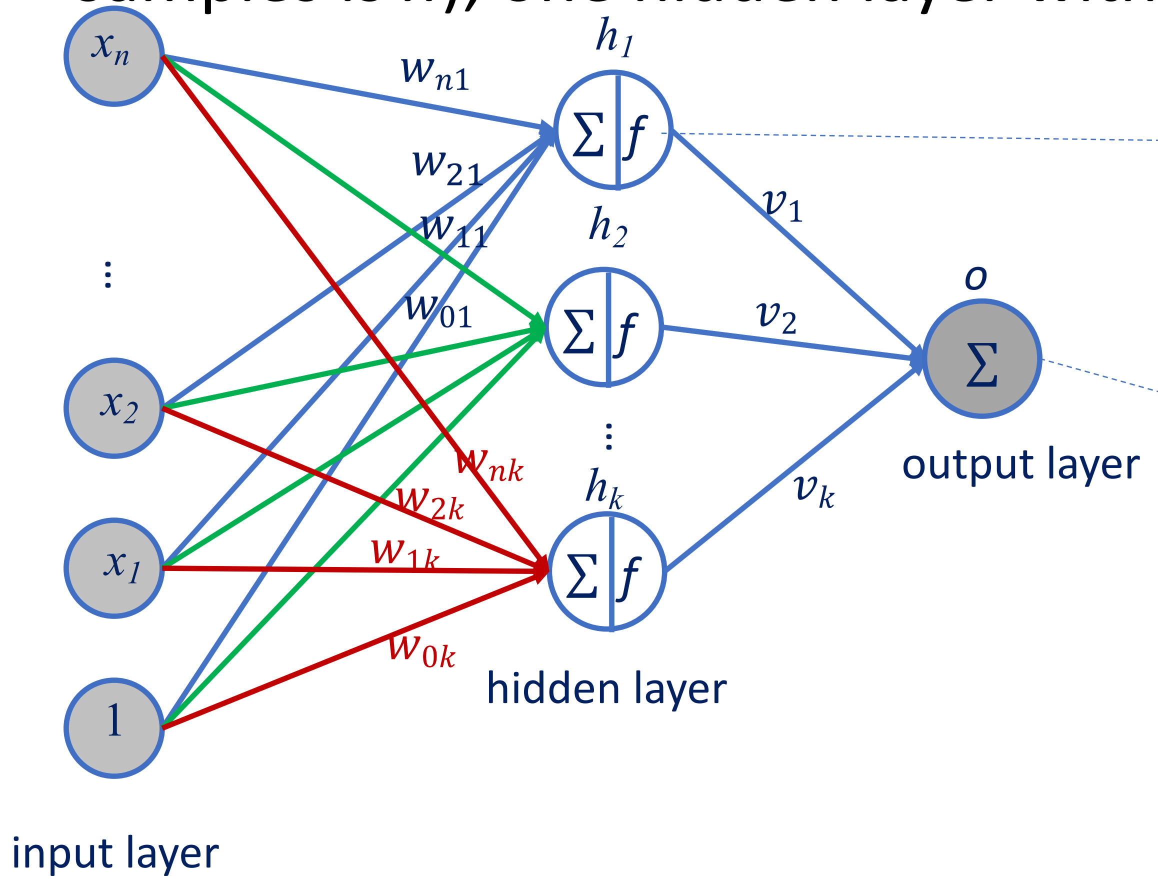
- Composed of several Perceptron-like units arranged in multiple layers



- Consists of an input layer, one or more hidden layers, and an output layer
- Nodes in the hidden layers carry out nonlinear transformation of the inputs
- Universal Approximator (Hornik, 1991)

Multi-Layer Perceptron (MLP) with One Hidden Layer

- Multi-Layer Perceptron with *n* inputs (i.e. the dimension of input samples is *n*), one hidden layer with *k* nodes, and one output.

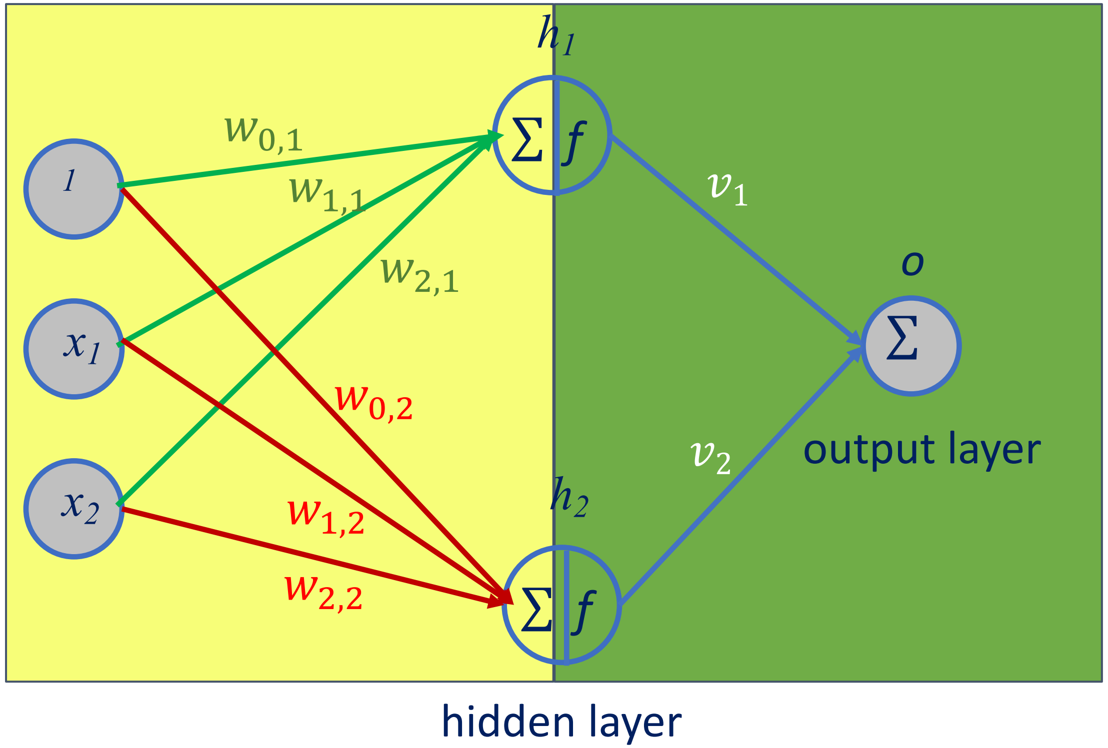


Nonlinear "activation function" is required. Otherwise, the model would reduce to a linear model. (Why?)

The output *o* is a weighted sum of the preceding layer's hidden nodes

Output of MLP with 2 Hidden Nodes and 2-dimensional Input

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$



$$o = v_1 h_1 + v_2 h_2 = \sum_{i=1}^{2} v_i h_i$$

$$h_1 = f\left(\sum_{i=0}^{2} w_{i1} x_i\right) = f(\mathbf{w}_1^T \mathbf{x})$$

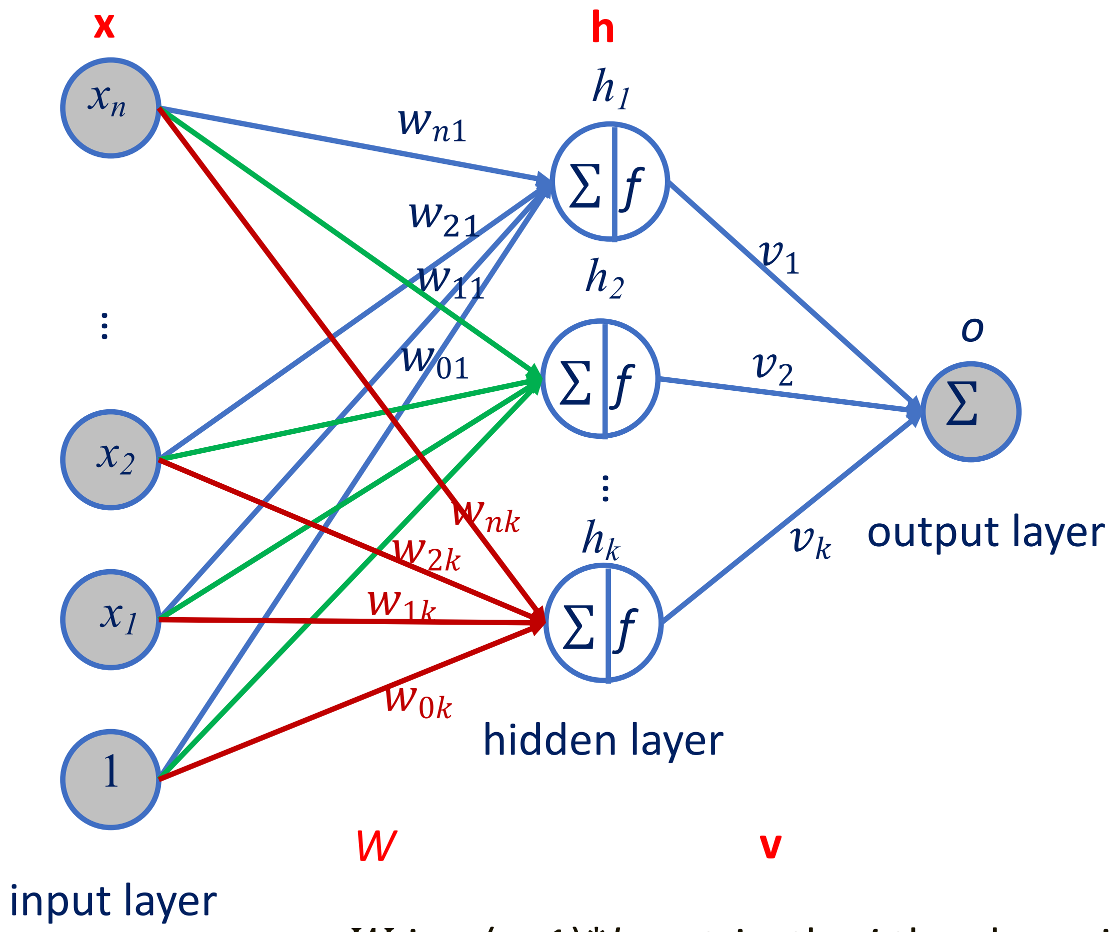
$$h_2 = f\left(\sum_{i=0}^{2} w_{i2} x_i\right) = f(\mathbf{w}_2^T \mathbf{x})$$

$$W = \begin{bmatrix} w_{0,1} & w_{0,2} \\ w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} = [\mathbf{w}_1 \quad w_2]$$

$$W^T = \begin{bmatrix} w_{0,1} & w_{1,1} & w_{2,1} \\ w_{0,2} & w_{1,2} & w_{2,2} \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1{}^T \\ \mathbf{w}_2{}^T \end{bmatrix}$$

W is a 3-by-2 matrix with the *j*-th column in W denoting the weights of the *j*-th node in the hidden layer.

Output of MLP with *k* Hidden Nodes and *d*-dimensional Input



$$o = \sum_{i=1}^{k} v_i h_i = \mathbf{v}^T \mathbf{h}$$

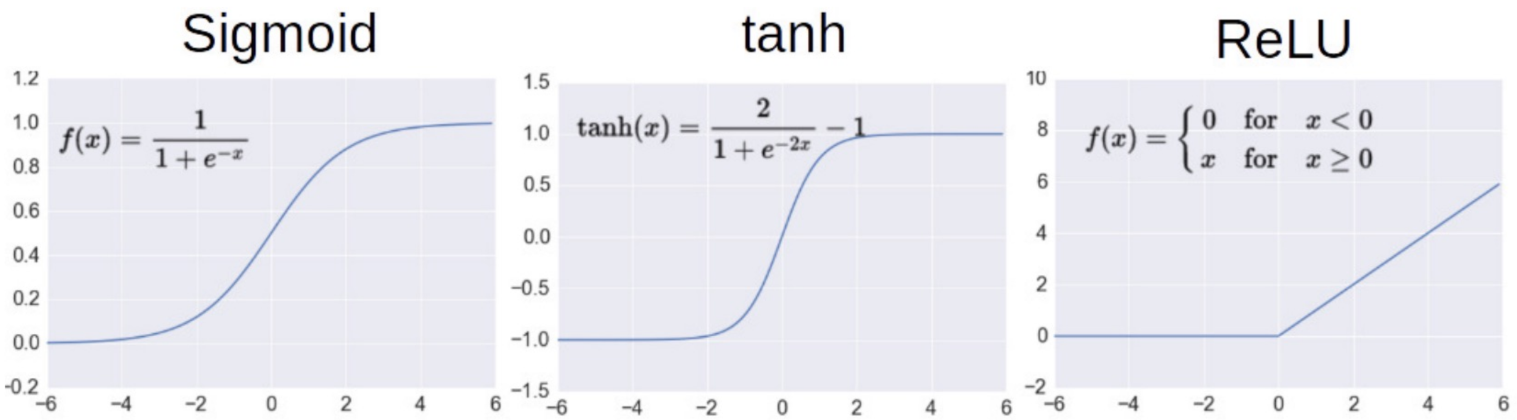
$$\mathbf{h} = f(W^T\mathbf{x}) \quad \left\{ \begin{array}{l} h_1 = f\left(\sum_{i=0}^{n} w_{i,1} x_i\right) = f(\mathbf{w}_1^T \mathbf{x}) \\ h_2 = f\left(\sum_{i=0}^{n} w_{i,2} x_i\right) = f(\mathbf{w}_2^T \mathbf{x}) \\ \vdots \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\ h_k = f\left(\sum_{i=0}^{n} w_{i,k} x_i\right) = f(\mathbf{w}_k^T \mathbf{x}) \end{array} \right.$$

