# CSC 411: Lecture 16: Kernels

Richard Zemel, Raquel Urtasun and Sanja Fidler

University of Toronto

# Today

- Kernel trick

# Summary of Linear SVM

- Binary and linear separable classification

# Summary of Linear SVM

- Binary and linear separable classification
- Linear classifier with maximal margin

# Summary of Linear SVM

- Binary and linear separable classification
- Linear classifier with maximal margin
- Training SVM by maximizing

$$\max_{\alpha_i \geq 0} \{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \}$$

$$\text{subject to} \quad \alpha_i \geq 0; \quad \sum_{i=1}^{N} \alpha_i t^{(i)} = 0$$

# Summary of Linear SVM

- Binary and linear separable classification
- Linear classifier with maximal margin
- Training SVM by maximizing

$$\max_{\alpha_i \geq 0} \{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \}$$

$$\text{subject to} \quad \alpha_i \geq 0; \quad \sum_{i=1}^{N} \alpha_i t^{(i)} = 0$$

- The weights are

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)}$$

# Summary of Linear SVM

- Binary and linear separable classification
- Linear classifier with maximal margin
- Training SVM by maximizing

$$\max_{\alpha_i \geq 0} \{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \}$$

$$\text{subject to} \quad \alpha_i \geq 0; \quad \sum_{i=1}^{N} \alpha_i t^{(i)} = 0$$

- The weights are

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)}$$

- Only a small subset of $\alpha_i$'s will be nonzero, and the corresponding $\mathbf{x}^{(i)}$'s are the support vectors $\mathbf{S}$

# Summary of Linear SVM

- Binary and linear separable classification
- Linear classifier with maximal margin
- Training SVM by maximizing

$$\max_{\alpha_i \geq 0}\{\sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j ({\mathbf{x}^{(i)}}^T \cdot \mathbf{x}^{(j)})\}$$

$$\text{subject to} \quad \alpha_i \geq 0; \quad \sum_{i=1}^{N} \alpha_i t^{(i)} = 0$$

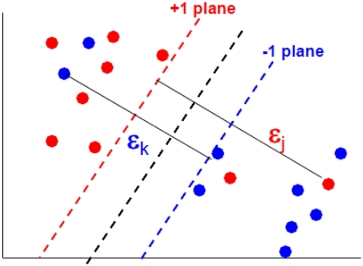
- The weights are

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)}$$

- Only a small subset of $\alpha_i$'s will be nonzero, and the corresponding $\mathbf{x}^{(i)}$'s are the support vectors **S**
- Prediction on a new example:

$$y = \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})] = \text{sign}[b + \mathbf{x} \cdot (\sum_{i \in \mathbf{S}} \alpha_i t^{(i)} \mathbf{x}^{(i)})]$$

# What if data is not linearly separable?

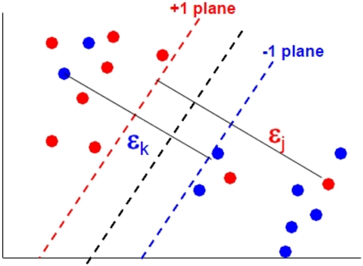



- Introduce slack variables $\xi_i$

$$\min \frac{1}{2}||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N} \xi_i$$

$$\text{s.t} \quad \xi_i \geq 0; \quad \forall i \quad t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}) \geq 1 - \xi_i$$

# What if data is not linearly separable?



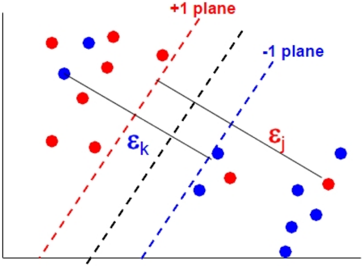

- Introduce slack variables $\xi_i$

$$\min \frac{1}{2}||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N} \xi_i$$

$$\text{s.t} \quad \xi_i \geq 0; \quad \forall i \;\; t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)}) \geq 1 - \xi_i$$

- Example lies on wrong side of hyperplane $\xi_i > 1$

# What if data is not linearly separable?



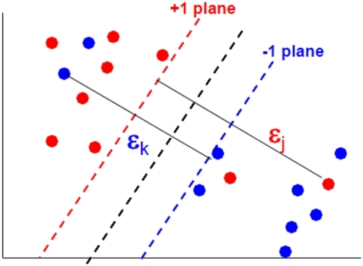

- Introduce slack variables $\xi_i$

$$\min \frac{1}{2}||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N} \xi_i$$

$$\text{s.t} \quad \xi_i \geq 0; \quad \forall i \;\; t^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \geq 1 - \xi_i$$

