CS 4900/5900: Machine Learning

The Perceptron Algorithm

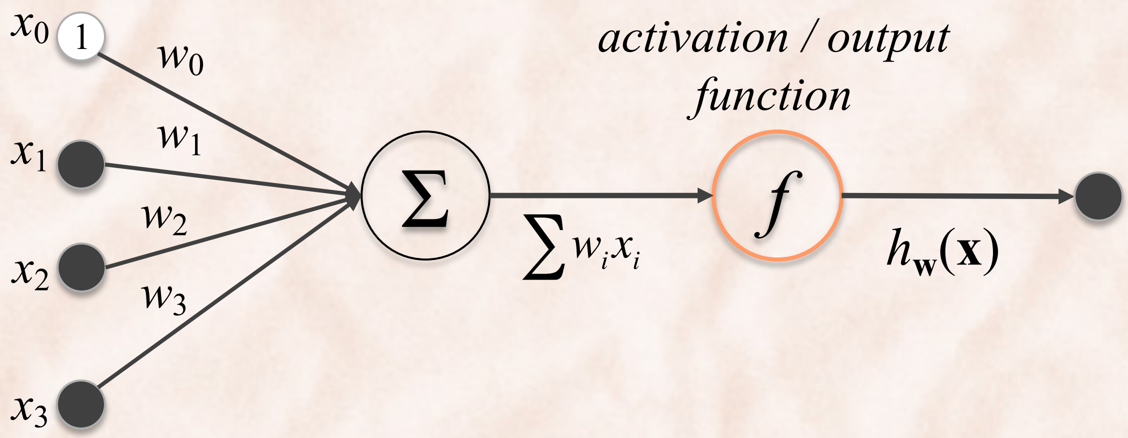
The Kernel Trick

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

McCulloch-Pitts Neuron Function



- Algebraic interpretation:
  - The output of the neuron is a **linear combination** of inputs from other neurons, **rescaled by** the synaptic **weights**.
    - weights w_i correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
  - It is often transformed through a monotonic **activation function**.

