Supervised Learning

Training

Training Examples \((x_k, t_k)\)

Learning Algorithm

Model \(h\)

Testing

Test Examples \((x, t)\)

Model \(h\)

Generalization Performance
Supervised Learning

• **Task** = learn an (unknown) function $t : X \rightarrow T$ that maps input instances $x \in X$ to output targets $t(x) \in T$:
  - **Classification**:
    • The output $t(x) \in T$ is one of a finite set of discrete categories.
  - **Regression**:
    • The output $t(x) \in T$ is continuous, or has a continuous component.

• Target function $t(x)$ is known (only) through (noisy) set of training examples:

$$ (x_1, t_1), (x_2, t_2), \ldots (x_n, t_n) $$
Three Parametric Approaches to Classification

1) **Discriminant Functions**: construct $f : X \rightarrow T$ that directly assigns a vector $x$ to a specific class $C_k$.
   - Inference and decision combined into a single learning problem.
   - **Linear Discriminant**: the decision surface is a hyperplane in $X$:
     - Perceptron
     - Support Vector Machines
     - Fisher’s Linear Discriminant
Three Parametric Approaches to Classification

2) **Probabilistic Discriminative Models**: directly model the posterior class probabilities $p(C_k \mid x)$.
   - Inference and decision are separate.
   - Less data needed to estimate $p(C_k \mid x)$ than $p(x \mid C_k)$.
   - Can accommodate many overlapping features.
     - Logistic Regression
     - Conditional Random Fields
Three Parametric Approaches to Classification

3) Probabilistic Generative Models:
   - Model class-conditional $p(x \mid C_k)$ as well as the priors $p(C_k)$, then use Bayes’s theorem to find $p(C_k \mid x)$.
     • or model $p(x,C_k)$ directly, then marginalize to obtain the posterior probabilities $p(C_k \mid x)$.
   - Inference and decision are separate.
   - Can use $p(x)$ for outlier or novelty detection.
   - Need to model dependencies between features.
     • Naïve Bayes.
     • Hidden Markov Models.
Generative and Discriminative Classifiers

Suppose we're distinguishing cat from dog images

ImageNet

ImageNet
Generative Classifier:

- Build a model of what's in a cat image
  - Knows about whiskers, ears, eyes
  - Assigns a probability to any image:
    - how cat-y is this image?

Also build a model for dog images

Given a new image:

Run both models and see which one fits better.
Discriminative Classifier

Just try to distinguish dogs from cats

Oh look, dogs have collars!

Let's ignore everything else.
Finding the correct class $c$ from a document $d$ in a document categorization scenario attempts to determine whether or not the document should be assigned to each of the possible categories.

**Generative vs Discriminative Classifiers**

- **Naive Bayes**

  $$\hat{c} = \arg\max_{c \in C} \left( P(d|c) \cdot P(c) \right)$$

- **Logistic Regression**

  $$\hat{c} = \arg\max_{c \in C} P(c|d)$$
Generative vs. Discriminative

Left-hand mode has no effect on posterior class probabilities.
Three Parametric Approaches to Classification

1) **Discriminant Functions**: construct $h : X \rightarrow T$ that directly assigns a vector $x$ to a specific class $C_k$.
   - Inference and decision combined into a single learning problem.
   - *Linear Discriminant*: the decision surface is a hyperplane in $X$:
     - Perceptron
     - Support Vector Machines
     - Fisher ‘s Linear Discriminant
Discriminant Function Approach to Classification

- **Task** = build a function $h(x)$ such that:
  - $h$ matches $t$ well on the training data:
    - $\Rightarrow h$ is able to fit data that it has seen.
  - $h$ also matches $t$ well on test data:
    - $\Rightarrow h$ is able to **generalize to unseen data**.