$$o = \mathbf{v}^T f(W^T \mathbf{x})$$

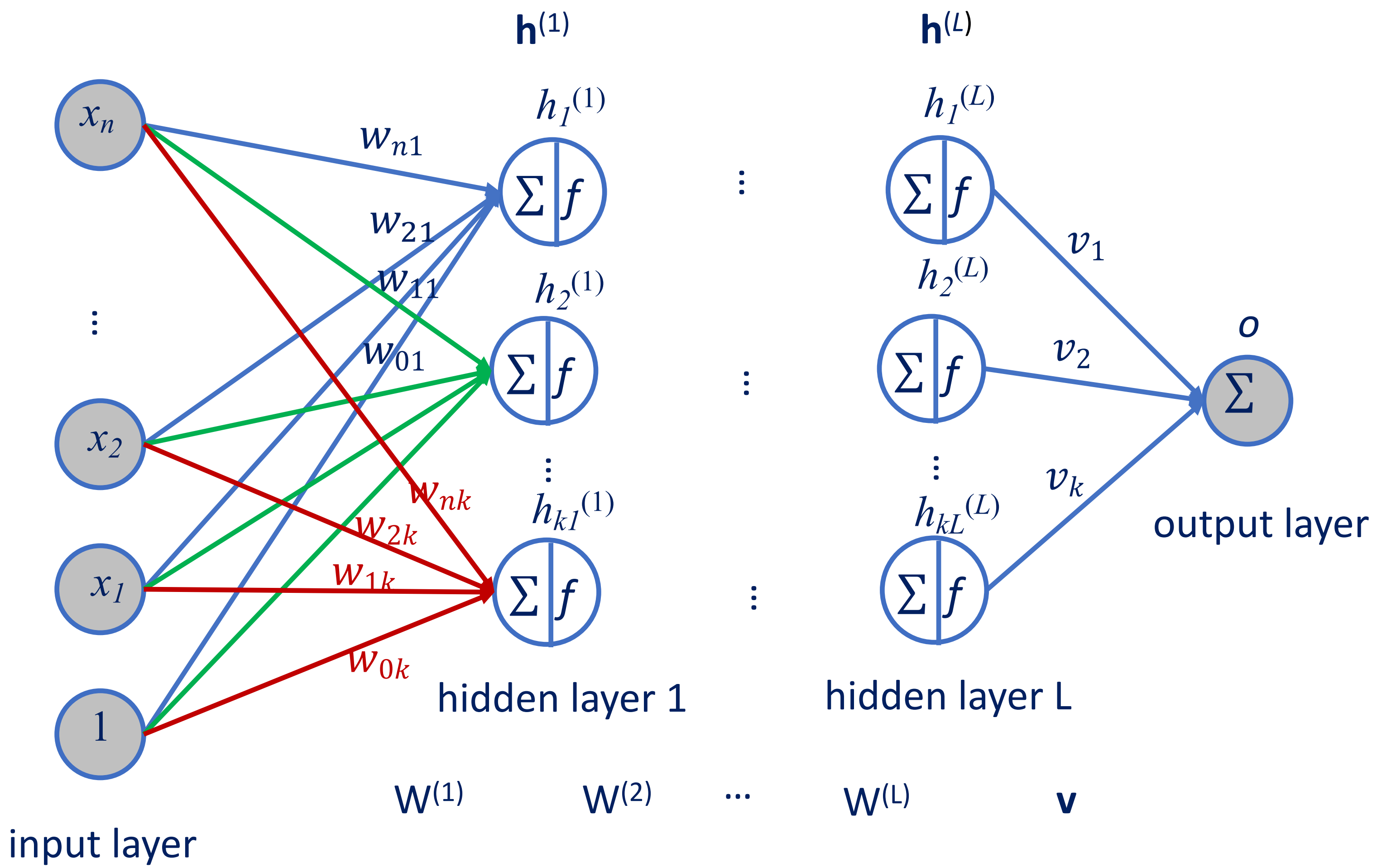
W is a $(n+1)*k$ matrix, the *j*-th column in W denotes the weights of the *j*-th node in the hidden layer.

Commonly Used Nonlinear Activation Functions

- Some common choices for nonlinear activation function f
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - Tanh: $f(x) = 2\sigma(2x) - 1 = \frac{2}{1+\exp(-2x)} - 1$(range between -1 and +1)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



Multi-Layer Perceptron



- For an MLP with L hidden layers, $\mathbf{h}^{(1)}$, …, $\mathbf{h}^{(L)}$ and the scalar-valued output is computed as

$$o = \sum_{i=1}^{k} h_i v_i = \mathbf{v}^T \mathbf{h}^{(L)}$$

$$\mathbf{h}^{(L)} = f({W^{(L)}}^T \mathbf{h}^{(L-1)})$$
$$\mathbf{h}^{(L-1)} = f({W^{(L-1)}}^T \mathbf{h}^{(L-2)})$$
$$\vdots \qquad \vdots$$
$$\mathbf{h}^{(2)} = f({W^{(2)}}^T \mathbf{h}^{(1)})$$
$$\mathbf{h}^{(1)} = f({W^{(1)}}^T \mathbf{x})$$

Why nonlinear "activation function" is required?

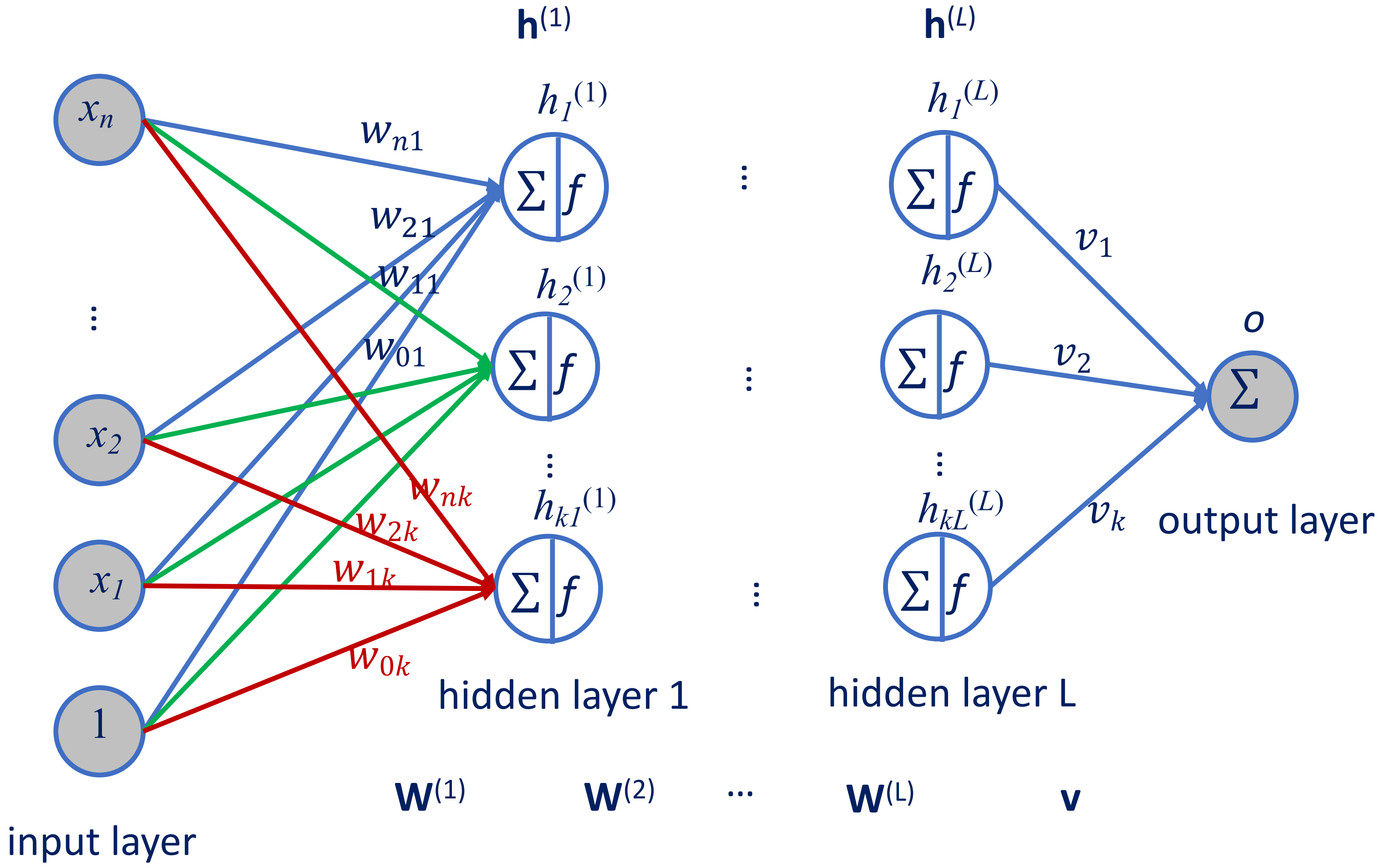
Why nonlinear "activation function" is required

- If we remove the nonlinear "activation function", the output of the neural network will be reduced to

$$
\begin{aligned}
o &= \sum_{i=1}^{k} h_i v_i = \mathbf{v}^T \mathbf{h}^{(L)} \\
&= \mathbf{v}^T {W^{(L)}}^T \mathbf{h}^{(L-1)} \\
&= \mathbf{v}^T {W^{(L)}}^T {W^{(L-1)}}^T \mathbf{h}^{(L-2)} \\
&\dots \\
&= \underbrace{\mathbf{v}^T {W^{(L)}}^T {W^{(L-1)}}^T \dots {W^{(2)}}^T {W^{(1)}}^T}\mathbf{x}
\end{aligned}
$$

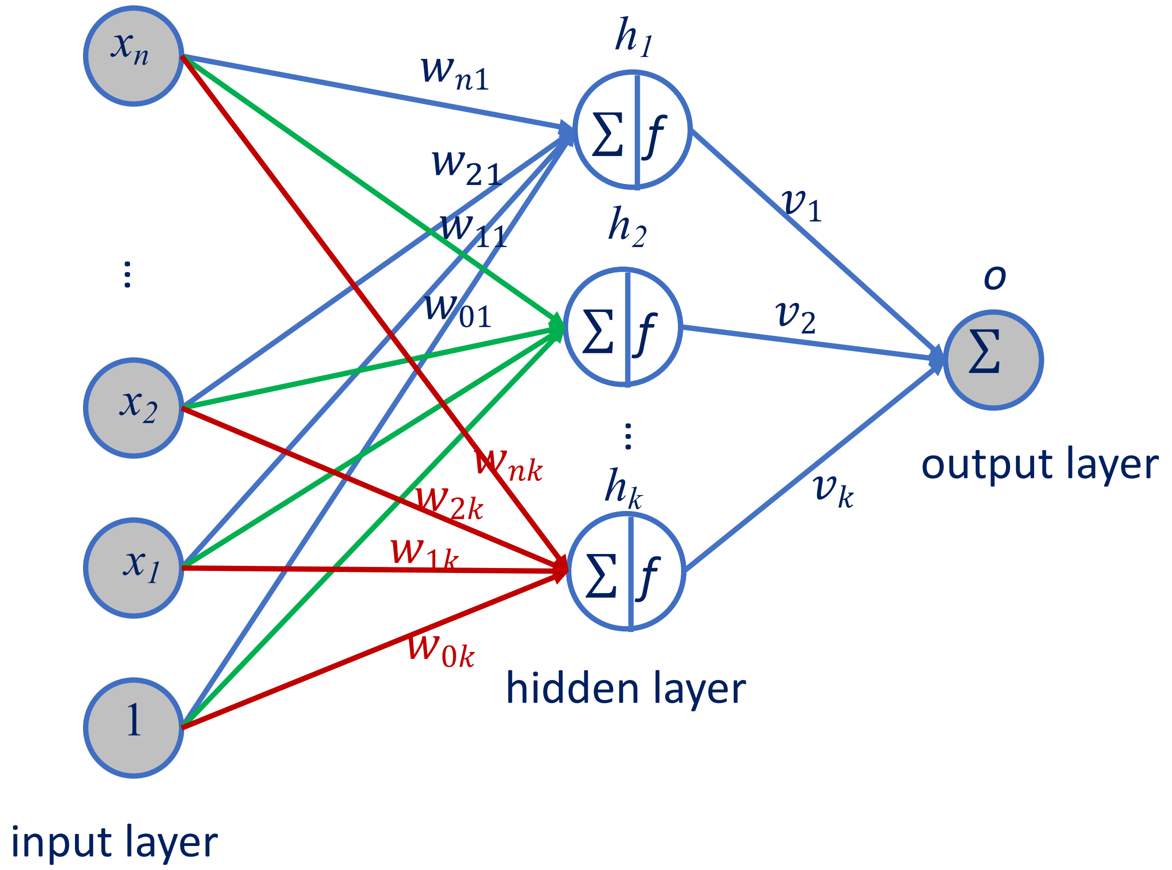
All these operations are linear transformation. Therefore, without nonlinear activation function, it will reduce to a simple linear model.

Learning MLP Via Backpropagation



- For a given training dataset, we want to learn the parameter ($W^{(1)}$, …, $W^{(L)}$, **v**) by minimizing some loss function.
- **Backpropagation** (gradient descent + chain rule for derivatives) is commonly used to do this efficiently.

Learning MLP (one hidden layer)



- Given one data sample of input and true output $\{\boldsymbol{x}, y\}$, our goal is to minimize $(y - o)^2 = (y - \mathbf{v}^T f(W^T \mathbf{x}))^2$

- Given m data sample of input and true output $\{\{\boldsymbol{x}_1, y_1\}, \{\boldsymbol{x}_2, y_2\}, \dots, \{\boldsymbol{x}_m, y_m\}\}$, the goal becomes:

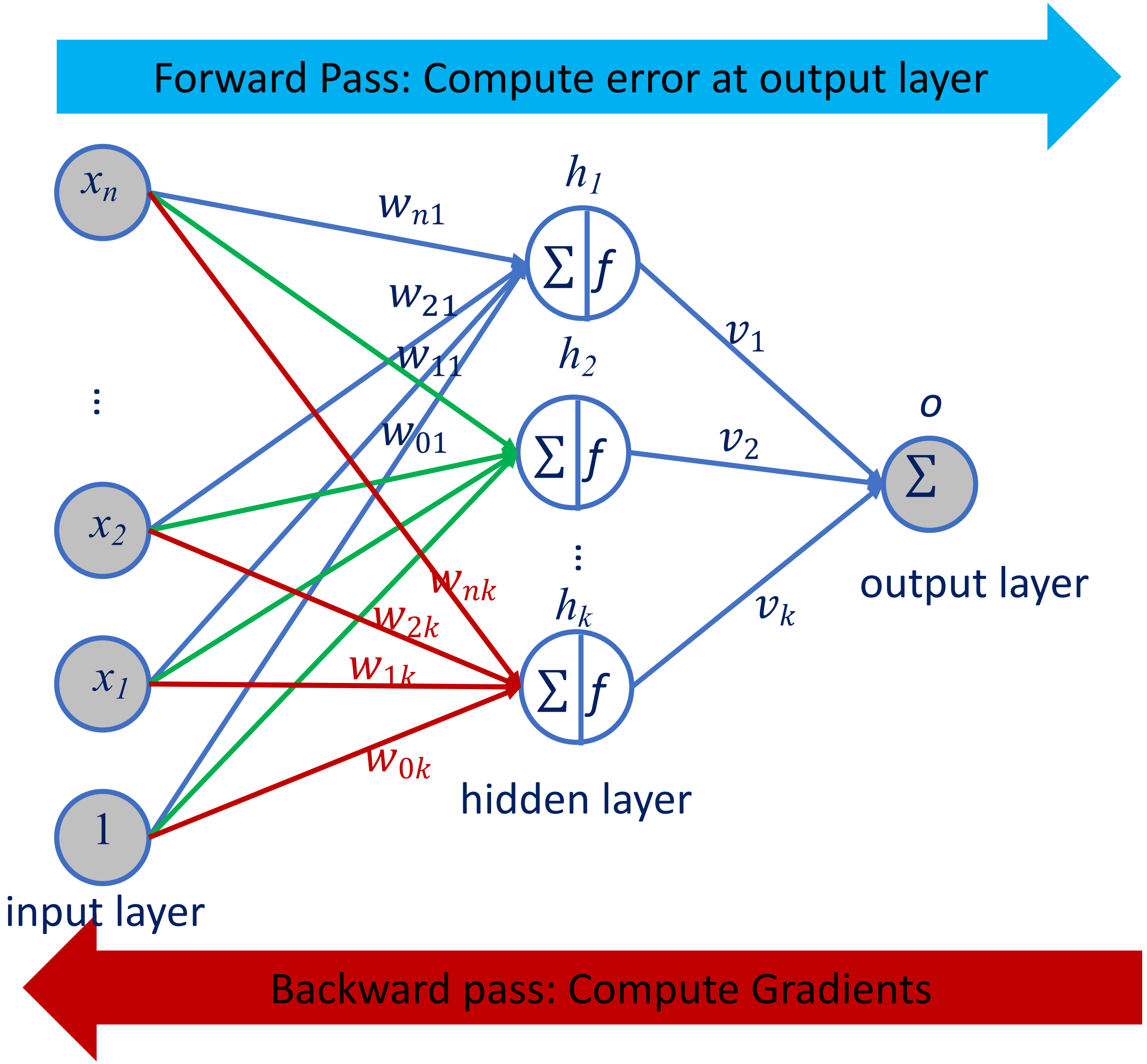
$$\min_{W,\mathbf{v}} \frac{1}{2m} \sum_{i=1}^{m} (y_i - o_i)^2$$

$$\min_{W,\mathbf{v}} \frac{1}{2m} \sum_{i=1}^{m} \left(y_i - \mathbf{v}^T f(W^T \mathbf{x}_i)\right)^2$$

$$= \min_{W,\mathbf{v}} \frac{1}{2m} \sum_{i=1}^{m} \left(y_i - \sum_{j=1}^{k} v_j f(\mathbf{w}_j^T \mathbf{x}_i) \right)^2$$

Note: We use square loss here for simply illustrating the key idea of learning MLP. In practice, we usually use hinge loss or cross entropy loss (log loss) for classification problem.

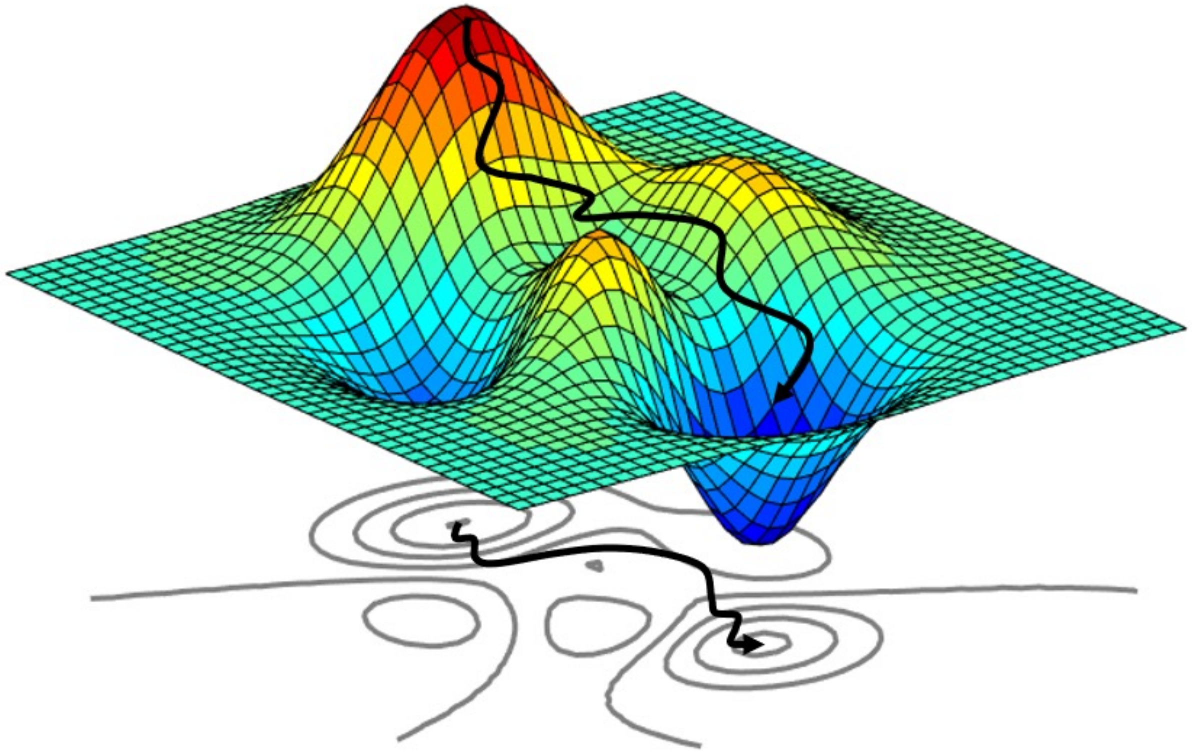
Backpropagation



- Basically consists of a forward pass and a backward pass
- Forward pass computes the error e_i using the current parameters
- Backwards pass computes the gradient and updates the parameters, starting from the parameter at the rightmost layer and the moving backwards.
- Also good at reusing previous computations (updates of parameters at any layer depend on parameters of the upper layer)

Solutions are Local Minima

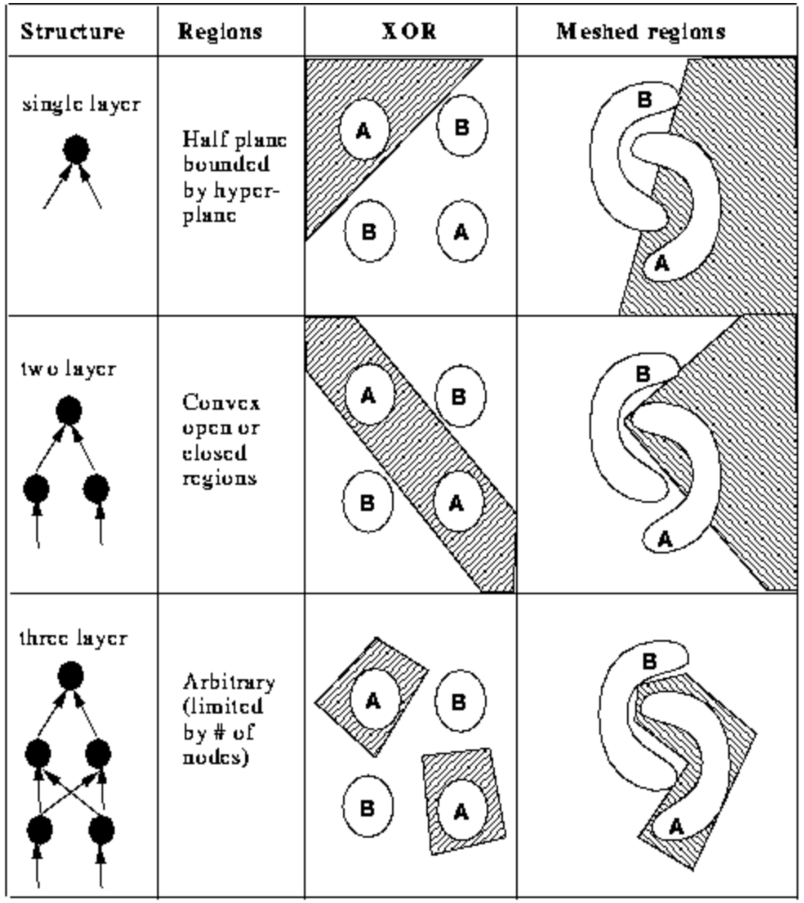
Parameter Initialization and Learning Rate Do Matter



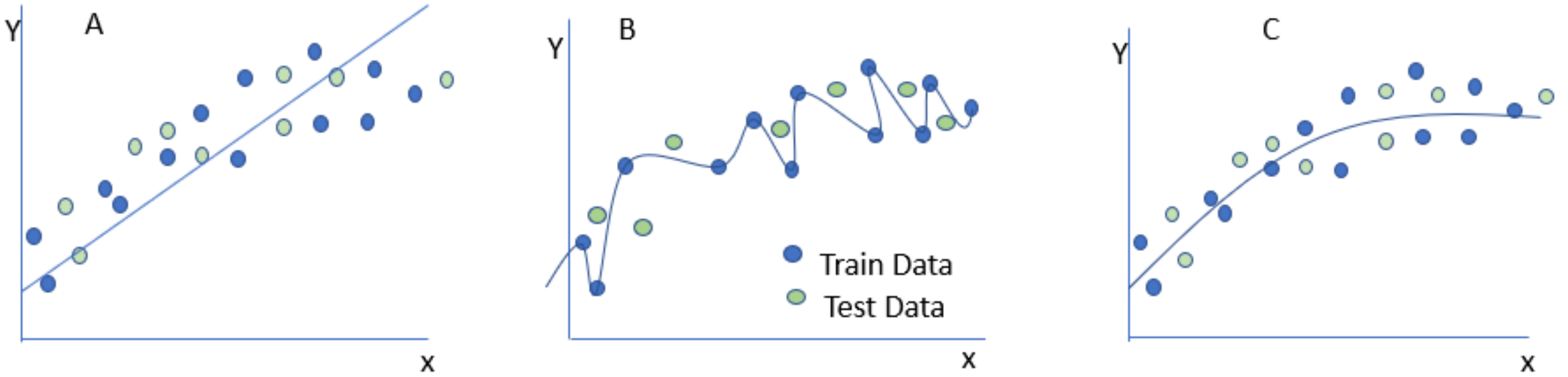
Pros and Cons of MLP

- Pros
  - Versatile: Adaptive to many datasets
  - Can capture nonlinear dependence of input and output

- Cons
  - Does not work for small data sets
  - Blackbox; Hard to interpret
  - Speed of convergence
  - Local minimum
  - Overfitting issue (how to select the structure; how to achieve good generalization)



Underfitting/Overfitting and Model Complexity



Underfitting Overfitting Well fitting

Outline for Data Preprocessing and Data Mining

- **Data Preprocessing**
- **Supervised learning**

❖ Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

❖ Classification

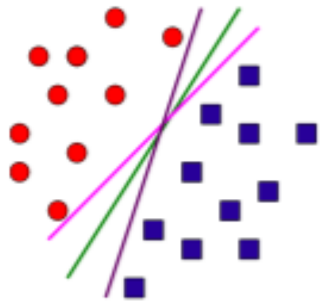
1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

What Is the Best Hyperplane Separator?

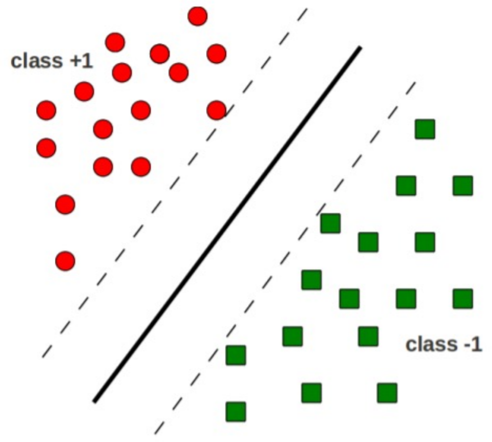
- Perceptron finds one of many possible hyperplanes separating the data
  - If the hyperplane exist
- Among the many possible hyperplanes, which one is the best?



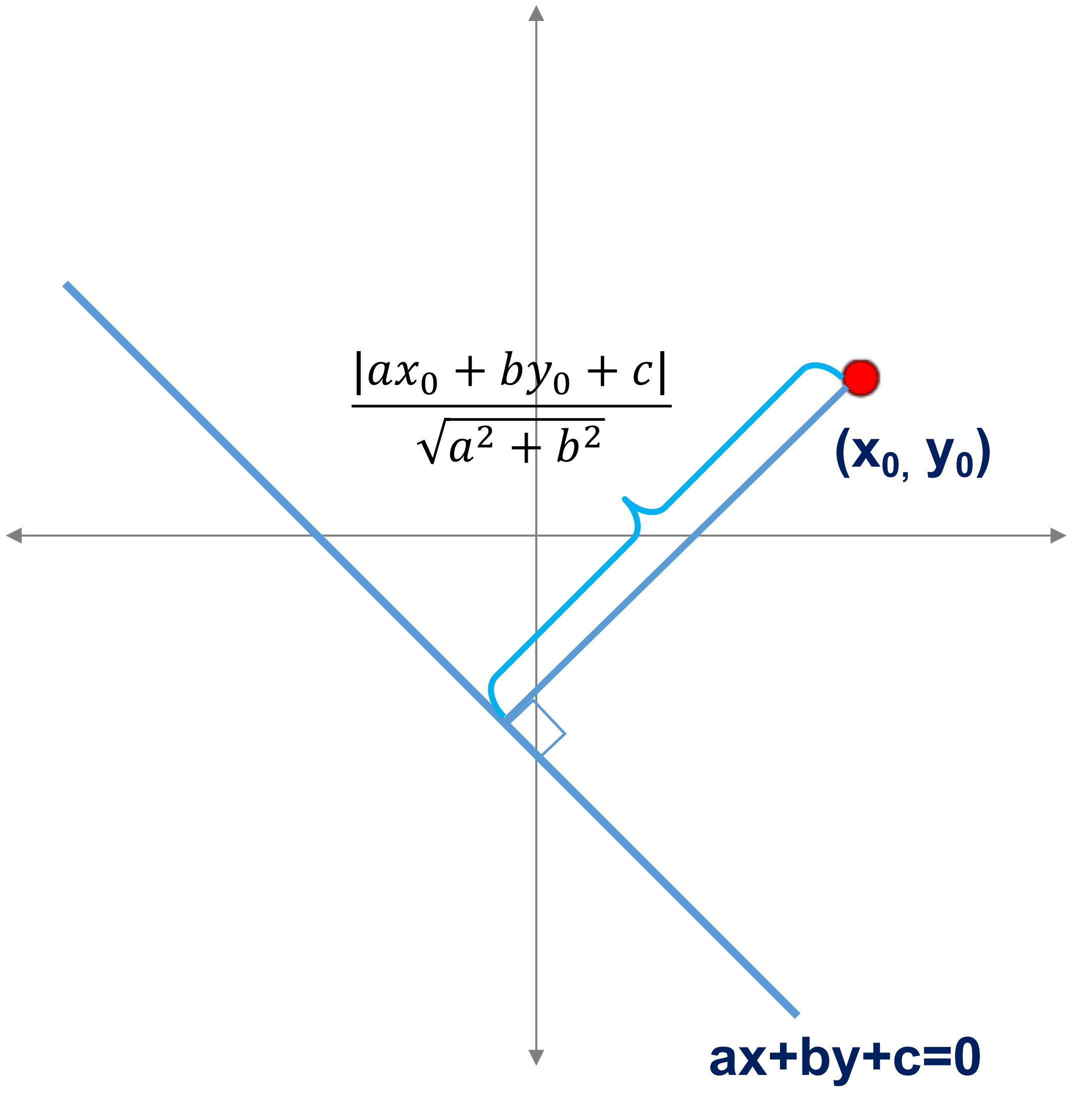
- Intuitively, we want the hyperplane not too close to each of the classes. In other words, the one with the maximum margin is preferred.
- A large margin can lead to good generalization on the test (unseen) data.