Activation/Output Functions

unit step $f(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$

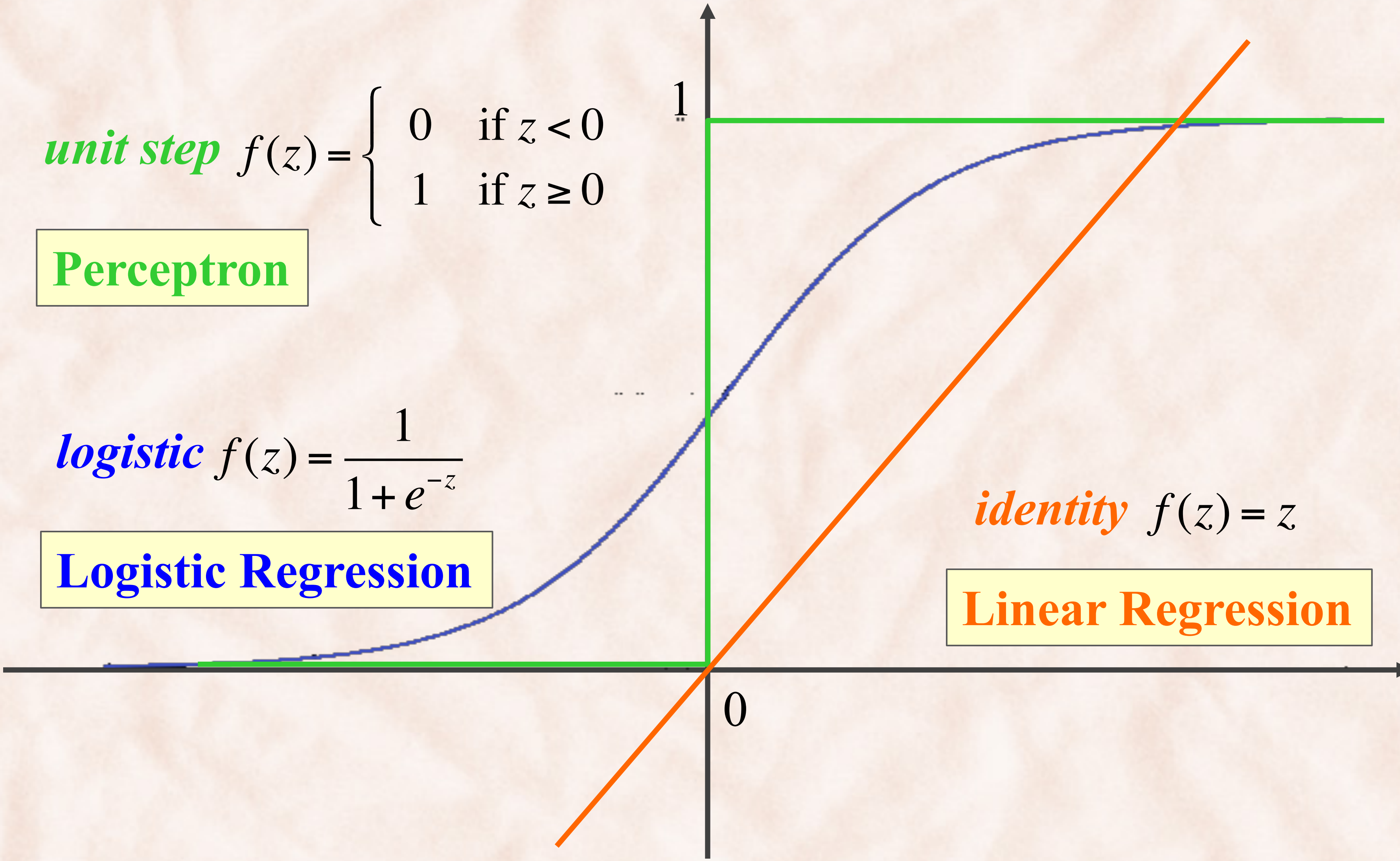
Perceptron

logistic $f(z) = \dfrac{1}{1 + e^{-z}}$

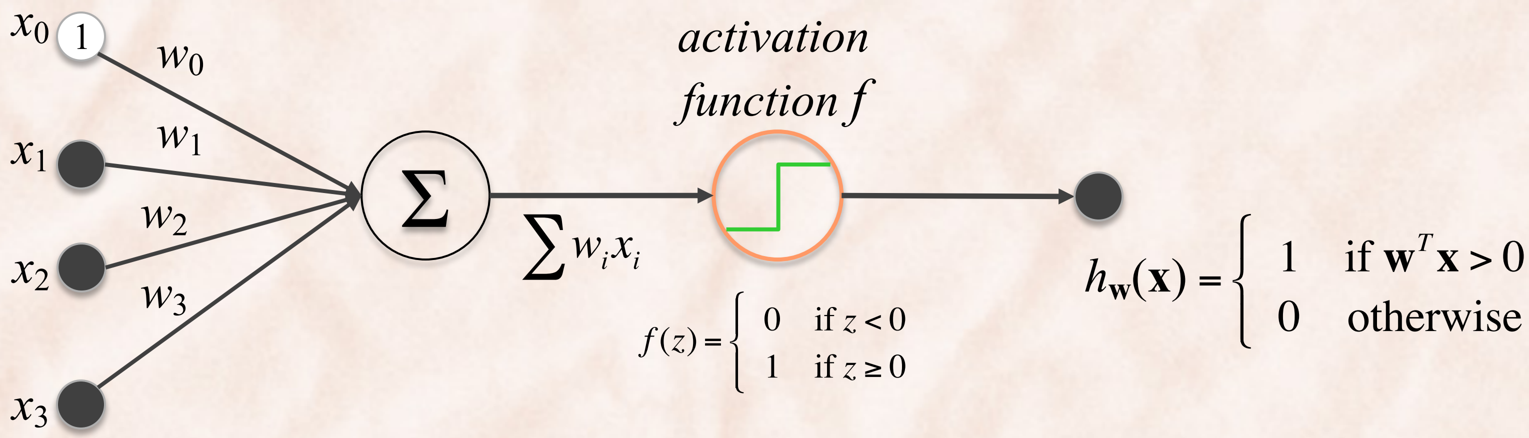
Logistic Regression

identity $f(z) = z$

Linear Regression



Perceptron



- Assume classes T = $\{c_1, c_2\} = \{1, -1\}$.
- Training set is $(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots (\mathbf{x}_n, t_n)$.

  $\mathbf{x} = [1, x_1, x_2, \ldots, x_k]^T$

  $h(\mathbf{x}) = step(\mathbf{w}^T\mathbf{x})$

Perceptron Learning

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1, \ldots, w_k]$
  - Find $\mathbf{w}$ that minimizes an *error function* $E(\mathbf{w})$ which measures the misfit between $h(\mathbf{x}_n, \mathbf{w})$ and t_n.
  - Expect that $h(\mathbf{x},\mathbf{w})$ performing well on training examples $x_n \Rightarrow h(x,\mathbf{w})$ will perform well on arbitrary test examples $\mathbf{x} \in X$.

- **Least Squares** error function?

$$E(\mathbf{w}) = \frac{1}{2}\boxed{\sum_{n=1}^{N}\{h(\mathbf{x}_n, \mathbf{w}) - t_n\}^2}$$

# of mistakes

Least Squares vs. Perceptron Criterion

- **Least Squares** => cost is # of misclassified patterns:
  - Piecewise constant function of $\mathbf{w}$ with discontinuities.
  - Cannot find closed form solution for $\mathbf{w}$ that minimizes cost.
  - Cannot use gradient methods (gradient zero almost everywhere).

- **Perceptron Criterion**:
  - Set labels to be +1 and − 1. Want $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n > 0$ for $t_n = 1$, and $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n < 0$ for $t_n = -1$.
    - $\Rightarrow$ would like to have $\mathbf{w}^{\mathrm{T}}\mathbf{x}_n t_n > 0$ for all patterns.
    - $\Rightarrow$ want to minimize $-\mathbf{w}^{\mathrm{T}}\mathbf{x}_n t_n$ for all missclassified patterns M.

$$\boxed{\Rightarrow \text{minimize } E_p(\mathbf{w}) = -\sum_{n\in M} \mathbf{w}^T \mathbf{x}_n t_n}$$

Stochastic Gradient Descent

- **Perceptron Criterion**:

$$\boxed{\text{minimize } E_p(\mathbf{w}) = -\textstyle\sum_{n \in M} \mathbf{w}^T \mathbf{x}_n t_n}$$

- Update parameters **w** sequentially **after each mistake**:

$$\begin{aligned} \mathbf{w}^{(\tau+1)} &= \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}^{(\tau)}, \mathbf{x}_n) \\ &= \mathbf{w}^{(\tau)} + \eta \mathbf{x}_n t_n \end{aligned}$$

- The magnitude of **w** is inconsequential => set $\eta = 1$.

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \mathbf{x}_n t_n$$

The Perceptron Algorithm: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $n = 1 \ldots N$
3. $\quad h_n = sgn(\mathbf{w}^T \mathbf{x}_n)$
4. $\quad$ **if** $h_n \neq t_n$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + t_n \mathbf{x}_n$

Repeat:
a) until convergence.
b) for a number of epochs E.

Theorem [Rosenblatt, 1962]:

If the training dataset is linearly separable, the perceptron learning algorithm is guaranteed to find a solution in a finite number of steps.

- see Theorem 1 (Block, Novikoff) in [Freund & Schapire, 1999].