- **Task** = choose $h$ from a “nice” *class of functions* that depend on a vector of parameters $w$:
  - $h(x) \equiv h_w(x) \equiv h(w, x)$
  - what classes of functions are “nice”?
Neurons

**Soma** is the central part of the neuron:
- where the input signals are combined.

**Dendrites** are cellular extensions:
- where majority of the input occurs.

**Axon** is a fine, long projection:
- carries nerve signals to other neurons.

**Synapses** are molecular structures between axon terminals and other neurons:
- where the communication takes place.
## Neuron Models


<table>
<thead>
<tr>
<th>Year</th>
<th>Model Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1907</td>
<td>Integrate and fire</td>
<td>[13]</td>
</tr>
<tr>
<td>1943</td>
<td>McCulloch and Pitts</td>
<td>[11]</td>
</tr>
<tr>
<td>1952</td>
<td>Hodgkin-Huxley</td>
<td>[12]</td>
</tr>
<tr>
<td>1958</td>
<td>Perceptron</td>
<td>[14]</td>
</tr>
<tr>
<td>1961</td>
<td>FitzHugh-Nagumo</td>
<td>[15]</td>
</tr>
<tr>
<td>1965</td>
<td>Leaky integrate-and-fire</td>
<td>[16]</td>
</tr>
<tr>
<td>1981</td>
<td>Morris-Lecar</td>
<td>[17]</td>
</tr>
<tr>
<td>1986</td>
<td>Quadratic integrate-and-fire</td>
<td>[18]</td>
</tr>
<tr>
<td>1989</td>
<td>Hindmarsh-Rose</td>
<td>[19]</td>
</tr>
<tr>
<td>1999</td>
<td>Wilson Polynomial</td>
<td>[21]</td>
</tr>
<tr>
<td>2000</td>
<td>Integrate-and-fire or burst</td>
<td>[22]</td>
</tr>
<tr>
<td>2001</td>
<td>Resonate-and-fire</td>
<td>[23]</td>
</tr>
<tr>
<td>2003</td>
<td>Izhikevich</td>
<td>[24]</td>
</tr>
<tr>
<td>2003</td>
<td>Exponential integrate-and-fire</td>
<td>[25]</td>
</tr>
<tr>
<td>2004</td>
<td>Generalized integrate-and-fire</td>
<td>[26]</td>
</tr>
<tr>
<td>2005</td>
<td>Adaptive exponential integrate-and-fire</td>
<td>[27]</td>
</tr>
<tr>
<td>2009</td>
<td>Mihalas-Neibur</td>
<td>[28]</td>
</tr>
</tbody>
</table>
Fig. 2. (a) Illustration and (b) functional description of a leaky integrate-and-fire neuron. Weighted and delayed input signals are summed into the input current $I_{\text{app}}(t)$, which travel to the soma and perturb the internal state variable, the voltage $V$. Since $V$ is hysteric, the soma performs integration and then applies a threshold to make a spike or no-spike decision. After a spike is released, the voltage $V$ is reset to a value $V_{\text{reset}}$. The resulting spike is sent to other neurons in the network.
# Neuron Models


<table>
<thead>
<tr>
<th>Year</th>
<th>Model Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1907</td>
<td>Integrate and fire</td>
<td>[13]</td>
</tr>
<tr>
<td>1943</td>
<td>McCulloch and Pitts</td>
<td>[11]</td>
</tr>
<tr>
<td>1952</td>
<td>Hodgkin-Huxley</td>
<td>[12]</td>
</tr>
<tr>
<td>1958</td>
<td>Perceptron</td>
<td>[14]</td>
</tr>
<tr>
<td>1961</td>
<td>Fitzhugh-Nagumo</td>
<td>[15]</td>
</tr>
<tr>
<td>1965</td>
<td>Leaky integrate-and-fire</td>
<td>[16]</td>
</tr>
<tr>
<td>1981</td>
<td>Morris-Lecar</td>
<td>[17]</td>
</tr>
<tr>
<td>1986</td>
<td>Quadratic integrate-and-fire</td>
<td>[18]</td>
</tr>
<tr>
<td>1989</td>
<td>Hindmarsh-Rose</td>
<td>[19]</td>
</tr>
<tr>
<td>1999</td>
<td>Wilson Polynomial</td>
<td>[21]</td>
</tr>
<tr>
<td>2000</td>
<td>Integrate-and-fire or burst</td>
<td>[22]</td>
</tr>
<tr>
<td>2001</td>
<td>Resonate-and-fire</td>
<td>[23]</td>
</tr>
<tr>
<td>2003</td>
<td>Izhikevich</td>
<td>[24]</td>
</tr>
<tr>
<td>2003</td>
<td>Exponential integrate-and-fire</td>
<td>[25]</td>
</tr>
<tr>
<td>2004</td>
<td>Generalized integrate-and-fire</td>
<td>[26]</td>
</tr>
<tr>
<td>2005</td>
<td>Adaptive exponential integrate-and-fire</td>
<td>[27]</td>
</tr>
<tr>
<td>2009</td>
<td>Mihalas-Neibur</td>
<td>[28]</td>
</tr>
</tbody>
</table>
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 \( f(z) = \frac{1}{1 + e^{-z}} \)