Support Vector Machine (SVM)

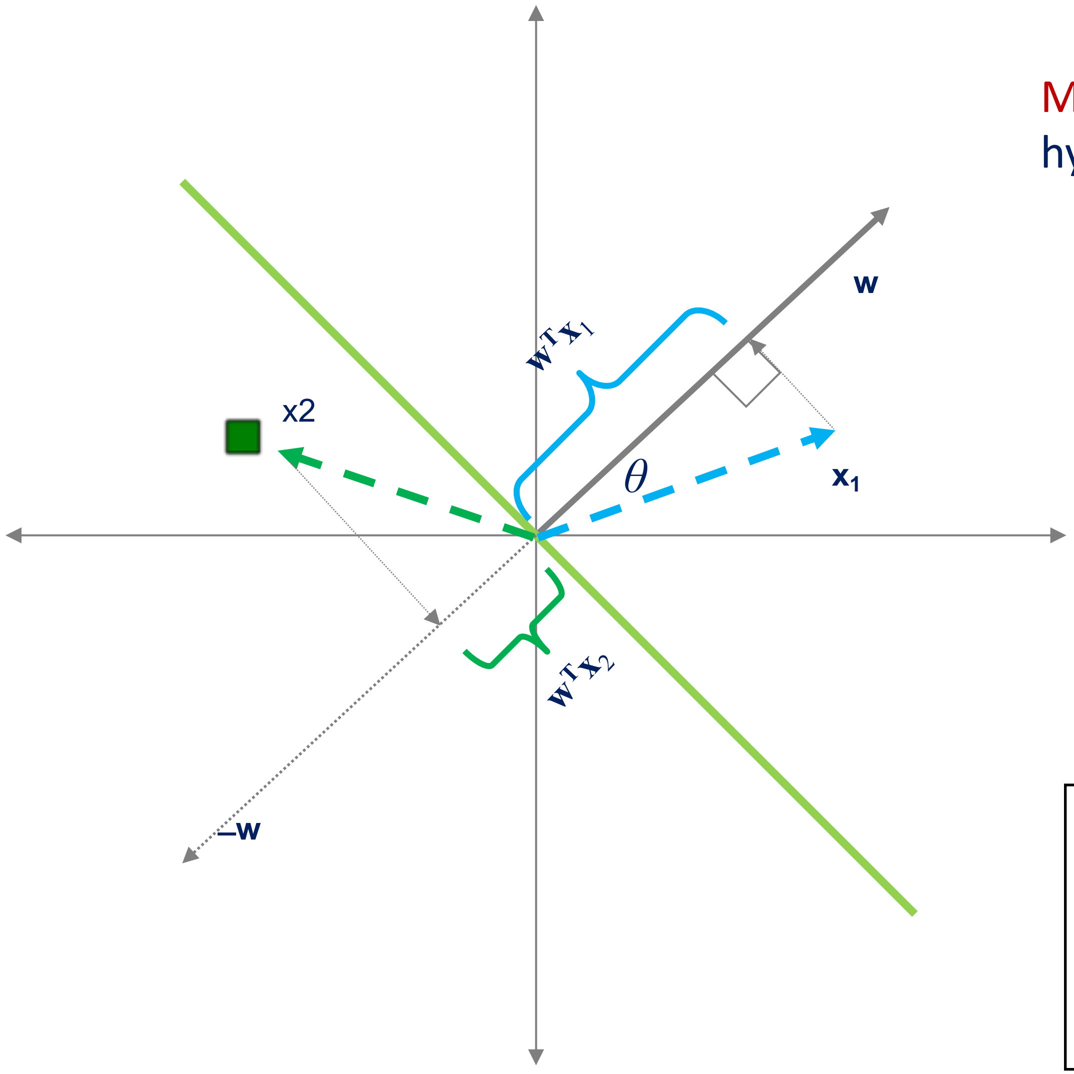
- Probably the most popular/influential classification algorithm
- Backed by solid theoretical groundings
- A hyperplane based classifier (like the Perceptron)
- Additionally uses the maximum margin Principle
    - Finds the hyperplane with maximum separation margin on the training data



Calculate distance from a point to a line



The Concept of Margins



Margin of a sample : γ_i of a sample $\mathbf{x}_i$ is its distance from the hyperplane

$$\|\boldsymbol{w}\| = \sqrt{\sum_{i=1}^{|\boldsymbol{w}|} {w_i}^2} \qquad \|\boldsymbol{w}\|^2 = \sum_{i=1}^{|\boldsymbol{w}|} {w_i}^2$$

$$\gamma_i = \frac{|\mathbf{w}^\mathrm{T}\mathbf{x}_i + b|}{\|\mathbf{w}\|}$$

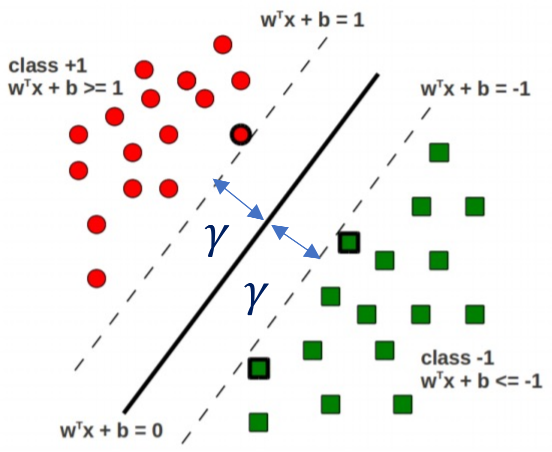
Margin of a set: is the minimum margin of all samples

$$\gamma = \min_{1 \le i \le m} \gamma_i = \min_{1 \le i \le m} \frac{|\mathbf{w}^\mathrm{T}\mathbf{x}_i + b|}{\|\mathbf{w}\|}$$

If we are asked to move the hyperplane to achieve better generalization, what should we do?

Support Vector Machine

- A hyperplane based linear classifier defined by **w** and b
- Prediction rule: $y = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x} + b)$
- **Given:** Training data $\{\mathbf{x}_i, y_i\}_{i=1}^{m}$
- **Goal:** Learn **w** and b that achieve the maximum margin
- For now, assume the entire training data is linearly separable. We will handle linearly unseparable cases later



- Assume the hyperplane is such that
  - $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \geq 1$ for $y_i = +1$
  - $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b \leq -1$ for $y_i = -1$
  - Equivalently, $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) \geq 1$
  - The hyperplane's margin:

$$\min_{1 \leq i \leq m} |\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b| = 1$$

$$\gamma = \min_{1 \leq i \leq m} \gamma_i = \min_{1 \leq i \leq m} \frac{|\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$$

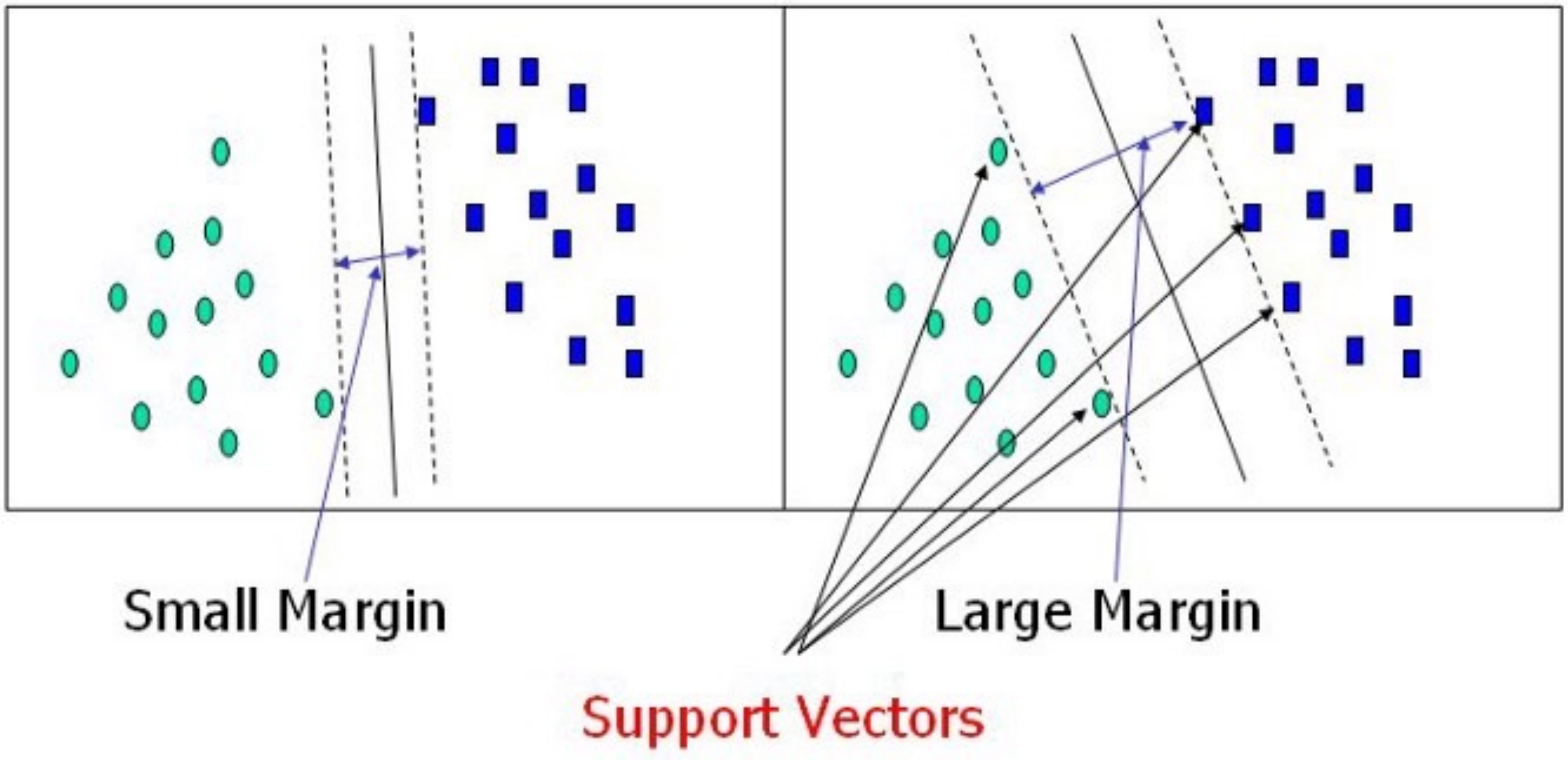
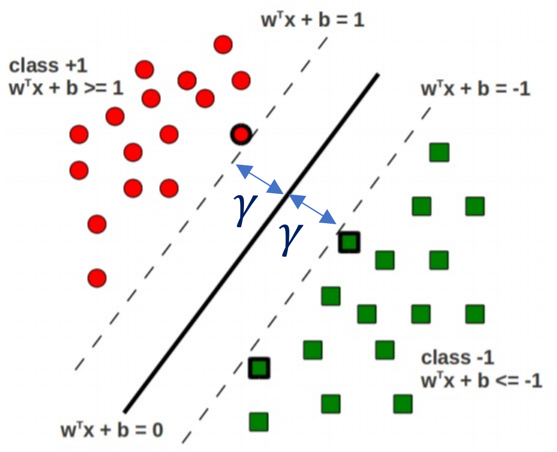


image credit: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

Support Vector Machine: the optimization problem

- We want to maximize the margin $\gamma = \frac{1}{\|\mathbf{w}\|}$, which is equivalent to minimize $\|\mathbf{w}\|$ or $\frac{\|\mathbf{w}\|^2}{2}$



- Therefore, the optimization problem of SVM for the *separable case* would be

$$\min \frac{\|\mathbf{w}\|^2}{2}$$
$$\text{subject to} \quad y_i(\mathbf{w}^{\mathbf{T}}\mathbf{x}_i + b) \geq 1, i = 1, \dots, m$$

This is a Quadratic Program (QP) with *n* linear inequality constraints.

SVM: Solving the Optimization Problem (optional)

- The optimization problem is

$$\min \frac{\|\mathbf{w}\|^2}{2}$$
$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, i = 1, \ldots, m$$

- Introducing Lagrange Multipliers α_i, one for each constraint, leads to the primal Lagrangian:

$$\min L_p = \frac{\|\mathbf{w}\|^2}{2} + \sum_{i=1}^{m} \alpha_i(1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$$
$$\text{subject to} \quad \alpha_i \geq 0, i = 1, \ldots, m$$

SVM: Solution! (optional)

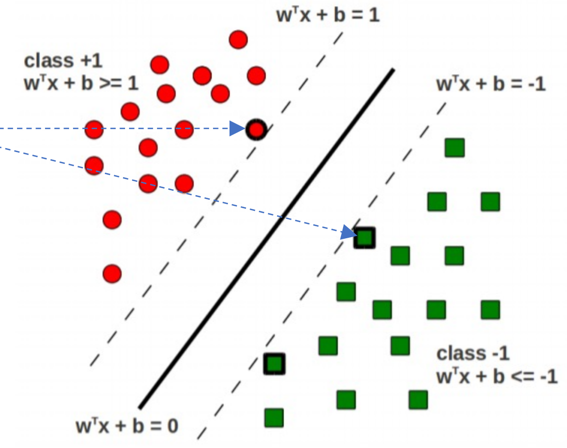
- Once we have the α_i, we can compute $\boldsymbol{w}$ and b as:

(duality optimization, KTT, Sequential Minimal Optimization)

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i\, y_i \mathbf{x}_i \qquad b = y_i - \sum_{j=1}^{m} \alpha_j\, y_j \mathbf{x}_j^T \mathbf{x}_i$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b = y_i$$

any support vector will work



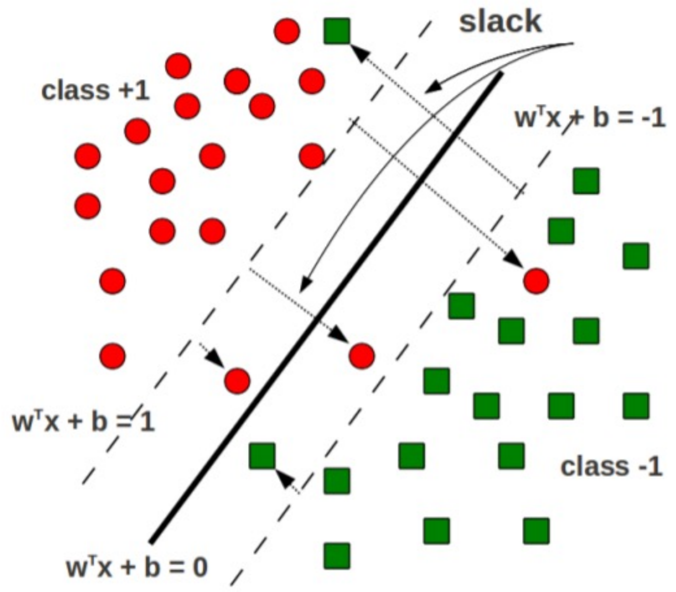
- An important consequence:
  - α_i is non-zero only if $\mathbf{x}_i$ lies on one of the two margin boundaries, i.e., for which $$y_i(\mathbf{w}^T\mathbf{x}_i + b) = 1$$
  - The samples are called **support vectors**.