Averaged Perceptron: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$, $\tau = 1$, $\overline{\mathbf{w}} = 0$
2. **for** $n = 1 \ldots N$
3. $\quad h_n = sgn(\mathbf{w}^T \mathbf{x}_n)$
4. $\quad$ **if** $h_n \neq t_n$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + t_n \mathbf{x}_n$
6. $\quad \overline{\mathbf{w}} = \overline{\mathbf{w}} + \mathbf{w}$
7. $\quad \tau = \tau + 1$
8. **return** $\overline{\mathbf{w}} / \tau$

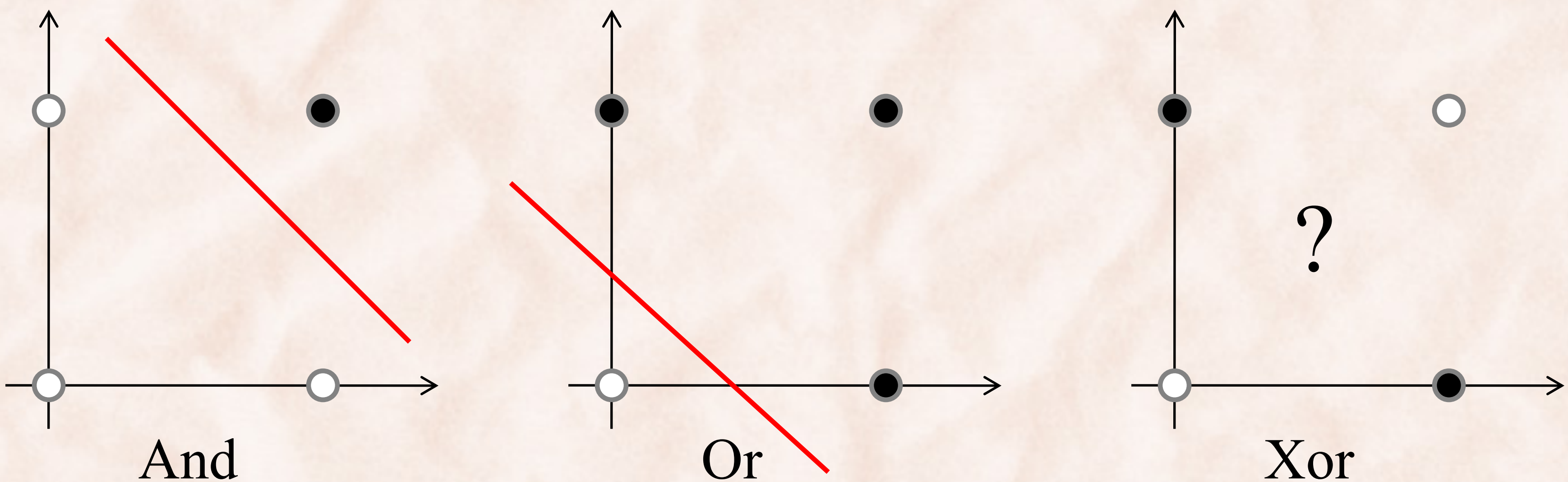
Repeat:

a) until convergence.

b) for a number of epochs E.

During testing: $h(\mathbf{x}) = sgn(\overline{\mathbf{w}}^T \mathbf{x})$

Linear vs. Non-linear Decision Boundaries



$$\left.\begin{aligned}\varphi(\mathbf{x}) &= [1, x_1, x_2]^T \\ \mathbf{w} &= [w_0, w_1, w_2]^T\end{aligned}\right\} \Rightarrow \mathbf{w}^T \varphi(\mathbf{x}) = [w_1, w_2]^T [x_1, x_2] + w_0$$

How to Find Non-linear Decision Boundaries

1) Logistic Regression with manually engineered features:
   – Quadratic features.

2) Kernel methods (e.g. SVMs) with non-linear kernels:
   – Quadratic kernels, Gaussian kernels.

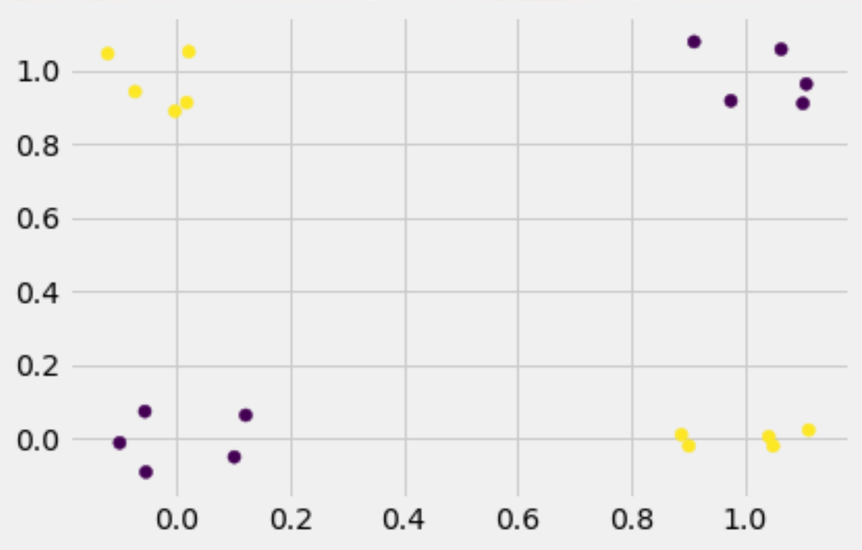
Deep Learning class

3) Unsupervised feature learning (e.g. auto-encoders):
   – Plug learned features in any linear classifier.

4) Neural Networks with one or more hidden layers:
   – Automatically learned features.