- Example lies on wrong side of hyperplane $\xi_i > 1$
- Therefore $\sum_i \xi_i$ upper bounds the number of training errors

# What if data is not linearly separable?



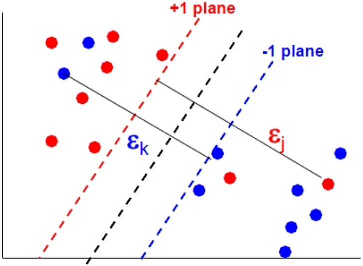

- Introduce slack variables $\xi_i$

$$\min \frac{1}{2}||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N} \xi_i$$

$$\text{s.t} \quad \xi_i \geq 0; \quad \forall i \;\; t^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \geq 1 - \xi_i$$

- Example lies on wrong side of hyperplane $\xi_i > 1$
- Therefore $\sum_i \xi_i$ upper bounds the number of training errors
- $\lambda$ trades off training error vs model complexity

# What if data is not linearly separable?




- Introduce slack variables $\xi_i$

$$\min \frac{1}{2}||\mathbf{w}||^2 + \lambda \sum_{i=1}^{N} \xi_i$$

$$\text{s.t} \quad \xi_i \geq 0; \quad \forall i \quad t^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)}) \geq 1 - \xi_i$$

- Example lies on wrong side of hyperplane $\xi_i > 1$
- Therefore $\sum_i \xi_i$ upper bounds the number of training errors
- $\lambda$ trades off training error vs model complexity
- This is known as the soft-margin extension

# Non-linear Decision Boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$
\begin{aligned}
\ell &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \\
y &= \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})]
\end{aligned}
$$

# Non-linear Decision Boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$
\begin{aligned}
\ell &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \\
y &= \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})]
\end{aligned}
$$

- How to form non-linear decision boundaries in input space?

# Non-linear Decision Boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)})$$

$$y = \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})]$$

- How to form non-linear decision boundaries in input space?
  1. Map data into feature space $\mathbf{x} \rightarrow \phi(\mathbf{x})$

# Non-linear Decision Boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)})$$

$$y = \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})]$$

- How to form non-linear decision boundaries in input space?
  1. Map data into feature space $\mathbf{x} \rightarrow \phi(\mathbf{x})$
  2. Replace dot products between inputs with feature points

$$\mathbf{x}^{(i)^T} \mathbf{x}^{(j)} \rightarrow \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

# Non-linear Decision Boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

$$
\begin{aligned}
\ell &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \\
y &= \text{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})]
\end{aligned}
$$

- How to form non-linear decision boundaries in input space?
  1. Map data into feature space $\mathbf{x} \to \phi(\mathbf{x})$
  2. Replace dot products between inputs with feature points

$$\mathbf{x}^{(i)^T} \mathbf{x}^{(j)} \to \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

  3. Find linear decision boundary in feature space

# Non-linear Decision Boundaries

- Note that both the learning objective and the decision function depend only on dot products between patterns

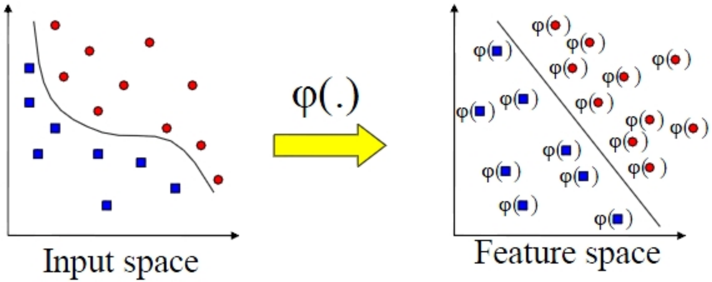
$$
\begin{aligned}
\ell &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \\
y &= \mathrm{sign}[b + \mathbf{x} \cdot (\sum_{i=1}^{N} \alpha_i t^{(i)} \mathbf{x}^{(i)})]
\end{aligned}
$$

- How to form non-linear decision boundaries in input space?
  1. Map data into feature space $\mathbf{x} \to \phi(\mathbf{x})$
  2. Replace dot products between inputs with feature points

$$
\mathbf{x}^{(i)^T} \mathbf{x}^{(j)} \to \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})
$$