SVM: Non-separable case

- Non-separable case: No hyperplane can separate the classes perfectly (common in practice).

- Still want to find the maximum margin hyperplane, but…
  - Allow some training samples to be misclassified (can you identify those points on the right?)

  - Allow some training samples to fall within the margin region (can you identify those points on the right?)



SVM: Use Slack Variables

- Solution: introduce slack variables

- Recall: for the separable case, the constraints are:

$$y_i(\mathbf{w}^\mathrm{T}\mathbf{x}_i + b) \geq 1, i = 1, \dots, m$$

- For the non-separable case, we relax the constraints by adding slack variables

$$y_i(\mathbf{w}^\mathrm{T}\mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \dots, m$$

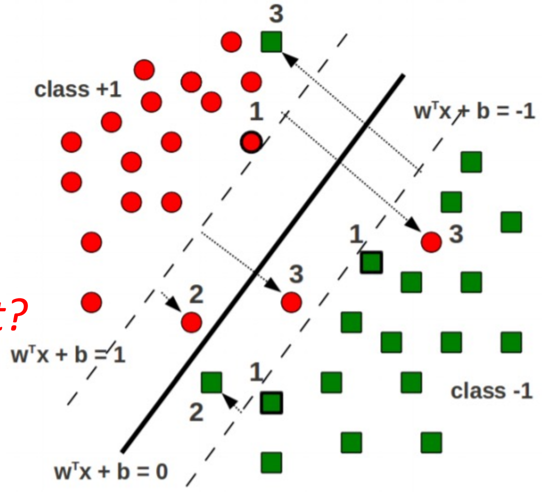
- $\xi_i \geq 0$ is required
- $\xi_i > 1$ for misclassified samples (for outside the margin)

Support Vectors for non-separable cases

- Recall: the separable case has only one type of support vectors
  - Ones that lies on the margin boundaries $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b = 1$ or $\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b = -1$

- The non-separable case has three types of support vectors
  1. Lying on the margin boundaries $\xi_i = 0$
  2. Lying with the margin region $0 < \xi_i < 1$ but still on the correct side
  3. Lying on the wrong side the hyperplane $\xi_i \geq 1$

Can you identify them on the right?



The Optimization Problem (optional)

- While we "allow" misclassified training samples
  - We want the number of misclassified training samples to be minimized
  - By minimizing the sum of slack variables $\sum_{i=1}^{m} \xi_i$

- Therefore, the new optimization problem for non-separable case will be

$$\min \frac{\|\mathbf{w}\|^2}{2} + \boldsymbol{C} \sum_{i=1}^{\mathrm{m}} \xi_i$$

$$\text{subject to} \quad y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, i = 1, \dots, m$$

$$\xi_i \geq 0$$

C is a hyper-parameter to control the tradeoff between training errors and margins

- Large C, prefer low training errors
- Small C, prefer large margins

The Optimization Problem (optional)

- As in the linearly separable problem, by following the derivations, we will obtain the following dual problem

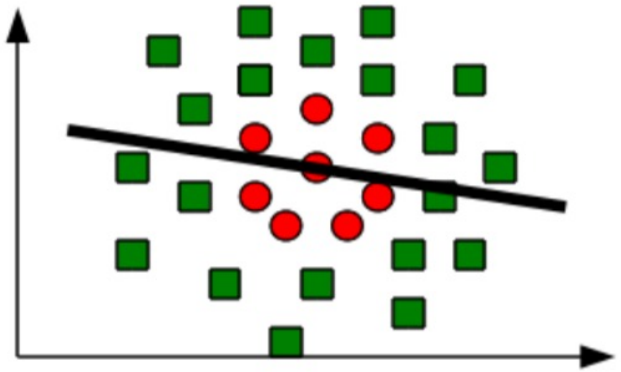
$$\max \; L_d = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i \, y_i = 0, 0 \leq \alpha_i \leq C, i = 1, \ldots, m$$

- Again a Quadratic Programming problem for α
- Given $\boldsymbol{\alpha}$, the solution $\mathbf{w}$, b has the same form as the separable case
- Note: $\boldsymbol{\alpha}$ is again sparse. Non-zero α_i corresponds to the **support vectors**

SVM for Nonlinear Classification

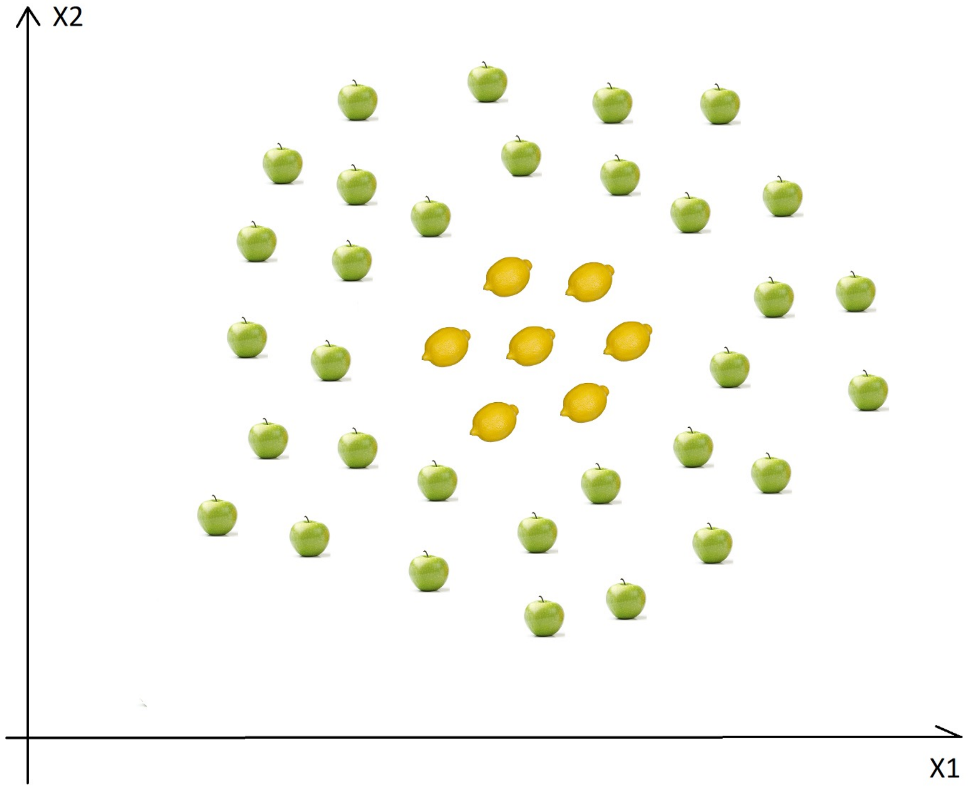
- Problem: SVM with linear function $\mathbf{w}^{\mathrm{T}}\mathbf{x} + b$ have very limited representation power. Therefore, it can not solve nonlinear classification problem.

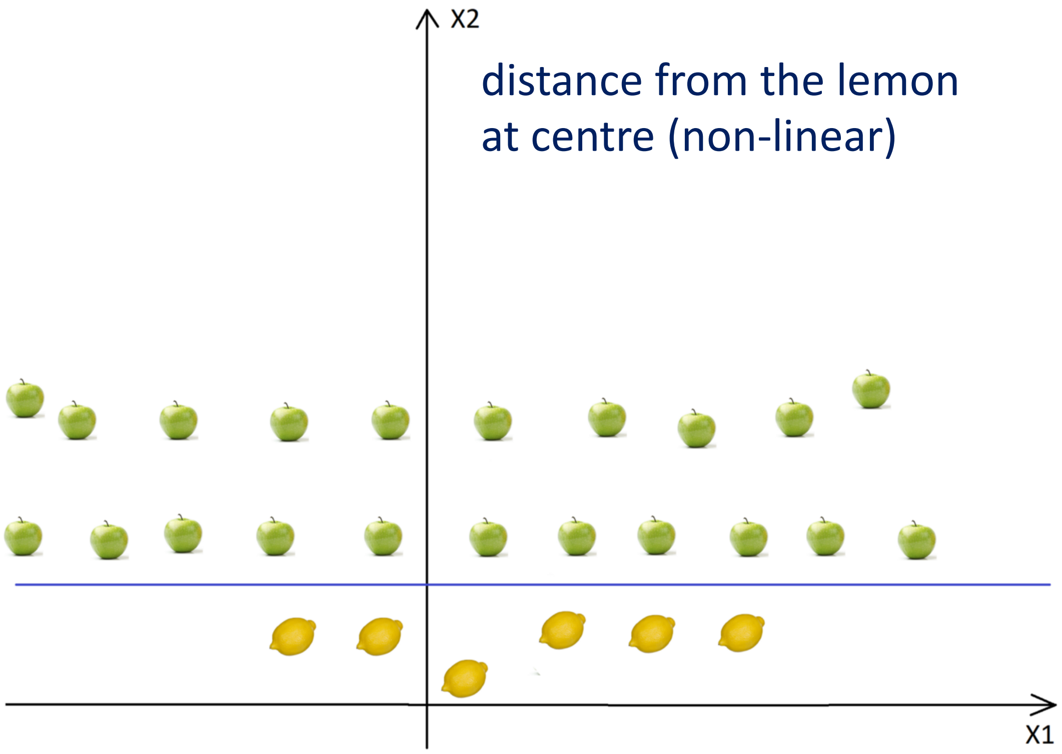


- Good news: With a slight modification using **kernel trick**, SVM can solve highly nonlinear classification problems.

Kernel SVM for Nonlinear Classification

- Key idea: Projecting the input to a high dimensional feature space so that non-linear classification problem becomes linearly separable again!

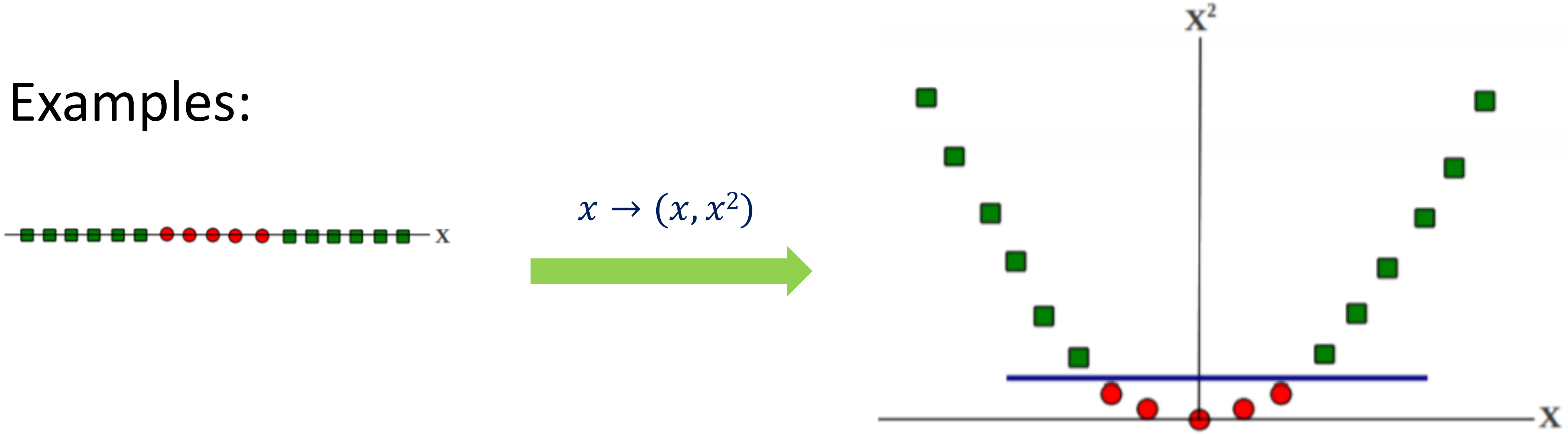




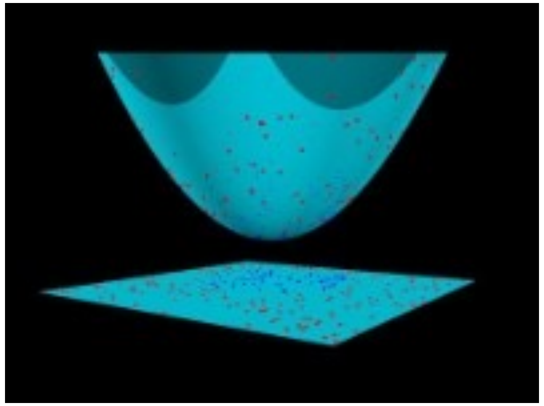
Kernels

- Kernels: Make linear models works in nonlinear settings
  - By mapping data to higher dimensions where it exhibits linear patterns.
  - Apply the linear model in the new input space
  - Mapping means changing the feature representation

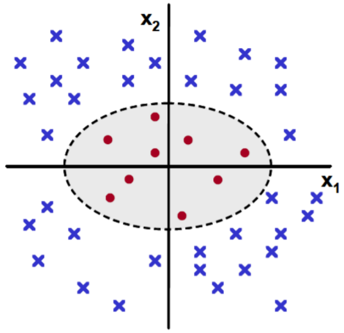
- Examples:



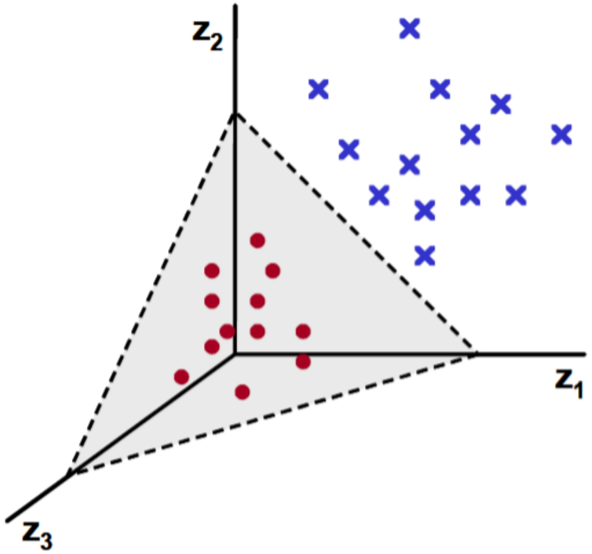
Kernels



- Kernels: Make linear models works in nonlinear settings
  - By mapping data to higher dimensions where it exhibits linear patterns.
  - Apply the linear model in the new input space
  - Mapping means changing the feature representation
  - https://www.youtube.com/watch?v=3liCbRZPrZA
- Examples:



$\mathbf{x}$: $[x_1, x_2] \rightarrow \mathbf{z}$: $[x_1^2, \sqrt{2}x_1x_2, x_2^2]$



Feature Mapping (optional)

- Consider the following mapping ϕ for a sample $\mathbf{x} = [x_1, x_2, \dots, x_n]$

$$\phi\colon\ \mathbf{x} \to [x_1^2, x_2^2, \dots, x_n^2, x_1x_2, x_1x_3 \dots, x_1x_n, \dots, x_{n-1}x_n]$$

- It is an example of quadratic mapping
  - Each new feature uses a pair of the original features

- Problem: Explicit mapping leads to the number of features blow up!