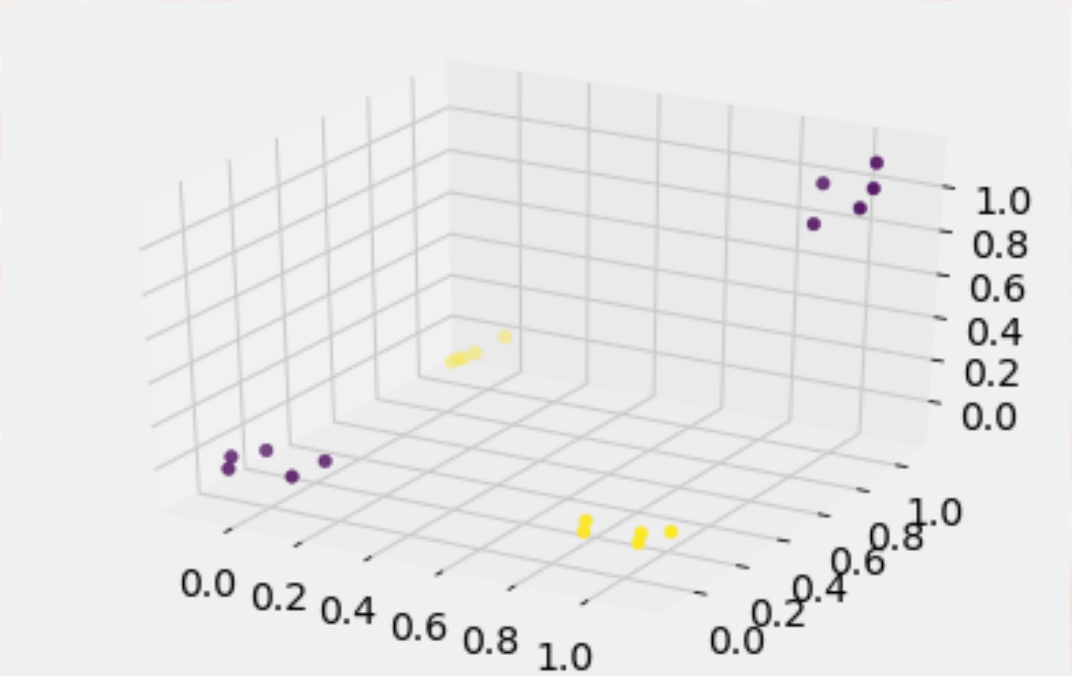
Non-Linear Classification: XOR Dataset

$$\mathbf{x} = [x_1, x_2]$$



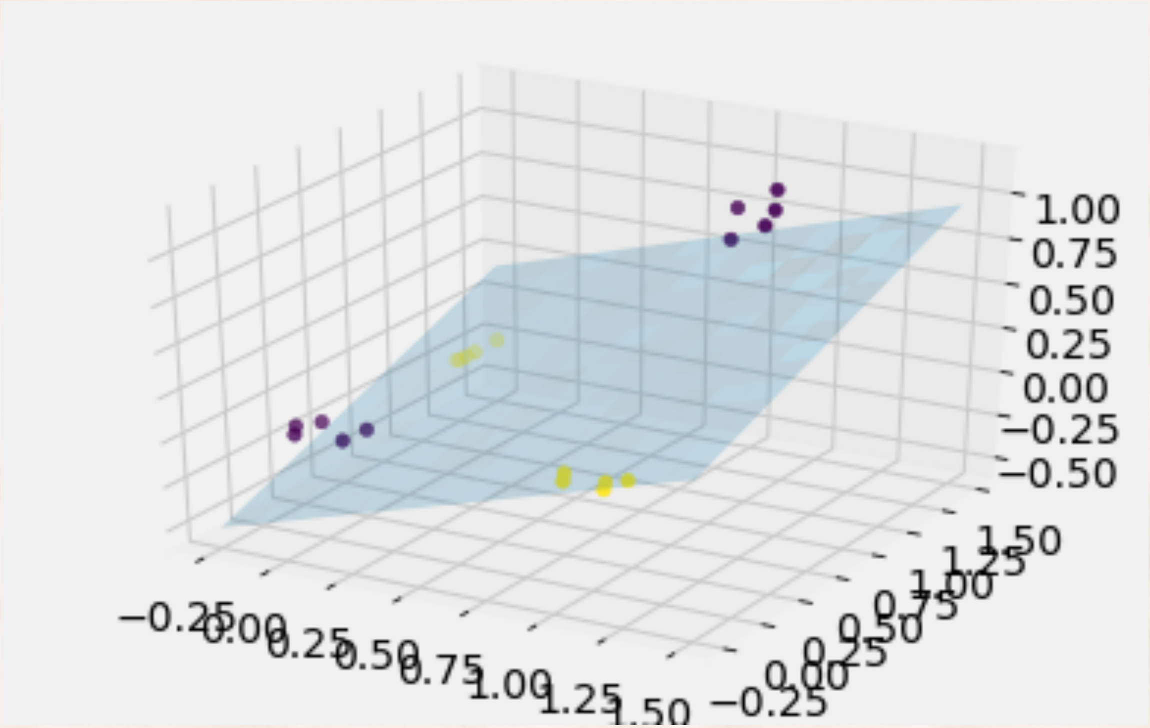
1) Manually Engineered Features: Add x_1x_2

$$\mathbf{x} = [x_1, x_2, x_1x_2]$$



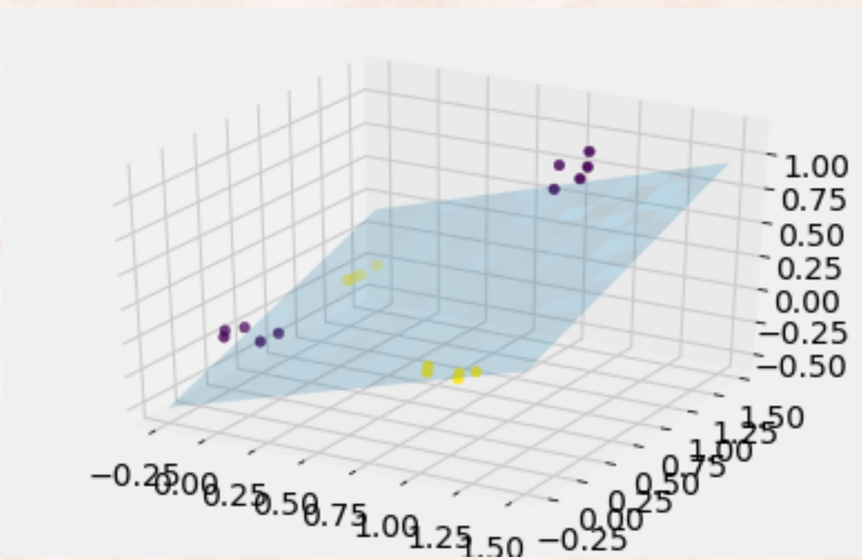
Logistic Regression with Manually Engineered Features