Logistic Regression

identity \( f(z) = z \)

Linear Regression
Perceptron

• Assume classes $T = \{c_1, c_2\} = \{1, -1\}$.
• Training set is $(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)$.
  \[ x = [1, x_1, x_2, \ldots, x_k]^T \]
  \[ h(x) = \text{sgn}(w^T x) = \text{sgn}(w_0 + w_1 x_1 + \ldots + w_k x_k) \]

**activation function** $f$

\[ f(z) = \begin{cases} -1 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \]

**Linear discriminant function** $h_w(x) = \begin{cases} 1 & \text{if } w^T x > 0 \\ -1 & \text{otherwise} \end{cases}$
Linear Discriminant Functions

- Use a linear function of the input vector:

\[ h(\mathbf{x}) = \mathbf{w}^T \varphi(\mathbf{x}) + w_0 \]

- Decision:

\[ \mathbf{x} \in C_1 \text{ if } h(\mathbf{x}) \geq 0, \text{ otherwise } \mathbf{x} \in C_2. \]

\[ \Rightarrow \text{ decision boundary is hyperplane } h(\mathbf{x}) = 0. \]

- Properties:

  - \( \mathbf{w} \) is orthogonal to vectors lying within the decision surface.
  - \( w_0 \) controls the location of the decision hyperplane.
Geometric Interpretation

$h > 0$
$h = 0$
$h < 0$
Linear Discriminant Functions:
Two Classes ($K = 2$)

- What algorithms can be used to learn $y(x) = w^T \phi(x) + w_0$?
  Assume a training dataset of $N = N_1 + N_2$ examples in $C_1$ and $C_2$.

  - Perceptron:
    - Voted/Averaged Perceptron
    - Kernel Perceptron

  - Support Vector Machines:
    - Linear
    - Kernel
  - Fisher’s Linear Discriminant
Assume classes $T = \{c_1, c_2\} = \{1, -1\}$.

Training set is $(x_1, t_1), (x_2, t_2), \ldots (x_n, t_n)$.

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

$h(x) = sgn(w^T x) = sgn(w_0 + w_1 x_1 + \ldots + w_k x_k)$

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

$h_w(x) = \begin{cases} 1 & \text{if } w^T x > 0 \\ -1 & \text{otherwise} \end{cases}$

A linear discriminant function
Perceptron Learning

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

• Least Squares error function?

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} \left( h(x_n, w) - t_n \right)^2
\]

\( 2 \times \# \text{ of mistakes} \)
Least Squares vs. Perceptron Criterion

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

- **Perceptron Criterion**:
  - Set labels to be $+1$ and $-1$. Want $w^T x_n > 0$ for $t_n = 1$, and $w^T x_n < 0$ for $t_n = -1$.
    $\Rightarrow$ would like to have $w^T x_n t_n > 0$ for all patterns.
    $\Rightarrow$ want to minimize $-w^T x_n t_n$ for all misclassified patterns $M$.

$$\Rightarrow \text{minimize } E_p(w) = -\sum_{n \in M} w^T x_n t_n$$
Stochastic Gradient Descent

- **Perceptron Criterion:**
  \[
  \text{minimize } E_p(w) = -\sum_{n \in M} w^T x_n t_n
  \]