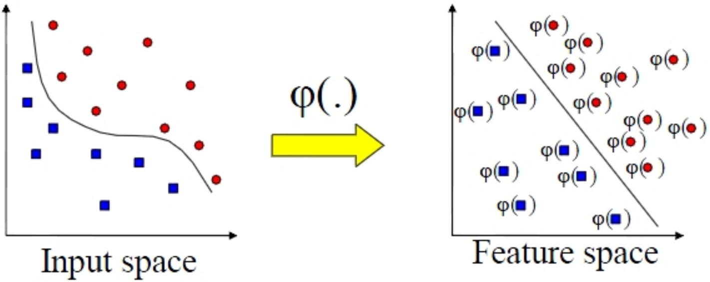
  3. Find linear decision boundary in feature space
- Problem: what is a good feature function $\phi(\mathbf{x})$?

# Input Transformation



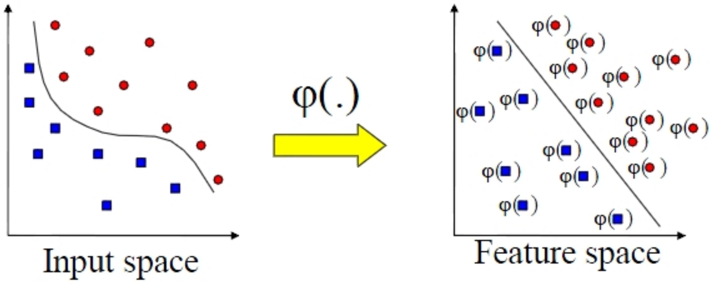

- Mapping to a feature space can produce problems:

# Input Transformation



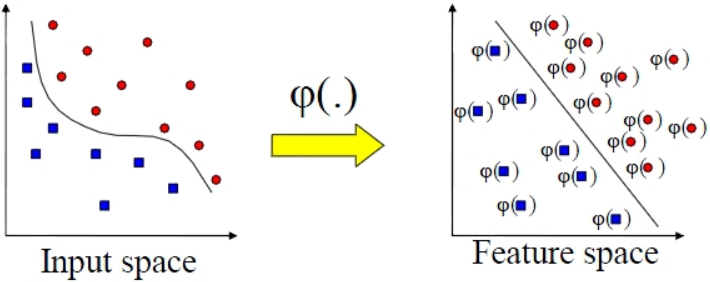

- Mapping to a feature space can produce problems:
  - High computational burden due to high dimensionality

# Input Transformation



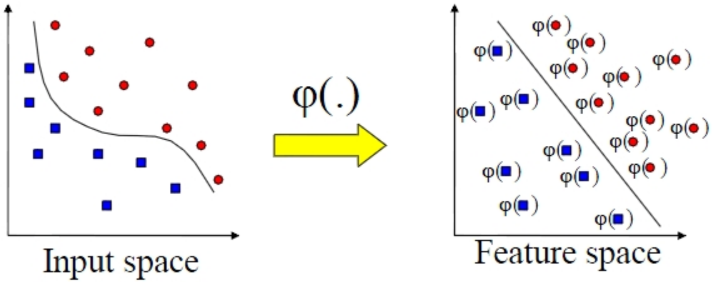

- Mapping to a feature space can produce problems:
  - High computational burden due to high dimensionality
  - Many more parameters

# Input Transformation



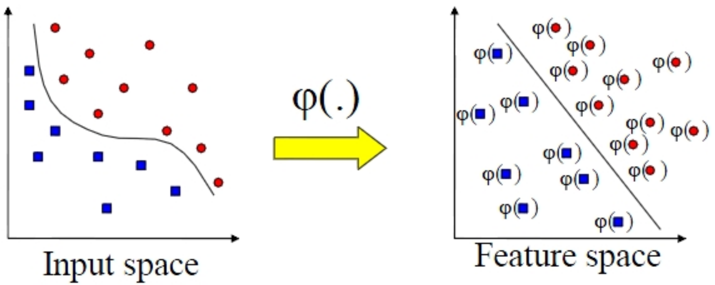

- Mapping to a feature space can produce problems:
  - High computational burden due to high dimensionality
  - Many more parameters
- SVM solves these two issues simultaneously