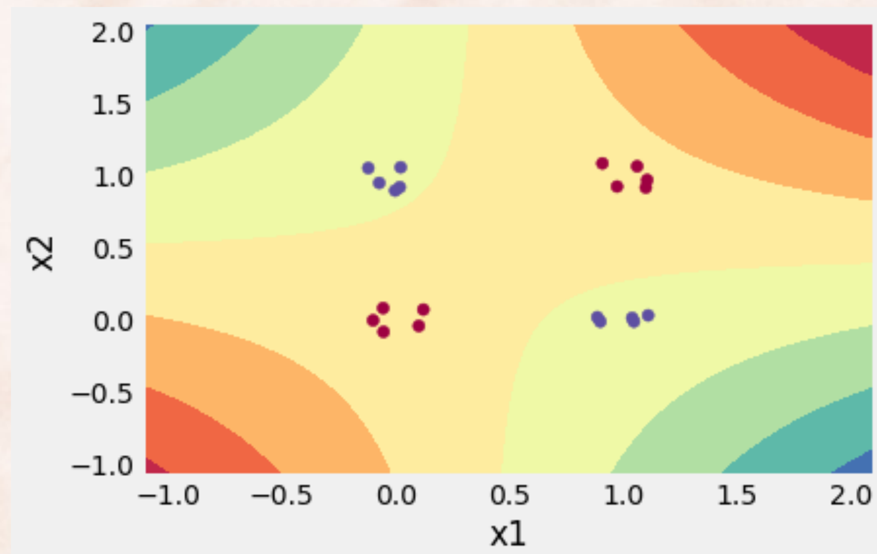
$$\mathbf{x} = [x_1, x_2, x_1 x_2]$$



Perceptron with Manually Engineered Features

Project $\mathbf{x} = [x_1, x_2, x_1 x_2]$ and decision hyperplane back to $\mathbf{x} = [x_1, x_2]$





2) Kernel Methods with Non-Linear Kernels

- Perceptrons, SVMs can be '*kernelized*':
  1. Re-write the algorithm such that during training and testing feature vectors $\mathbf{x}$, $\mathbf{y}$ appear only in dot-products $\mathbf{x}^T\mathbf{y}$.
  2. Replace dot-products $\mathbf{x}^T\mathbf{y}$ with *non-linear kernels* $K(\mathbf{x}, \mathbf{y})$:
     - K is a kernel if and only if $\exists\varphi$ such that $K(\mathbf{x}, \mathbf{y}) = \varphi(\mathbf{x})^T \varphi(\mathbf{y})$
       - φ can be in a much higher dimensional space.
         - » e.g. combinations of up to k original features
       - $\varphi(\mathbf{x})^T \varphi(\mathbf{y})$ can be computed efficiently without enumerating $\varphi(\mathbf{x})$ or $\varphi(\mathbf{y})$.

The Perceptron Algorithm: Two Classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $n = 1 \ldots \mathrm{N}$
3. $\quad h_n = sgn(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)$
4. $\quad$ **if** $h_n \neq t_n$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + t_n\mathbf{x}_{\mathrm{n}}$

Repeat:
a) until convergence.
b) for a number of epochs E.

Loop invariant: **w** is a weighted sum of training vectors:

$$\mathbf{w} = \sum_n \alpha_n t_n \mathbf{x}_n \quad \Rightarrow \quad \mathbf{w}^T\mathbf{x} = \sum_n \alpha_n t_n \mathbf{x}_n^T \mathbf{x}$$

Kernel Perceptron: Two Classes

1. **define** $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_n \alpha_n t_n \mathbf{x}_n^T \mathbf{x} = \sum_n \alpha_n t_n K(\mathbf{x}_n, \mathbf{x})$
2. **initialize** dual parameters $\alpha_n = 0$
3. **for** $n = 1 \ldots \mathrm{N}$
4. $\quad h_n = sgn\, f(\mathbf{x}_n)$
5. $\quad$ **if** $h_n \neq t_n$ **then**
6. $\quad\quad \alpha_n = \alpha_n + 1$

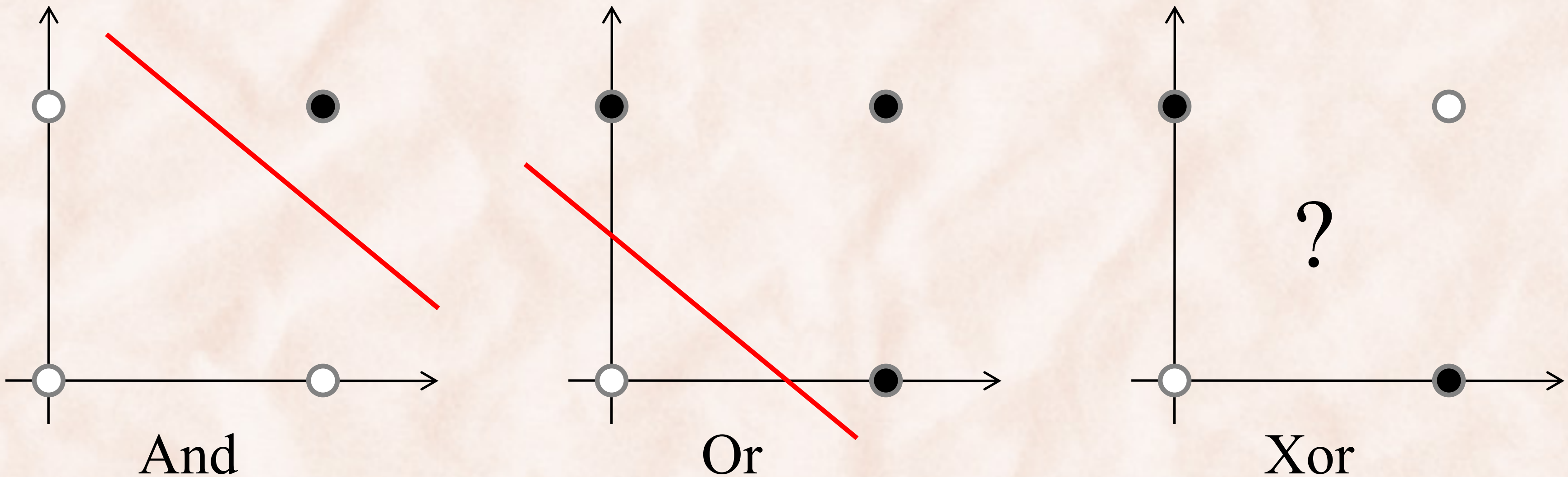
During testing: $h(\mathbf{x}) = sgn\, f(\mathbf{x})$

Kernel Perceptron: Two Classes

1. **define** $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_n \alpha_n \mathbf{x}_n^T \mathbf{x} = \sum_n \alpha_n K(\mathbf{x}_n, \mathbf{x})$
2. **initialize** dual parameters $\alpha_n = 0$
3. **for** $n = 1 \ldots N$
4. $\quad h_n = sgn\, f(\mathbf{x}_n)$
5. $\quad$ **if** $h_n \neq t_n$ **then**
6. $\quad\quad \alpha_n = \alpha_n + t_n$