- Update parameters $w$ sequentially after each mistake:
  \[
  w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_p(w^{(\tau)}, x_n)
  = w^{(\tau)} + \eta x_n t_n
  \]

- The magnitude of $w$ is inconsequential => set $\eta = 1.$
  \[
  w^{(\tau+1)} = w^{(\tau)} + x_n t_n
  \]
The Perceptron Algorithm: Two Classes

1. **initialize** parameters \( w = 0 \)
2. **for** \( n = 1 \ldots N \)
   3. \( h_n = sgn(w^T x_n) \)
   4. **if** \( h_n \neq t_n \) **then**
   5. \( w = w + t_n x_n \)

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].
The Perceptron Algorithm: Two Classes

1. **initialize** parameters \( w = 0 \)
2. **for** \( n = 1 \ldots N \)
3. \( h_n = w^T x_n \)
4. **if** \( h_n t_n \leq 0 \) **then**
5. \( w = w + t_n x_n \)

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].

\( sgn(z) = +1 \) if \( z > 0 \),
\( 0 \) if \( z = 0 \),
\( -1 \) if \( z < 0 \)
The Perceptron Algorithm: Two Classes

1. initialize parameters $w = 0$
2. for $n = 1 \ldots N$
3. \[ h_n = w^T x_n \]
4. if $h_n \geq 0$ and $t_n = -1$
5. \[ w = w - x_n \]
6. if $h_n \leq 0$ and $t_n = +1$
7. \[ w = w + x_n \]

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

What is the impact of the perceptron update on the score $w^T x_n$ of the misclassified example $x_n$?
Linear vs. Non-linear Decision Boundaries

\( \varphi(\mathbf{x}) = [1, x_1, x_2]^T \)
\[ \mathbf{w} = [w_0, w_1, w_2]^T \]

\[ \Rightarrow \mathbf{w}^T \varphi(\mathbf{x}) = [w_1, w_2]^T [x_1, x_2] + w_0 \]
How to Find Non-linear Decision Boundaries

1) Perceptron with manually engineered features:
   – Quadratic features.

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

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

\[ \mathbf{x} = [x_1, x_2, x_1x_2] \]
Perceptron with Manually Engineered Features

Project \( x = [x_1, x_2, x_1x_2] \) and decision hyperplane back to \( x = [x_1, x_2] \)
Averaged Perceptron: Two Classes

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

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

$s\text{gn}(z) = +1$ if $z > 0$,  
$0$ if $z = 0$,  
$-1$ if $z < 0$

Repeat:
 a) until convergence.
 b) for a number of epochs $E$. 
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 \( x, y \) appear only in dot-products \( x^T y \).
  2. Replace dot-products \( x^T y \) with non-linear kernels \( K(x, y) \):
     • \( K \) is a kernel if and only if \( \exists \varphi \) such that \( K(x, y) = \varphi(x)^T \varphi(y) \)
       – \( \varphi \) can be in a much higher dimensional space.
       » e.g. combinations of up to \( k \) original features
       – \( \varphi(x)^T \varphi(y) \) can be computed efficiently without enumerating \( \varphi(x) \) or \( \varphi(y) \).
The Perceptron Algorithm: Two Classes

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

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

Loop invariant: $\mathbf{w}$ is a weighted sum of training vectors:

$$\mathbf{w} = \sum_{n=1..N} \alpha_n t_n \mathbf{x}_n \Rightarrow \mathbf{w}^T \mathbf{x} = \sum_{n=1..N} \alpha_n t_n \mathbf{x}_n^T \mathbf{x}$$
Kernel Perceptron: Two Classes

1. define $f(x) = w^T x = \sum_{j=1..N} \alpha_j t_j x_j^T x = \sum_{j=1..N} \alpha_j t_j K(x_j, x)$
2. initialize dual parameters $\alpha_n = 0$
3. for $n = 1 \ldots N$
4. $h_n = sgn f(x_n)$
5. if $h_n \neq t_n$ then
6. $\alpha_n = \alpha_n + 1$