- Fortunately, kernel trick help us to avoid the problem.
  - The mapping does not have to be explicitly computed

Kernel as high dimensional feature mapping (optional)

- Consider two samples $\mathbf{x}$: $[x_1, x_2]$ and $\mathbf{z}$: $[z_1, z_2]$
- Let us assume there is a kernel function *k* that takes inputs **x** and **z**

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{xz})^2 \\
&= (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 x_2 z_1 z_2 \\
&= \left(x_1^2, \sqrt{2}x_1 x_2, x_2^2\right)\left(z_1^2, \sqrt{2}z_1 z_2, z_2^2\right) \\
&= \phi(\mathbf{x})\phi(\mathbf{z})
\end{aligned}
$$

$\phi(\mathbf{x})\phi(\mathbf{z})$ is computed efficiently in original input space.

- This kernel function k implicitly defines a mapping ϕ to a higher dimensional space

$$\phi(\mathbf{x}) = \left[x_1^2, \sqrt{2}x_1 x_2, x_2^2\right]$$

- Note that we do not have to define/compute this mapping. Simple defining the kernel is a certain way to give a higher dimensional mapping ϕ

Kernel: Formal definition (optional)

- ϕ takes input **x** in input space and maps to feature space

- Kernel $k(\mathbf{x}, \mathbf{z})$ takes two inputs and gives their similarity in feature space

$$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})\phi(\mathbf{z})$$

- Some Examples of Kernels
  - Linear kernel $k(\mathbf{x}, \mathbf{z}) = \mathbf{xz}$
  - Quadratic Kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz})^2$ or $k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{xz})^2$
  - Polynomial Kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{xz})^q$ or $k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{xz})^q$
  - Radial Basis Function (RBF) kernel $k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma\|\mathbf{x} - \mathbf{z}\|^2)$
    - The RBF kernel corresponds to an infinite dimensional feature space. We can not actually write down $\phi(\mathbf{x})$ for RBF kernel.

Using Kernel

- Kernel can turn a linear model into a nonlinear one
- Recall: Kernel $k(\mathbf{x}, \mathbf{z})$ represents a dot product in some high dimensional feature space.
- Any learning algorithm in which examples only appear as dot products $(\mathbf{x}_i \mathbf{x}_j)$ can be kernelized (i.e., non-linearized)
  - by replacing the $(\mathbf{x}_i \mathbf{x}_j)$ by $\phi(\mathbf{x}_i)\phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$
- Most learning algorithms can be kernelized:
  - Perceptron, SVM, linear regression, logistic regression, etc

Kernelize SVM Training (optional)

- Recall the dual Lagrangian for linear SVM

$$\max L_d = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_\mathrm{i}\mathbf{x}_j)$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i\, y_i = 0, 0 \le \alpha_i \le C, i = 1, \dots, m$$

- Replace $(\mathbf{x}_i\mathbf{x}_j)$ by $\phi(\mathbf{x}_i)\phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$, where $k(,)$ is some kernel function

$$\max L_d = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \quad \sum_{i=1}^{m} \alpha_i\, y_i = 0, 0 \le \alpha_i \le C, i = 1, \dots, m$$

- Now, SVM learns a linear separate in kernel defined feature space
  - This corresponds to non-linear separator in the original input space

Kernel SVM for Nonlinear Classification (optional)

- For a new input $\boldsymbol{x}$, the output will be:

$$y = \mathbf{w}^{T}\mathbf{x} = \sum_{i=1}^{m} \alpha_i \, y_i \mathbf{x}_i^{T} \, \mathbf{x}$$

Since $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \, y_i \mathbf{x}_i$

- For the non-linear classification, apply non-linear transformation $\Phi(\mathbf{x})$ to project $\mathbf{x}$ to a higher dimensional space and the inner product term becomes

$$y = \sum_{i=1}^{m} \alpha_i \, y_i \boldsymbol{\Phi}(\boldsymbol{x}_i)^{T} \boldsymbol{\Phi}(\boldsymbol{x})$$

Kernel SVM for Nonlinear Classification (optional)

- As only the inner product is needed, we can apply the kernel trick. That is we care only the the way to measure distance between two points.

$$y = \sum_{i=1}^{m} \alpha_i \, y_i \boldsymbol{\Phi}(x_i)^T \boldsymbol{\Phi}(x)$$

$$y = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{m} \alpha_i \, y_i k(\mathbf{x}_i, \mathbf{x})$$

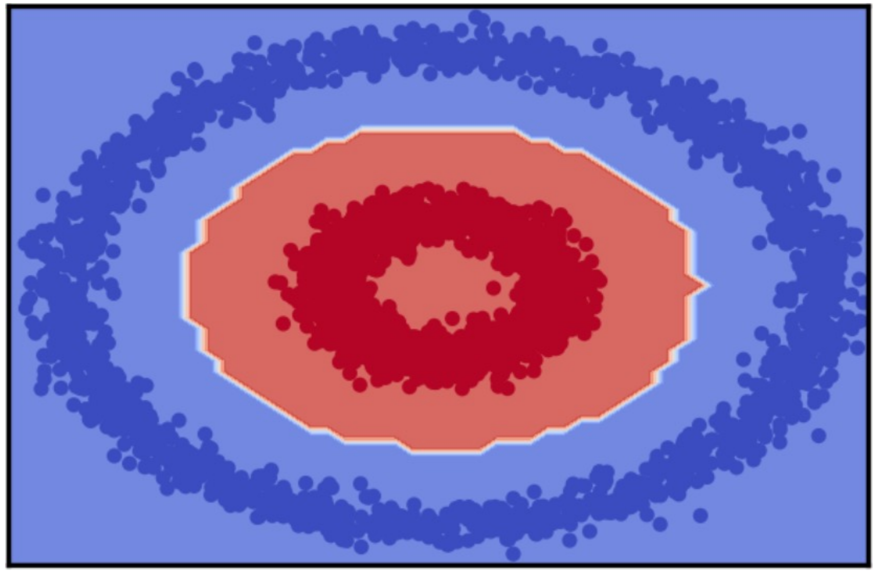
Note: We do not need to explicitly compute $\mathbf{w}$ and $\phi(\mathbf{x})$ for kernel SVM.

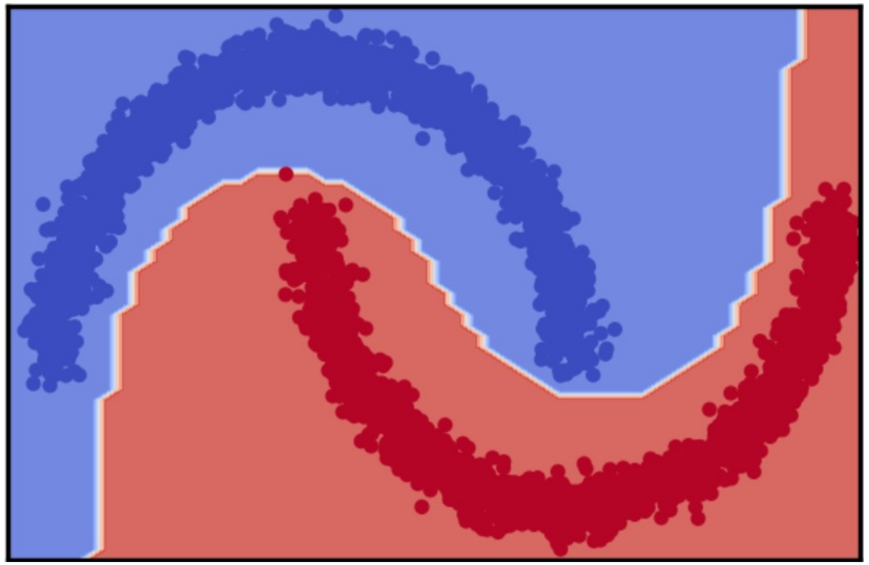
- One common kernel: Radial Basis Function (RBF)

$$k(\mathbf{x}_i, \mathbf{x}) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}\|)$$

Gamma is for determining how the distance is considered influential.

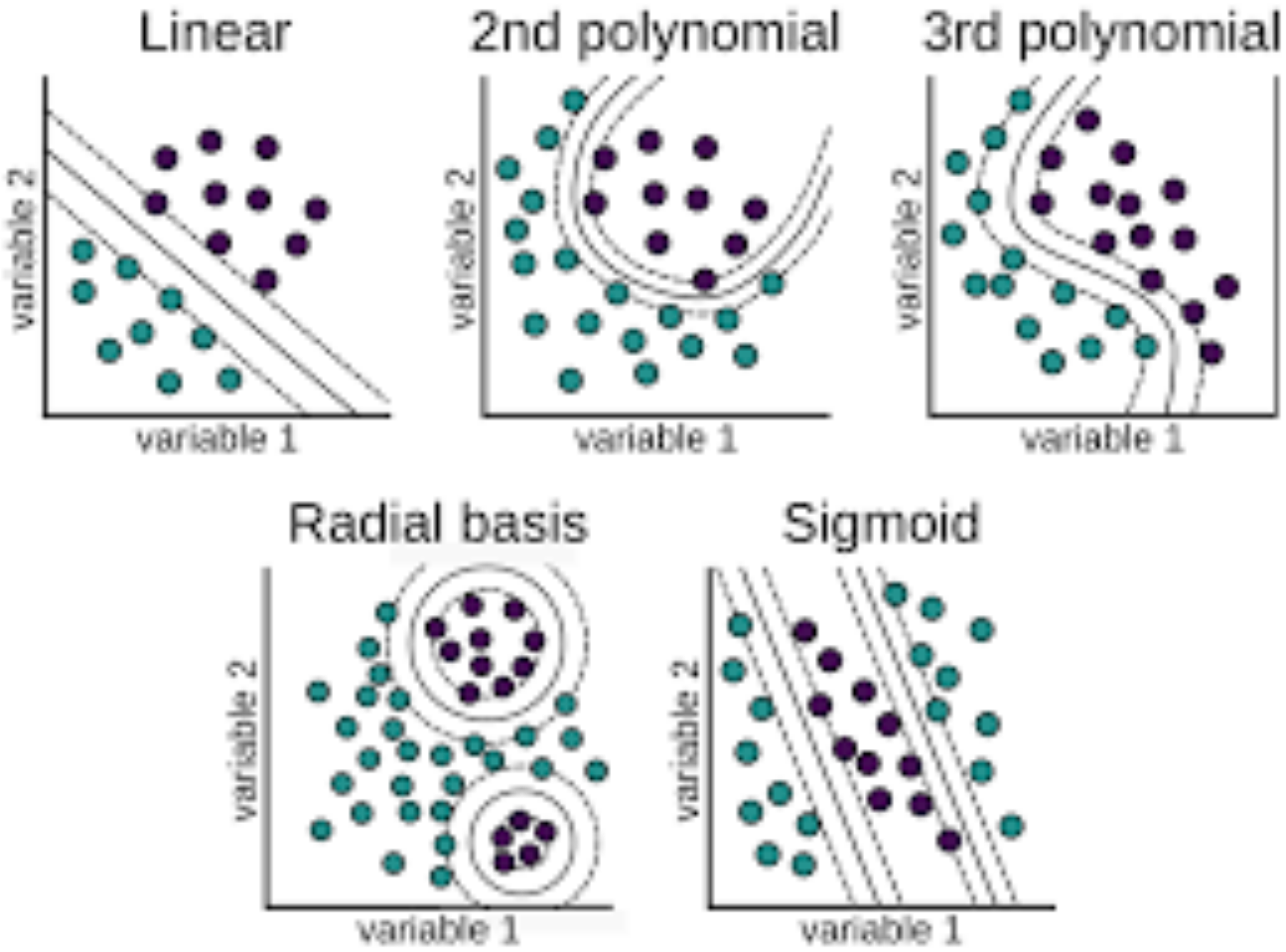
SVM with RBF kernel





The learned decision boundary by SVM with RBF kernel is nonlinear in the original space

SVM with Different Kernels and Decision Boundaries



Outline for Data Preprocessing and Data Mining

- **Data Preprocessing**
- **Supervised learning**

❖Regression

1. Linear regression with one variable
2. Linear Regression with multiple variables
3. The relationship between Correlation and Regression