During testing: $h(\mathbf{x}) = sgn\, f(\mathbf{x})$

The Perceptron vs. Boolean Functions



$$\left.\begin{aligned}\varphi(\mathbf{x}) &= [1, x_1, x_2]^T \\ \mathbf{w} &= [w_0, w_1, w_2]^T\end{aligned}\right\} \Rightarrow \mathbf{w}^T\varphi(\mathbf{x}) = [w_1, w_2]^T[x_1, x_2] + w_0$$

Perceptron with Quadratic Kernel

- Discriminant function:

$$f(\mathbf{x}) = \sum_i \alpha_i t_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}) = \sum_i \alpha_i t_i K(\mathbf{x}_i, \mathbf{x})$$

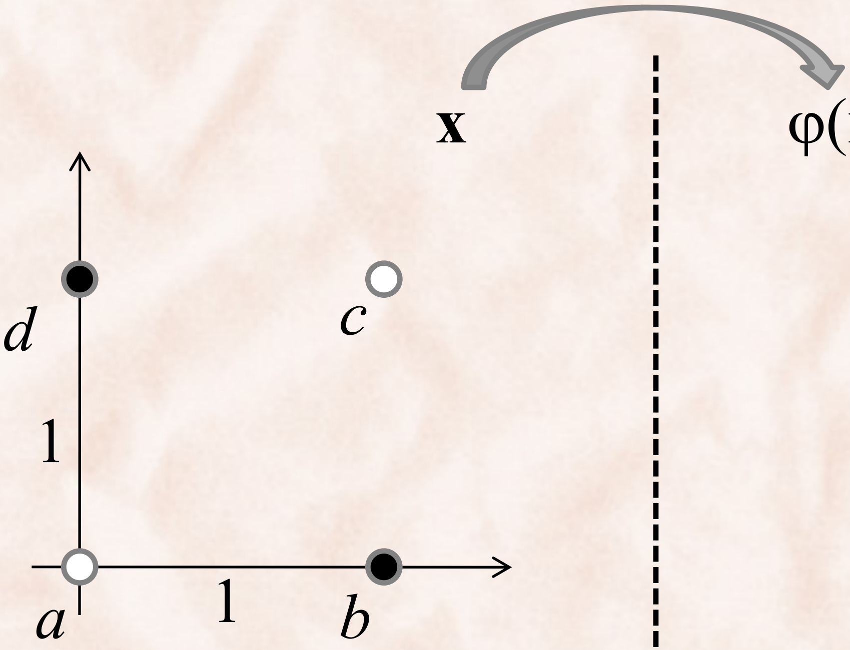
- Quadratic kernel:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2 = (x_1 y_1 + x_2 y_2)^2$$

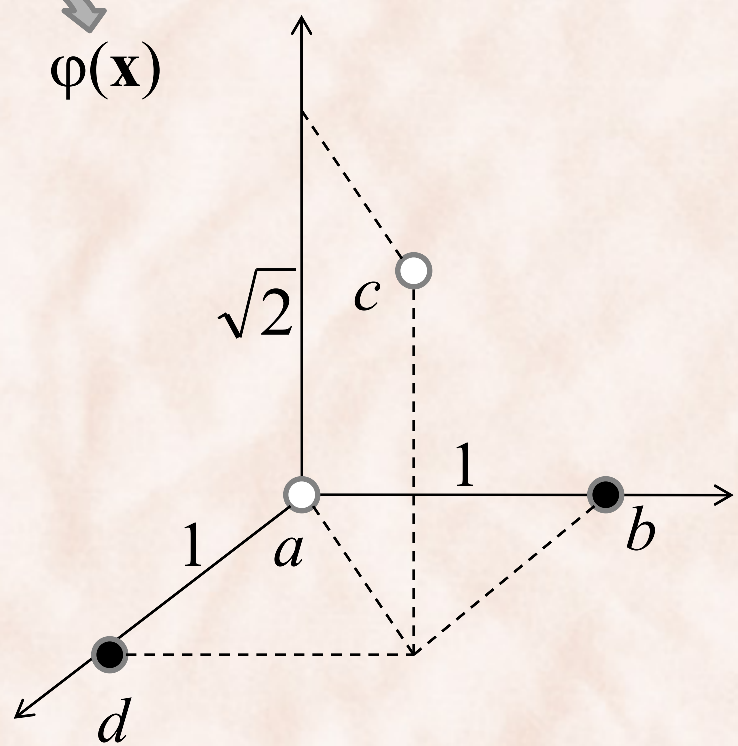
$\Rightarrow$ corresponding feature space $\varphi(\mathbf{x})$ = ?

conjunctions of two atomic features

Perceptron with Quadratic Kernel



Linear kernel $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{y}$



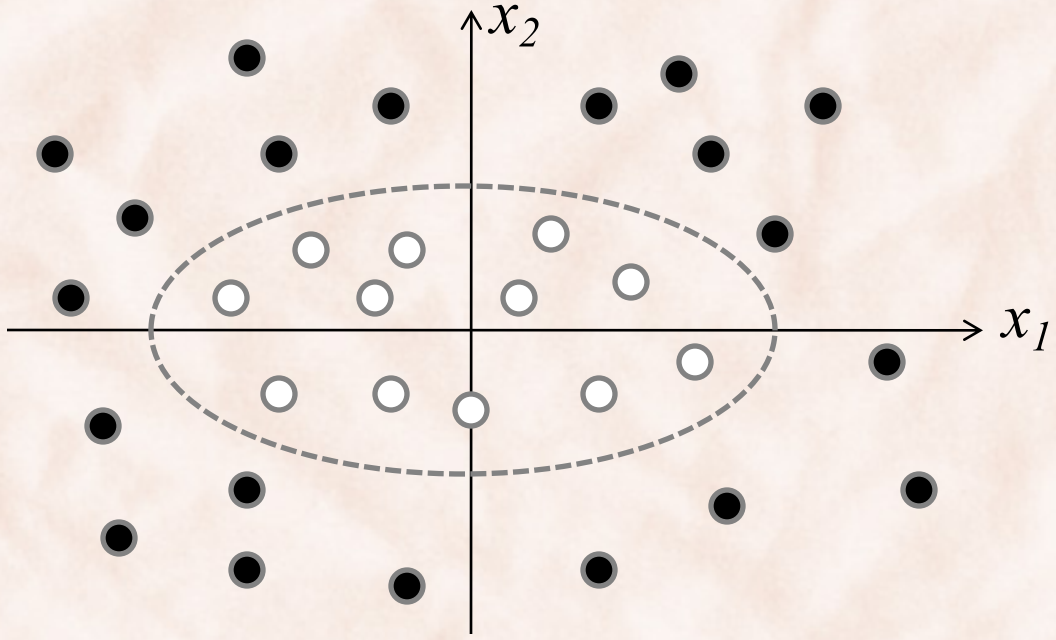
Quadratic kernel $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2$