# Input Transformation




- Mapping to a feature space can produce problems:
  - High computational burden due to high dimensionality
  - Many more parameters
- SVM solves these two issues simultaneously
  - "Kernel trick" produces efficient classification

# Input Transformation




- Mapping to a feature space can produce problems:
  - High computational burden due to high dimensionality
  - Many more parameters
- SVM solves these two issues simultaneously
  - "Kernel trick" produces efficient classification
  - Dual formulation only assigns parameters to samples, not features

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$
- Example:

$$K(\mathbf{a}, \mathbf{b}) \quad = \quad (\mathbf{a}^T \mathbf{b})^3 =$$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$
- Example:

$$K(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^T \mathbf{b})^3 = ((a_1, a_2)^T (b_1, b_2))^3$$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$
- Example:

$$\begin{aligned} K(\mathbf{a}, \mathbf{b}) &= (\mathbf{a}^T \mathbf{b})^3 = ((a_1, a_2)^T (b_1, b_2))^3 \\ &= (a_1 b_1 + a_2 b_2)^3 \end{aligned}$$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$
- Example:

$$\begin{aligned} K(\mathbf{a}, \mathbf{b}) &= (\mathbf{a}^T\mathbf{b})^3 = ((a_1, a_2)^T(b_1, b_2))^3 \\ &= (a_1 b_1 + a_2 b_2)^3 \\ &= a_1^3 b_1^3 + 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \end{aligned}$$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$
- Example:

$$\begin{aligned}
K(\mathbf{a}, \mathbf{b}) &= (\mathbf{a}^T\mathbf{b})^3 = ((a_1, a_2)^T (b_1, b_2))^3 \\
&= (a_1 b_1 + a_2 b_2)^3 \\
&= a_1^3 b_1^3 + 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \\
&= (a_1^3, \sqrt{3}a_1^2 a_2, \sqrt{3}a_1 a_2^2, a_2^3)^T (b_1^3, \sqrt{3}b_1^2 b_2, \sqrt{3}b_1 b_2^2, b_2^3)
\end{aligned}$$

# Kernel Trick

- Kernel trick: dot-products in feature space can be computed as a kernel function

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$$

- Idea: work directly on $\mathbf{x}$, avoid having to compute $\phi(\mathbf{x})$
- Example:

$$\begin{aligned}
K(\mathbf{a}, \mathbf{b}) &= (\mathbf{a}^T \mathbf{b})^3 = ((a_1, a_2)^T (b_1, b_2))^3 \\
&= (a_1 b_1 + a_2 b_2)^3 \\
&= a_1^3 b_1^3 + 3 a_1^2 b_1^2 a_2 b_2 + 3 a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \\
&= (a_1^3, \sqrt{3} a_1^2 a_2, \sqrt{3} a_1 a_2^2, a_2^3)^T (b_1^3, \sqrt{3} b_1^2 b_2, \sqrt{3} b_1 b_2^2, b_2^3) \\
&= \phi(\mathbf{a}) \cdot \phi(\mathbf{b})
\end{aligned}$$

# Kernels

- Examples of kernels: kernels measure similarity

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T} \mathbf{x}^{(j)} + 1)^d$$
     where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
  where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$

# Kernels

- Examples of kernels: kernels measure similarity

1. Polynomial

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T} \mathbf{x}^{(j)} + 1)^d$$

where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic

2. Gaussian

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$

3. Sigmoid

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T} \mathbf{x}^{(j)}) + a)$$

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
     where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2}\right)$$
  3. Sigmoid
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}) + a)$$
- Each kernel computation corresponds to dot product

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
  where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$
  3. Sigmoid
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}) + a)$$
- Each kernel computation corresponds to dot product
  - calculation for particular mapping $\phi(\mathbf{x})$ implicitly maps to high-dimensional space

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
     where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$
  3. Sigmoid
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}) + a)$$
- Each kernel computation corresponds to dot product
  - calculation for particular mapping $\phi(\mathbf{x})$ implicitly maps to high-dimensional space
- Why is this useful?