❖Classification

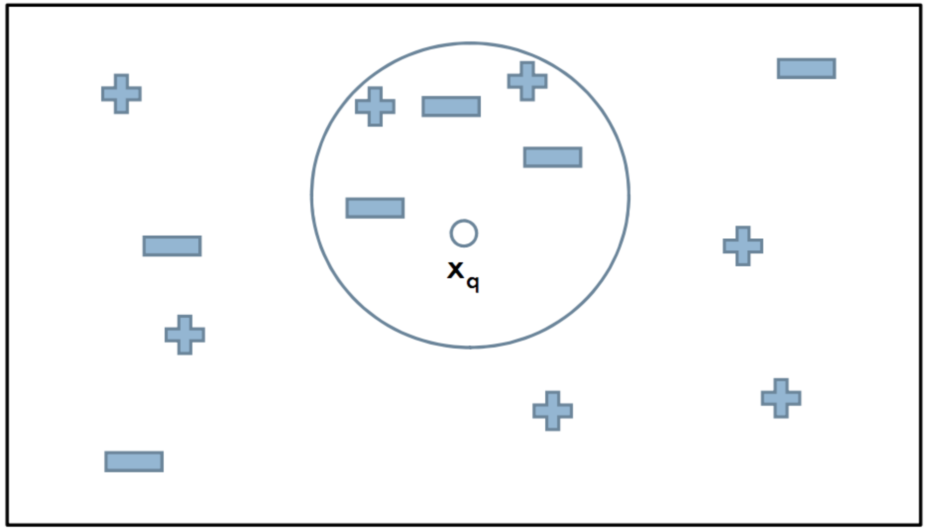
1. Perceptron
2. Artificial Neural Network
3. Support Vector Machine
4. K Nearest Neighbor

- **Unsupervised learning**

1. K-means Clustering
2. Hierarchical Clustering

k-NN Algorithm

- k Nearest Neighbor Algorithm
  - k is a user specified parameter, which means the number of nearest neighbors
- A Lazy Learning Algorithm
  - **Training:**
    - **No training process**, just store all training data in memory
  - **Prediction:**
    - Classify new samples based on most similar training samples via majority vote

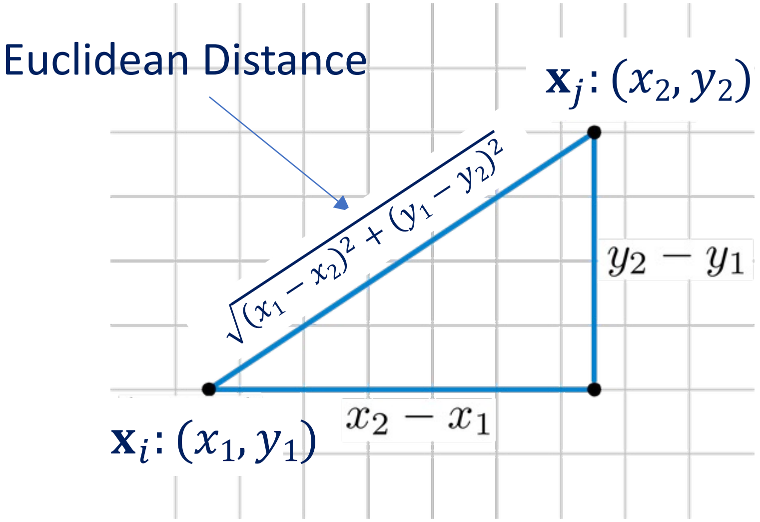


- $\mathbf{x}_q$ is the test sample.
- Assume k is equal to 5.
- Three out of its 5 nearest neighbors are from negative class.
- The predicted label for $\mathbf{x}_q$ is negative.

Nearest Neighbors

- Training Data {**X**, **y**}
- A test data point: $\mathbf{x}_{\text{test}}$
- Idea: the label of a test data point is estimated from the known label(s) of the nearest neighbors of $\mathbf{x}_{\text{test}}$ in the training data.
- Euclidean distance between feature vectors can be used to decide the nearest neighbors

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2 = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2}$$

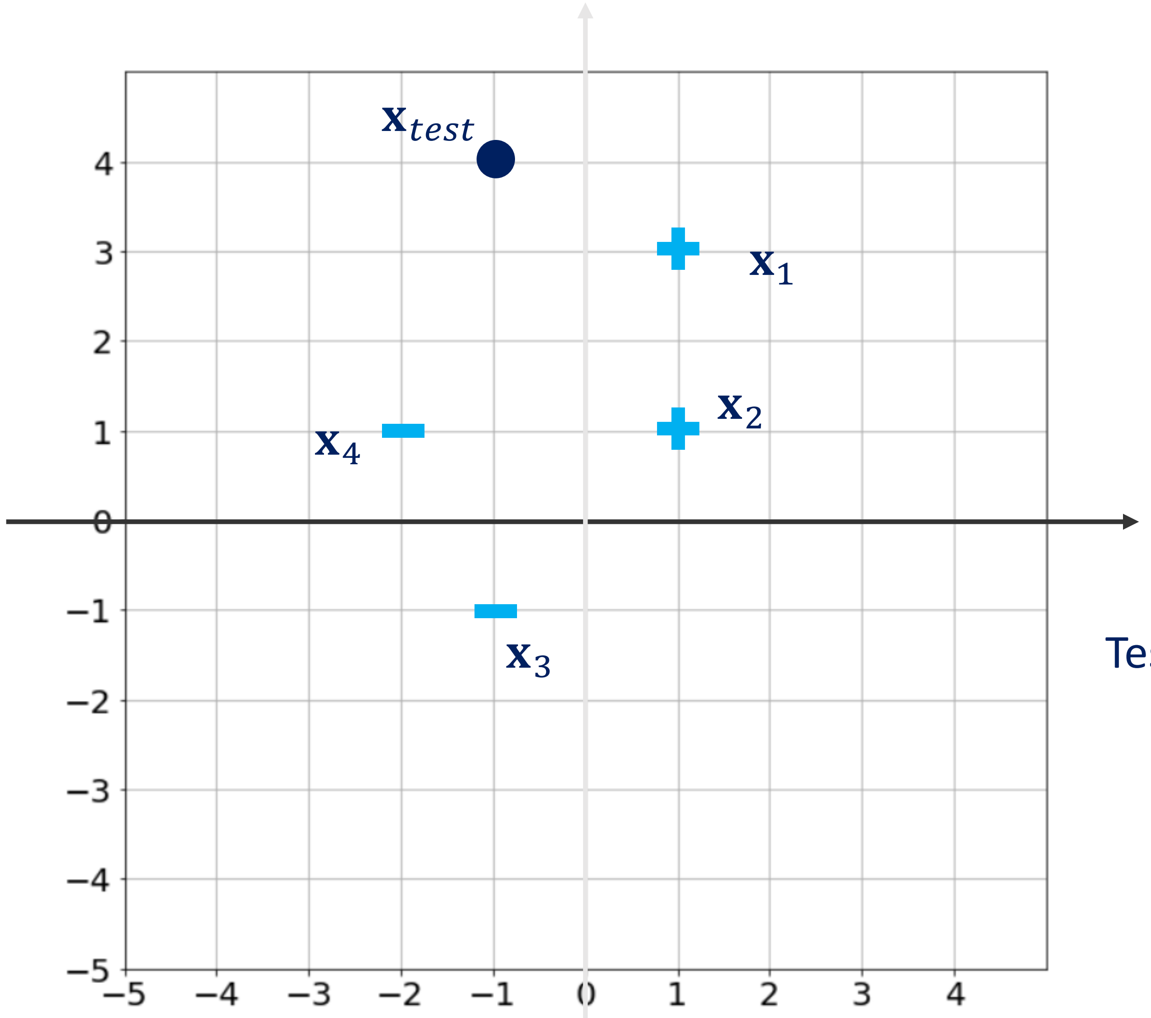


k -NN algorithm

- Input: training data {**X**, y}, a test data sample $\mathbf{x}_{\text{test}}$, parameter *k*
  - Compute the distances between the test sample $\mathbf{x}_{\text{test}}$ and each training data sample
  - Sort by distances and get the *k* nearest neighbors of $\mathbf{x}_{\text{test}}$ (*k* is usually set to an odd number to prevent tie situations)
  - Use majority vote to predict the class label of $\mathbf{x}_{\text{test}}$

- Output: predicted class label of $\mathbf{x}_{\text{test}}$

k-Nearest Neighbors Example (k=3)



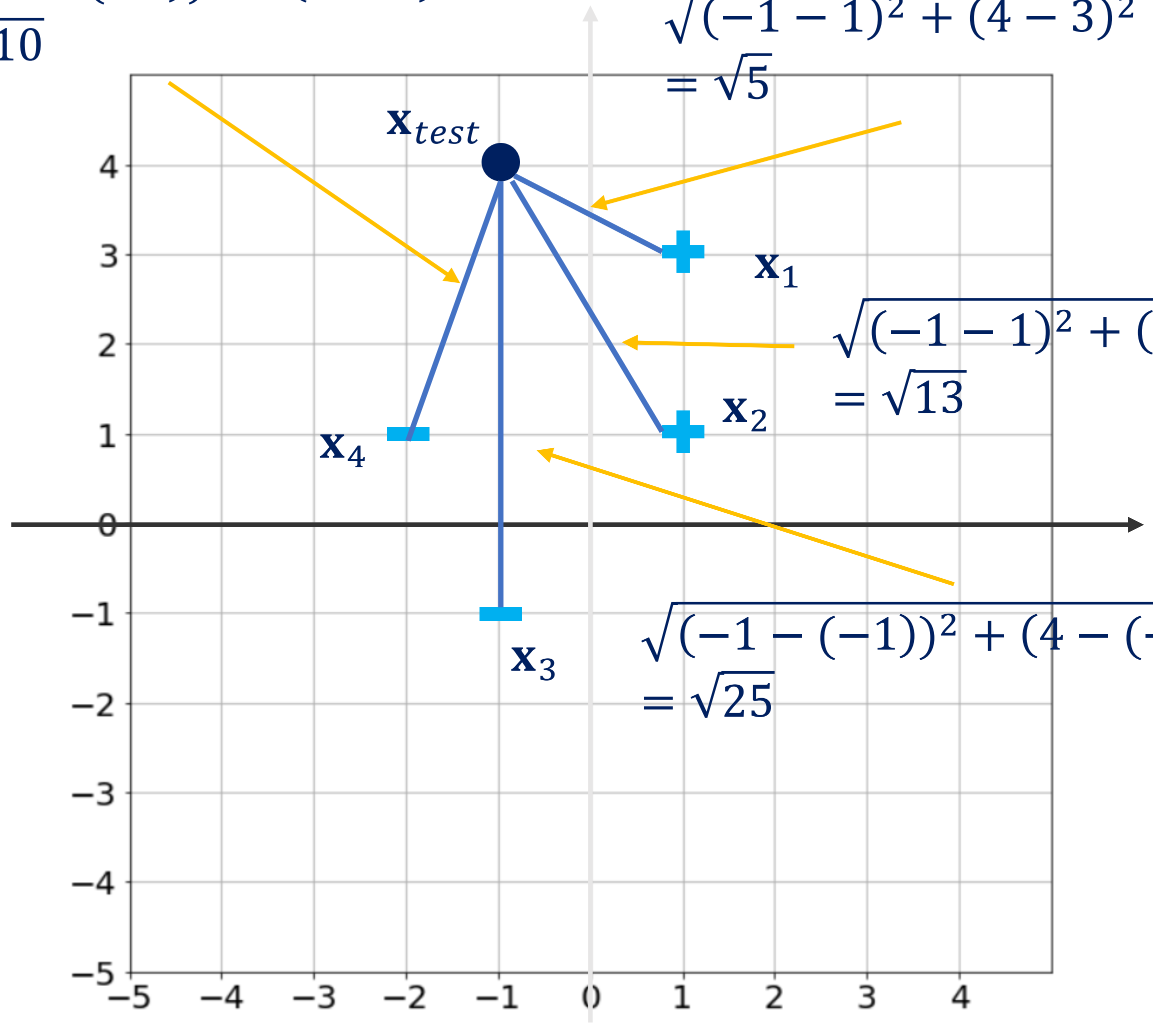
Training Data:

x_1	x_2	*label*
1	3	1
1	1	1
-1	-1	-1
-2	1	-1

Test Data:

x_1	x_2	*label*
-1	4	?

k-Nearest Neighbors Example (k=3)



$$\sqrt{(-1-(-2))^2+(4-1)^2} = \sqrt{10}$$

$$\sqrt{(-1-1)^2+(4-3)^2} = \sqrt{5}$$

$$\sqrt{(-1-1)^2+(4-1)^2} = \sqrt{13}$$

$$\sqrt{(-1-(-1))^2+(4-(-1))^2} = \sqrt{25}$$

Training Data:

x_1	x_2	*label*
1	3	1
1	1	1
-1	-1	-1
-2	1	-1

Test Data:

x_1	x_2	*label*
-1	4	1

	$\mathbf{x}_{test}$
$\mathbf{x}_1$	$\sqrt{5}$
$\mathbf{x}_2$	$\sqrt{13}$
$\mathbf{x}_3$	$\sqrt{25}$
$\mathbf{x}_4$	$\sqrt{10}$

Sort

	$\mathbf{x}_{test}$
$\mathbf{x}_1$	$\sqrt{5}$
$\mathbf{x}_4$	$\sqrt{10}$
$\mathbf{x}_2$	$\sqrt{13}$
$\mathbf{x}_3$	$\sqrt{25}$