Quadratic Kernels

- Circles, hyperbolas, and ellipses as separating surfaces:

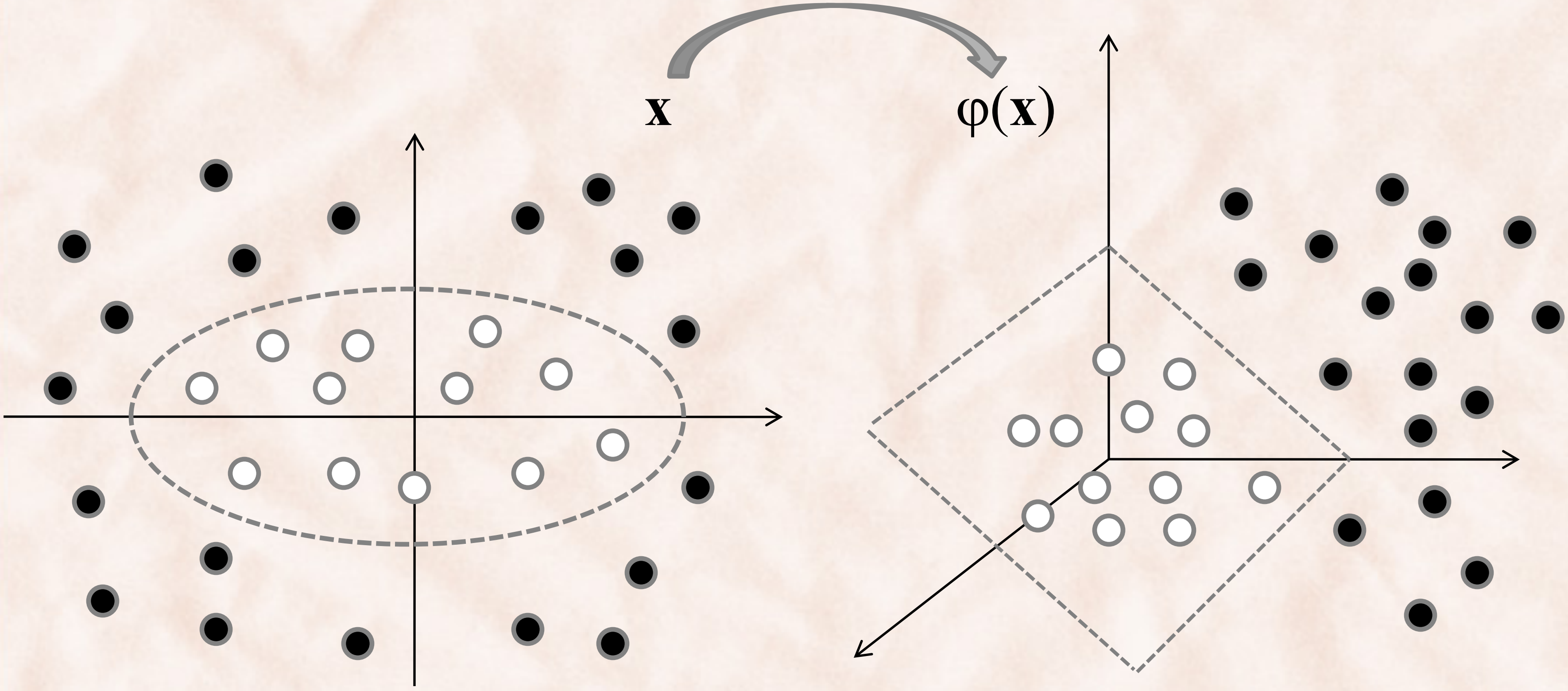
$$K(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}^T\mathbf{y})^2 = \varphi(x)^T\varphi(y)$$

$$\varphi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2]^T$$



Quadratic Kernels

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2 = \varphi(\mathbf{x})^T\varphi(\mathbf{y})$$



Explicit Features vs. Kernels

- Explicitly enumerating features can be prohibitive:
  - 1,000 basic features for $\mathbf{x}^T\mathbf{y}$ => 500,500 quadratic features for $(\mathbf{x}^T\mathbf{y})^2$
  - Much worse for higher order features.

- Solution:
  - Do not compute the feature vectors, compute kernels instead (i.e. compute dot products between implicit feature vectors).
    - $(\mathbf{x}^T\mathbf{y})^2$ takes 1001 multiplications.
    - $\varphi(\mathbf{x})^T \varphi(\mathbf{y})$ in feature space takes 500,500 multiplications.

Kernel Functions

- Definition:

  A function $k : \mathrm{X} \times \mathrm{X} \rightarrow \mathrm{R}$ is a kernel function if there exists a feature mapping $\varphi : \mathrm{X} \rightarrow \mathrm{R}^{\mathrm{n}}$ such that:

  $$k(\mathbf{x},\mathbf{y}) = \varphi(\mathbf{x})^{\mathrm{T}} \varphi(\mathbf{y})$$

- Theorem:

  $k : \mathrm{X} \times \mathrm{X} \rightarrow \mathrm{R}$ is a valid kernel $\Leftrightarrow$ the Gram matrix K whose elements are given by $k(\mathbf{x}_{\mathrm{n}},\mathbf{x}_{\mathrm{m}})$ is *positive semidefinite* for all possible choices of the set $\{\mathbf{x}_{\mathrm{n}}\}$.