During testing: $h(x) = sgn f(x)$
Kernel Perceptron: Two Classes

1. **define** \( f(x) = w^T x = \sum_{j=1..N} \alpha_j t_j x_j^T x = \sum_{j=1..N} \alpha_j t_j K(x_j, x) \)

2. **initialize** dual parameters \( \alpha_n = 0 \)

3. **for** \( n = 1 \ldots N \)

4. \( h_n = \text{sgn} f(x_n) \)

5. **if** \( h_n \neq t_n \) **then**

6. \( \alpha_n = \alpha_n + 1 \)

Let \( S = \{ j | \alpha_j \neq 0 \} \) be the set of *support vectors*. Then \( f(x) = \sum_{j \in S} \alpha_j t_j K(x_j, x) \)

During testing: \( h(x) = \text{sgn} f(x) \)
Kernel Perceptron: Equivalent Formulation

1. define $f(x) = w^T x = \sum_j \alpha_j x_j^T x = \sum_j \alpha_j K(x_j, x)$
2. initialize dual parameters $\alpha_n = 0$
3. for $n = 1 \ldots N$
4. $h_n = \text{sgn} f(x_n)$
5. if $h_n \neq t_n$ then
6. $\alpha_n = \alpha_n + t_n$

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

During testing: $h(x) = \text{sgn} f(x)$
The Perceptron vs. Boolean Functions

\[ \varphi(x) = [1, x_1, x_2]^T \]

\[ w = [w_0, w_1, w_2]^T \]

\[ \Rightarrow w^T \varphi(x) = [w_1, w_2]^T [x_1, x_2] + w_0 \]
Perceptron with Quadratic Kernel

- Discriminant function:
  \[ f(x) = \sum_i \alpha_i t_i \varphi(x_i)^T \varphi(x) = \sum_i \alpha_i t_i K(x_i, x) \]

- Quadratic kernel:
  \[ K(x, y) = (x^T y)^2 = (x_1 y_1 + x_2 y_2)^2 \]

\[ \Rightarrow \text{corresponding feature space } \varphi(x) = ? \]

*conjunctions of two atomic features*
Perceptron with Quadratic Kernel

Linear kernel \( K(x, y) = x^T y \)

Quadratic kernel \( K(x, y) = (x^T y)^2 \)
Quadratic Kernels

- Circles, hyperbolas, and ellipses as separating surfaces:
  
  $K(x, y) = (1 + x^T y)^2 = \varphi(x)^T \varphi(y)$

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

\[ K(x, y) = (x^T y)^2 = \phi(x)^T \phi(y) \]
Explicit Features vs. Kernels

• Explicitly enumerating features can be prohibitive:
  – 1,000 basic features for $\mathbf{x}^T \mathbf{y} \Rightarrow 500,500$ 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 $k : X \times X \to \mathbb{R}$ is a kernel function if there exists a feature mapping $\varphi : X \to \mathbb{R}^n$ such that:
  
  $$k(x,y) = \varphi(x)^T \varphi(y)$$

- **Theorem:**
  $k : X \times X \to \mathbb{R}$ is a valid kernel $\iff$ the Gram matrix $K$ whose elements are given by $k(x_n,x_m)$ is positive semidefinite for all possible choices of the set $\{x_n\}$. 
Kernel Examples

• **Linear kernel:** $K(x, y) = x^T y$

• **Quadratic kernel:** $K(x, y) = (c + x^T y)^2$
  – contains constant, linear terms and terms of order two ($c > 0$).

• **Polynomial kernel:** $K(x, y) = (c + x^T y)^M$
  – contains all terms up to degree $M$ ($c > 0$).