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
     where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$
  3. Sigmoid
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}) + a)$$
- Each kernel computation corresponds to dot product
  - calculation for particular mapping $\phi(\mathbf{x})$ implicitly maps to high-dimensional space
- Why is this useful?
  1. Rewrite training examples using more complex features

# Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
     where $d$ is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$
  3. Sigmoid
     $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}) + a)$$
- Each kernel computation corresponds to dot product
  - calculation for particular mapping $\phi(\mathbf{x})$ implicitly maps to high-dimensional space
- Why is this useful?
  1. Rewrite training examples using more complex features
  2. Dataset not linearly separable in original space may be linearly separable in higher dimensional space

# Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space

# Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space
- Reasonable means that the Gram matrix is positive definite

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

# Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space
- Reasonable means that the Gram matrix is positive definite

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Feature space can be very large

# Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space
- Reasonable means that the Gram matrix is positive definite

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Feature space can be very large
  - polynomial kernel $(1 + (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)})^d$ corresponds to feature space exponential in $d$

# Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space
- Reasonable means that the Gram matrix is positive definite

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Feature space can be very large
  - polynomial kernel $(1 + (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)})^d$ corresponds to feature space exponential in $d$
  - Gaussian kernel has infinitely dimensional features

# Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space
- Reasonable means that the Gram matrix is positive definite

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Feature space can be very large
  - polynomial kernel $(1 + (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)})^d$ corresponds to feature space exponential in $d$
  - Gaussian kernel has infinitely dimensional features
- Linear separators in these super high-dim spaces correspond to highly nonlinear decision boundaries in input space

# Classification with Non-linear SVMs

- Non-linear SVM using kernel function $K()$:

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

# Classification with Non-linear SVMs

- Non-linear SVM using kernel function $K()$:

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Maximize $\ell$ w.r.t. $\{\alpha\}$ under constraints $\forall i, \alpha_i \geq 0$

# Classification with Non-linear SVMs

- Non-linear SVM using kernel function $K()$:

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Maximize $\ell$ w.r.t. $\{\alpha\}$ under constraints $\forall i, \alpha_i \geq 0$
- Unlike linear SVM, cannot express **w** as linear combination of support vectors

# Classification with Non-linear SVMs

- Non-linear SVM using kernel function $K()$:

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Maximize $\ell$ w.r.t. $\{\alpha\}$ under constraints $\forall i, \alpha_i \geq 0$
- Unlike linear SVM, cannot express $\mathbf{w}$ as linear combination of support vectors
  - now must retain the support vectors to classify new examples

# Classification with Non-linear SVMs

- Non-linear SVM using kernel function $K()$:

$$\ell = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- Maximize $\ell$ w.r.t. $\{\alpha\}$ under constraints $\forall i, \alpha_i \geq 0$
- Unlike linear SVM, cannot express **w** as linear combination of support vectors
  - now must retain the support vectors to classify new examples
- Final decision function:

$$y = \text{sign}[b + \sum_{i=1}^{N} t^{(i)} \alpha_i K(\mathbf{x}, \mathbf{x}^{(i)})]$$

# Summary

- Advantages:

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples
  - Variable-sized hypothesis space

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples
  - Variable-sized hypothesis space
  - Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples
  - Variable-sized hypothesis space
  - Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)
- Disadvantages:

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples
  - Variable-sized hypothesis space
  - Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)
- Disadvantages:
  - Must choose kernel parameters

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples
  - Variable-sized hypothesis space
  - Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)
- Disadvantages:
  - Must choose kernel parameters
  - Very large problems computationally intractable

# Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Poly-time exact optimization methods rather than approximate methods
  - Soft-margin extension permits mis-classified examples
  - Variable-sized hypothesis space
  - Excellent results (1.1% error rate on handwritten digits vs. LeNet's 0.9%)
- Disadvantages:
  - Must choose kernel parameters
  - Very large problems computationally intractable
  - Batch algorithm

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available
- Key points:

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available
- Key points:
  - Difference between logistic regression and SVMs

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available
- Key points:
  - Difference between logistic regression and SVMs
  - Maximum margin principle

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available
- Key points:
  - Difference between logistic regression and SVMs
  - Maximum margin principle
  - Target function for SVMs

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available
- Key points:
  - Difference between logistic regression and SVMs
  - Maximum margin principle
  - Target function for SVMs
  - Slack variables for mis-classified points

# More Summary

- Software:
  - A list of SVM implementations can be found at `http://www.kernel-machines.org/software.html`
  - Some implementations (such as LIBSVM) can handle multi-class classification
  - SVMLight is among the earliest implementations
  - Several Matlab toolboxes for SVM are also available
- Key points:
  - Difference between logistic regression and SVMs
  - Maximum margin principle
  - Target function for SVMs
  - Slack variables for mis-classified points
  - Kernel trick allows non-linear generalizations