Kernel Examples

- Linear kernel: $K(\mathbf{x},\mathbf{y}) = \mathbf{x}^T\mathbf{y}$
- Quadratic kernel: $K(\mathbf{x},\mathbf{y}) = (c + \mathbf{x}^T\mathbf{y})^2$
  - contains constant, linear terms and terms of order two (c > 0).
- Polynomial kernel: $K(\mathbf{x},\mathbf{y}) = (c + \mathbf{x}^T\mathbf{y})^M$
  - contains all terms up to degree M (c > 0).
- Gaussian kernel: $K(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2 / 2\sigma^2)$
  - corresponding feature space has infinite dimensionality.

Techniques for Constructing Kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$k(\mathbf{x}, \mathbf{x}') = c k_1(\mathbf{x}, \mathbf{x}') \tag{6.13}$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') \tag{6.14}$$

$$k(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.15}$$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.16}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{6.17}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') \tag{6.18}$$

$$k(\mathbf{x}, \mathbf{x}') = k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \tag{6.19}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' \tag{6.20}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') \tag{6.21}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') k_b(\mathbf{x}_b, \mathbf{x}_b') \tag{6.22}$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Kernels over Discrete Structures

- Subsequence Kernels [Lodhi et al., JMLR 2002]:
  - Σ is a finite alphabet (set of symbols).
  - $\mathbf{x},\mathbf{y} \in \Sigma^*$ are two sequences of symbols with lengths $|\mathbf{x}|$ and $|\mathbf{y}|$
  - $k(\mathbf{x},\mathbf{y})$ is defined as the number of common substrings of length n.
  - $k(\mathbf{x},\mathbf{y})$ can be computed in $O(n|\mathbf{x}||\mathbf{y}|)$ time complexity.

- Tree Kernels [Collins and Duffy, NIPS 2001]:
  - T_1 and T_2 are two trees with N_1 and N_2 nodes respectively.
  - $k(T_1, T_2)$ is defined as the number of common subtrees.
  - $k(T_1, T_2)$ can be computed in $O(N_1 N_2)$ time complexity.
  - in practice, time is linear in the size of the trees.

Supplementary Reading

- PRML Chapter 6:
  - Section 6.1 on dual representations for linear regression models.
  - Section 6.2 on techniques for constructing new kernels.

The Perceptron Algorithm: K classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $i = 1 \ldots n$
3. $\quad y_i = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. $\quad$ **if** $y_i \neq t_i$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_i, t_i) - \varphi(\mathbf{x}_i, y_i)$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing:

$$t^* = \arg\max_{t \in T} \mathbf{w}^T \phi(\mathbf{x}, t)$$

Averaged Perceptron: K classes

1. **initialize** parameters $\mathbf{w} = 0$, $\tau = 1$, $\overline{\mathbf{w}} = 0$
2. **for** $i = 1 \ldots n$
3. $\quad y_\mathrm{i} = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. $\quad$ **if** $y_\mathrm{i} \neq t_i$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_\mathrm{i}, t_i) - \varphi(\mathbf{x}_\mathrm{i}, y_\mathrm{i})$
6. $\quad \overline{\mathbf{w}} = \overline{\mathbf{w}} + \mathbf{w}$
7. $\quad \tau = \tau + 1$
8. **return** $\overline{\mathbf{w}} / \tau$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing: $t^* = \arg\max_{t \in T} \overline{\mathbf{w}}^T \varphi(\mathbf{x}, t)$

The Perceptron Algorithm: K classes

1. **initialize** parameters $\mathbf{w} = 0$
2. **for** $i = 1 \ldots n$
3. $\quad c_j = \arg\max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}_i, t)$
4. $\quad$ **if** $c_j \neq t_i$ **then**
5. $\quad\quad \mathbf{w} = \mathbf{w} + \varphi(\mathbf{x}_i, t_i) - \varphi(\mathbf{x}_i, c_j)$

Repeat:
a) until convergence.
b) for a number of epochs E.

Loop invariant: **w** is a weighted sum of training vectors:

$$\mathbf{w} = \sum_{i,j} \alpha_{ij} (\phi(\mathbf{x}_i, t_i) - \phi(\mathbf{x}_i, c_j))$$

$$\Rightarrow \quad \mathbf{w}^T \phi(\mathbf{x}, t) = \sum_{i,j} \alpha_{ij} (\phi(\mathbf{x}_i, t_i)^T \phi(\mathbf{x}, t) - \phi(\mathbf{x}_i, c_j)^T \phi(\mathbf{x}, t))$$

Kernel Perceptron: K classes

1. **define** $f(\mathbf{x},t) = \sum_{i,j} \alpha_{ij} (\phi(\mathbf{x}_i, t_i)^T \phi(\mathbf{x},t) - \phi(\mathbf{x}_i, c_j)^T \phi(\mathbf{x},t))$
2. **initialize** dual parameters $\alpha_{ij} = 0$
3. **for** $i = 1 \ldots n$
4. $\quad c_j = \arg\max_{t \in T} f(\mathbf{x}_i, t)$
5. $\quad$ **if** $y_i \neq t_i$ **then**
6. $\quad\quad \alpha_{ij} = \alpha_{ij} + 1$

Repeat:
a) until convergence.
b) for a number of epochs E.

During testing:

$$t^* = \arg\max_{t \in T} f(\mathbf{x}, t)$$

Kernel Perceptron: K classes

- Discriminant function:

$$f(\mathbf{x},t) = \sum_{i,j} \alpha_{i,j} (\phi(\mathbf{x}_i,\, t_i)^T \phi(\mathbf{x},t) - \phi(\mathbf{x}_i, c_j)^T \phi(\mathbf{x},t))$$

$$= \sum_{i,j} \alpha_{ij} (K(\mathbf{x}_i,\, t_i, \mathbf{x}, t) - K(\mathbf{x}_i, c_j, \mathbf{x}, t))$$

where:

$$K(\mathbf{x}_i, t_i, \mathbf{x}, t) = \varphi^T(\mathbf{x}_i, t_i)\varphi(\mathbf{x}, t)$$

$$K(\mathbf{x}_i, y_i, \mathbf{x}, t) = \phi^T(\mathbf{x}_i, y_i)\phi(\mathbf{x}, t)$$