• **Gaussian kernel:** $K(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2)$
  – corresponding feature space has infinite dimensionality.
Techniques for Constructing Kernels

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

\[
\begin{align*}
    k(x, x') &= c k_1(x, x') \\
    k(x, x') &= f(x) k_1(x, x') f(x') \\
    k(x, x') &= q(k_1(x, x')) \\
    k(x, x') &= \exp(k_1(x, x')) \\
    k(x, x') &= k_1(x, x') + k_2(x, x') \\
    k(x, x') &= k_1(x, x') k_2(x, x') \\
    k(x, x') &= k_3(\phi(x), \phi(x')) \\
    k(x, x') &= x^T A x' \\
    k(x, x') &= k_a(x_a, x'_a) + k_b(x_b, x'_b) \\
    k(x, x') &= k_a(x_a, x'_a) k_b(x_b, x'_b)
\end{align*}
\]

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\phi(x)$ is a function from $x$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $A$ is a symmetric positive semidefinite matrix, $x_a$ and $x_b$ are variables (not necessarily disjoint) with $x = (x_a, x_b)$, 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]:
  - \( \Sigma \) is a finite alphabet (set of symbols).
  - \( x, y \in \Sigma^* \) are two sequences of symbols with lengths \(|x|\) and \(|y|\).
  - \( k(x, y) \) is defined as the number of common substrings of length \( n \).
  - \( k(x, y) \) can be computed in \( O(n|x||y|) \) 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 \( O(N_1N_2) \) 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.
Linear Discriminant Functions: Multiple Classes (K > 2)

1) Train K or K–1 one-versus-the-rest classifiers.
2) Train K(K–1)/2 one-versus-one classifiers.

3) Train K linear functions:
\[ y_k(x) = w_k^T \varphi(x) + w_{k0} \]

• Decision:
\[ x \in C_k \text{ if } y_k(x) > y_j(x), \text{ for all } j \neq k. \]
\[ \Rightarrow \text{decision boundary between classes } C_k \text{ and } C_j \text{ is hyperplane defined by } y_k(x) = y_j(x) \text{ i.e. } (w_k - w_j)^T \varphi(x) + (w_{k0} - w_{j0}) = 0 \]
\[ \Rightarrow \text{same geometrical properties as in binary case.} \]
4) More general ranking approach:

\[ y(x) = \arg \max_{t \in T} w^T \phi(x, t) \quad \text{where} \quad T = \{c_1, c_2, \ldots, c_K\} \]

- It subsumes the approach with K separate linear functions.
- Useful when T is very large (e.g. exponential in the size of input x), assuming inference can be done efficiently.
The Perceptron Algorithm: K classes

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

During testing:

$$t^* = \arg \max_{t \in T} \mathbf{w}^T \varphi(\mathbf{x}, t)$$
Averaged Perceptron: K classes

1. **initialize** parameters $w = 0$, $\tau = 1$, $\bar{w} = 0$
2. **for** $i = 1 \ldots n$
3. $y_i = \arg\max_{t \in T} w^T \varphi(x_i, t)$
4. **if** $y_i \neq t_i$ **then**
5. $w = w + \varphi(x_i, t_i) - \varphi(x_i, y_i)$
6. $\bar{w} = \bar{w} + w$
7. $\tau = \tau + 1$
8. **return** $\bar{w} / \tau$

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

Repeat:
- a) until convergence.
- b) for a number of epochs $E$. 
The Perceptron Algorithm: K classes

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

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

Loop invariant: \( w \) is a weighted sum of training vectors:

\[
 w = \sum_{i,j} \alpha_{ij} (\varphi(x_i, t_i) - \varphi(x_i, c_j))
\]

\[
 w^T \varphi(x, t) = \sum_{i,j} \alpha_{ij} (\varphi(x_i, t_i)^T \varphi(x, t) - \varphi(x_i, c_j)^T \varphi(x, t))
\]
Kernel Perceptron: K classes

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

During testing:
\[
t^* = \arg \max_{t \in T} f(x, t)
\]
Kernel Perceptron: K classes

- Discriminant function:

\[ f(x,t) = \sum_{i,j} \alpha_{i,j} (\phi(x_i, t_i)^T \phi(x,t) - \phi(x_i, c_j)^T \phi(x,t)) \]

\[ = \sum_{i,j} \alpha_{ij} (K(x_i, t_i, x, t) - K(x_i, c_j, x, t)) \]

where:

\[ K(x_i, t_i, x, t) = \phi^T (x_i, t_i) \phi(x,t) \]

\[ K(x_i, y_i, x, t) = \phi^T (x_i, y_i) \phi(x,t) \]