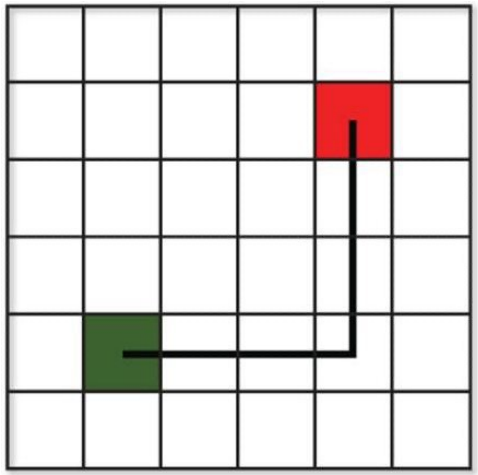
Other k-NN Distance metrics

Minkowski Distance

$$d = (\sum_{i=1}^{m} |x_i - y_i|^p)^{1/p}$$

Manhattan Distance

$$d = \sum_{i=1}^{m} |x_i - y_i|$$



Manhattan Distance

Other k-NN Distance metrics

Cosine Distance

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|\cdot\|\vec{b}\|}$$

Jaccard Distance

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

A = {X, Y, Z} , **B** = {W, X}, C = {X, Y}
J(A,B) = 1 / 4
J(A,C) = 2 / 3

k-NN - Generalize to multiple classes

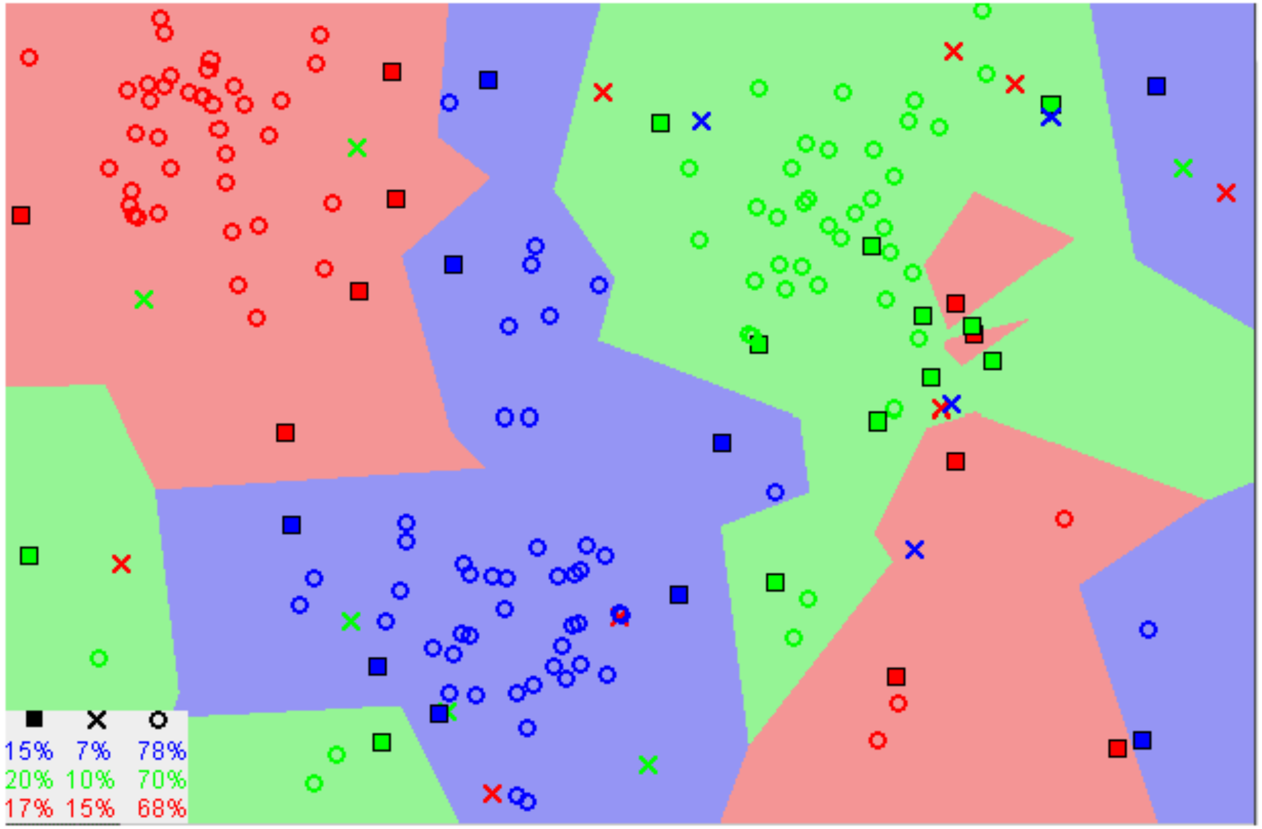


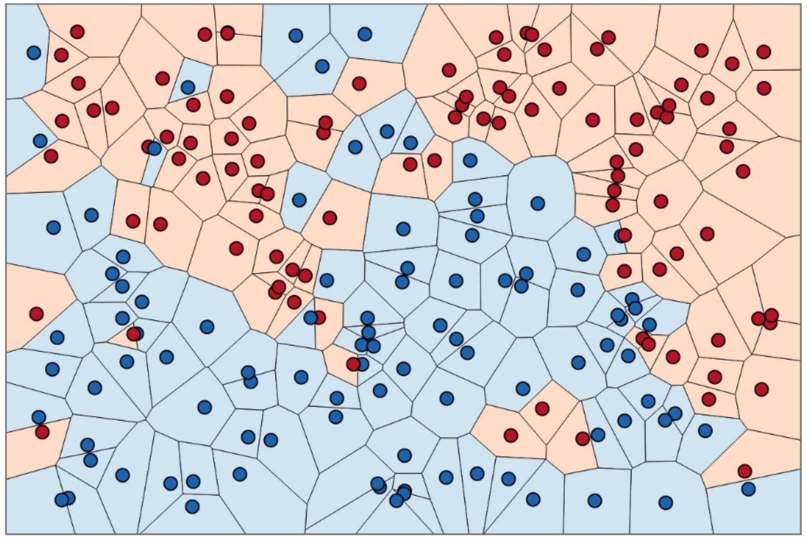
Image showing how similar data points typically exist close to each other

image credit: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

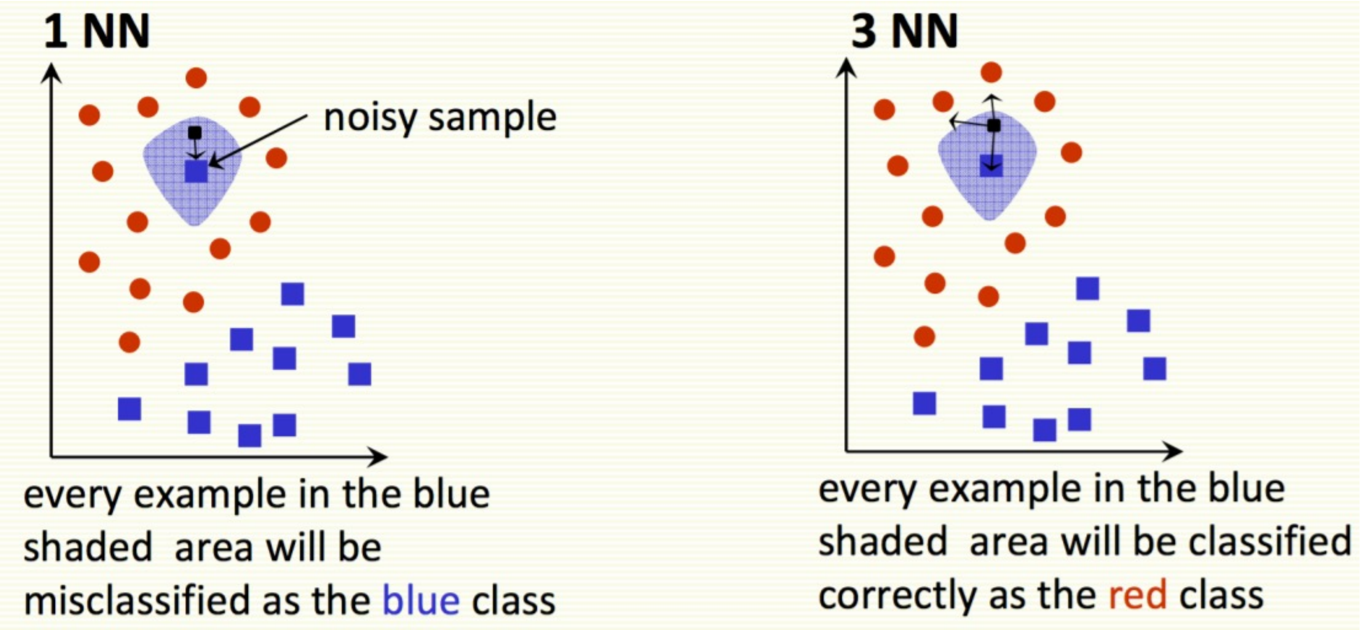
k-NN: Decision Boundaries

- k-NN algorithm does not explicitly compute decision boundaries, but the decision boundaries can be inferred.

- Decision boundaries of 1-NN: Voronoi diagram
  - Show how input space divided into classes
  - Each line segment is equidistant between two neighboring data points.

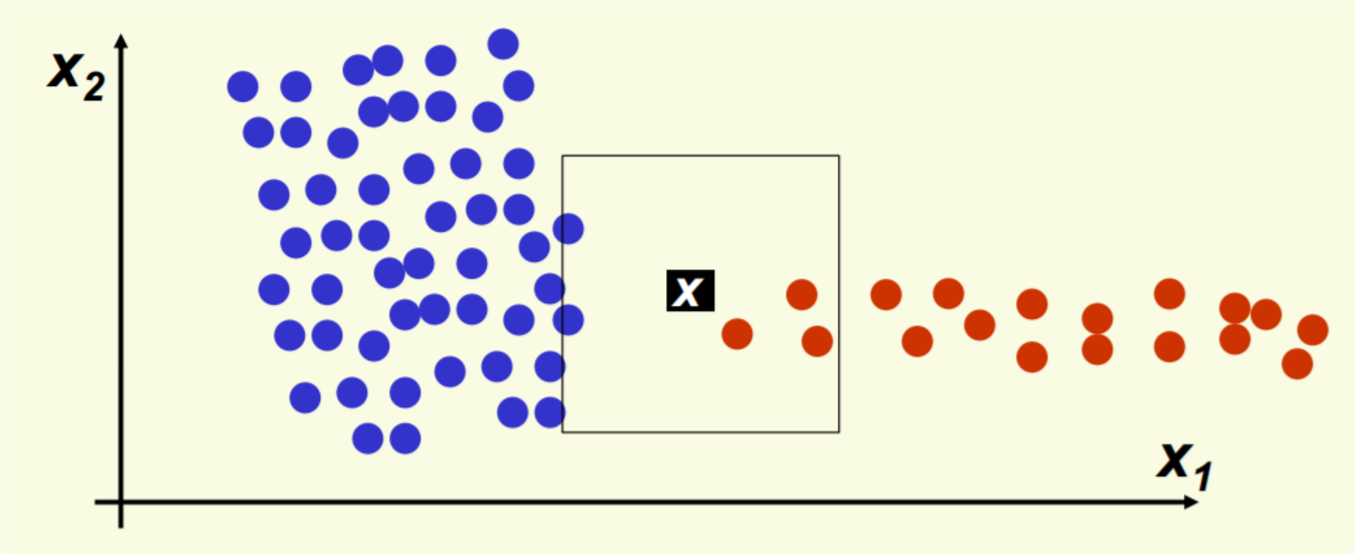


The Effect of k in k-NN



- If k is too small, k-NN will be very sensitive to “noisy samples” and lead to noisy decision boundaries.
- Large k will smooth the decision boundaries and may lead to better performance.

The Effect of k in k-NN



- If k is smaller than 5, data sample **x** is correctly predicted as red class.
- For larger k, data sample **x** is wrongly predicted as blue class.
- Therefore, if k is too large, we may end up with over-smoothed boundaries since it looks neighbors that are far away from the test data sample.

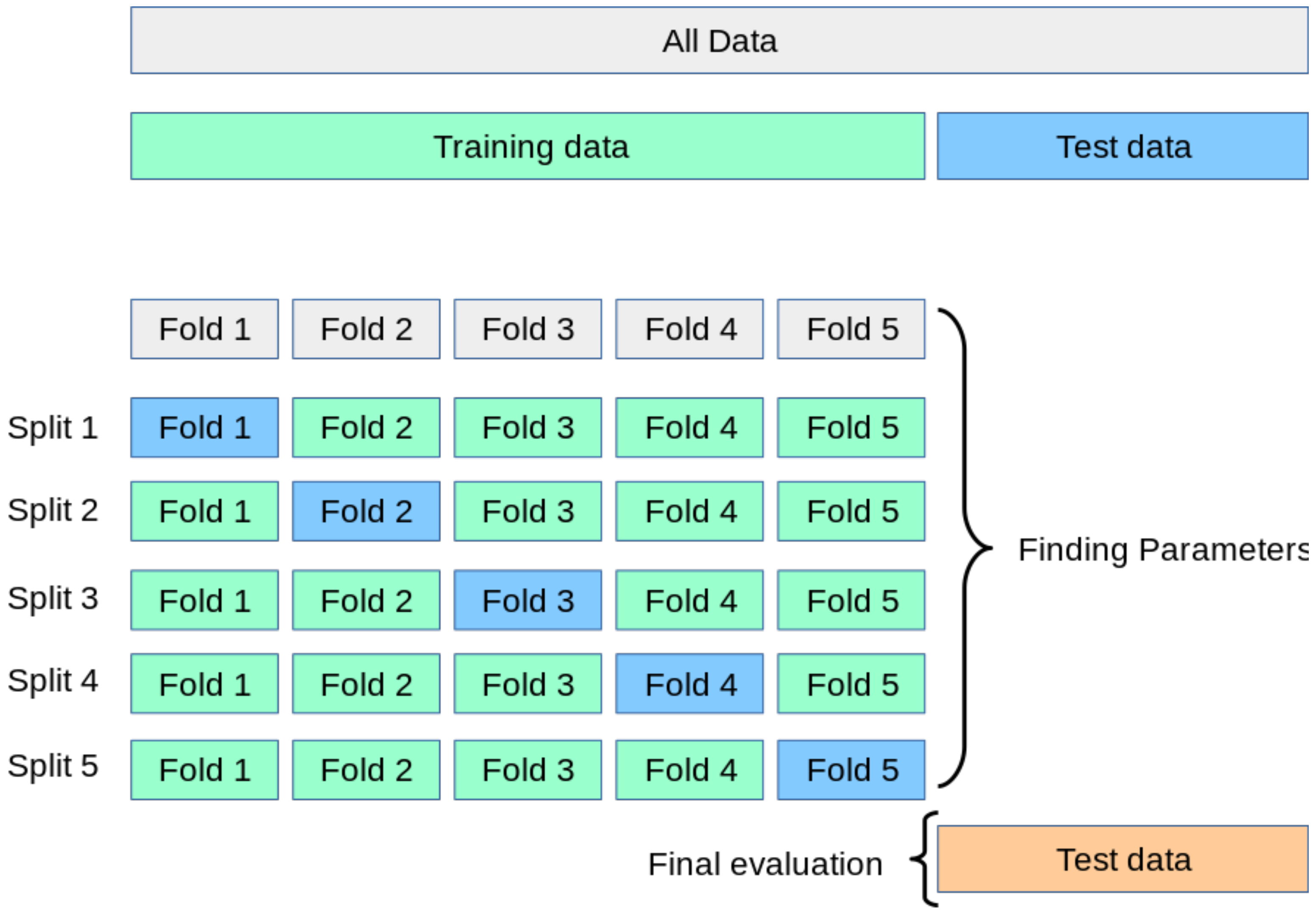
How to choose k?

- k being too small will be very sensitive to “noisy samples”, leading to noisy decision boundaries.

- k being too large will lead to over-smoothed boundaries since it looks neighbors that are far away from the test data sample.

- We can use cross validation to find k.
  - Try several values of k : {5, 10, 20, 30}
  - Select the best k based on cross validation.

Cross validation



k-NN: Some Issues and Remedies

- If some features (columns of data matrix) have large ranges, they will dominate the calculation of the distance.
  - Data Normalization
    - Min-max normalization: scale the range of each feature to be in range [0, 1]
    - Decimal normalization: scale each feature to be in range (-1, 1)
    - Z-score normalization: scale each feather to follow standard normal distribution
- Irrelevant, correlated features add noise to distance measures
  - Eliminate irrelevant features
  - Principal Component Analysis to reduce correlation among features
- Categorical Features, e.g., {‘red’, ’green’, ’blue’}
  - One-hot encoding