Kernels and the Kernel Trick

Machine Learning

THE UNIVERSITY OF UTAH

Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

Support vector machines

- Training by maximizing margin
- The SVM objective
- Solving the SVM optimization problem
- Support vectors, duals and kernels

This lecture

1. Support vectors

2. Kernels

3. The kernel trick

4. Properties of kernels

5. Another example of the kernel trick

This lecture

1. **Support vectors**
2. Kernels
3. The kernel trick
4. Properties of kernels
5. Another example of the kernel trick

So far we have seen

- Support vector machines

- Hinge loss and optimizing the regularized loss

More broadly, different algorithms for learning linear classifiers

So far we have seen

- Support vector machines

- Hinge loss and optimizing the regularized loss

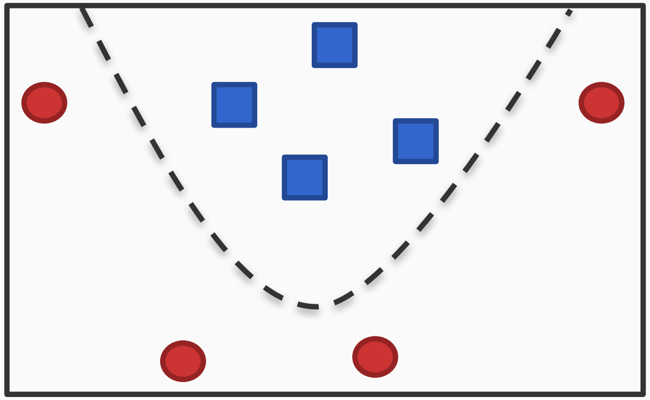
More broadly, different algorithms for learning linear classifiers

What about non-linear models?

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

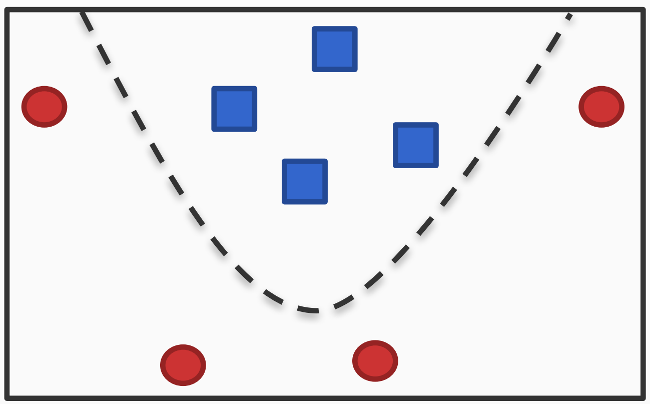
If the true separator is quadratic



One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



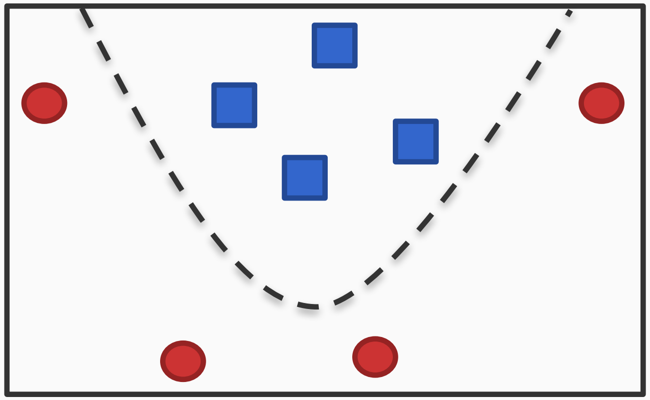
Transform all input points as

$$\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



Transform all input points as

$$\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using $\mathbf{w}^T \phi(x_1, x_2) \geq b$

SVM: Primals and duals

The SVM objective

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\ \text{s.t.} \quad & \forall i, & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i \\ & \forall i, & \xi_i \geq 0. \end{aligned}$$

This is called the *primal form* of the objective

This can be converted to its *dual form*, which will let us prove a very useful property

SVM: Primals and duals

The SVM objective

$$
\begin{aligned}
\min_{\mathbf{w},\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\
\text{s.t.} \quad \forall i, \quad & y_i \mathbf{w}^T \mathbf{x}_i \geq 1 - \xi_i \\
\forall i, \quad & \xi_i \geq 0.
\end{aligned}
$$

Another optimization problem

Has the property that max Dual = min Primal

This is called the *primal form* of the objective

This can be converted to its *dual form*, which will let us prove a very useful property

Support vector machines

$$\begin{aligned} \min_{\mathbf{w},\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_i \xi_i \\ \text{s.t.} \quad & \forall i, \quad y_i \mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i \\ & \forall i, \quad \xi_i \geq 0. \end{aligned}$$

Let **w** be the minimizer of the SVM problem for some dataset with m examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for i = 1…m, there exist $\alpha_i \geq 0$ such that the optimum w can be written as

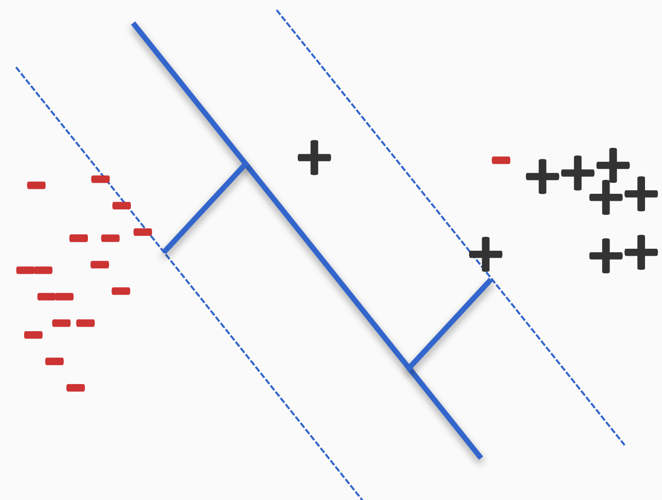
$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

Support vector machines

Let **w** be the minimizer of the SVM problem for some dataset with m examples: {($\mathbf{x}_i$, y_i)}

Then, for i = 1…m, there exist $\alpha_i \geq 0$ such that the optimum w can be written as

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



Furthermore,

$\alpha_i = 0 \quad\quad \Rightarrow y_i \mathbf{w}^T x_i \geq 1$

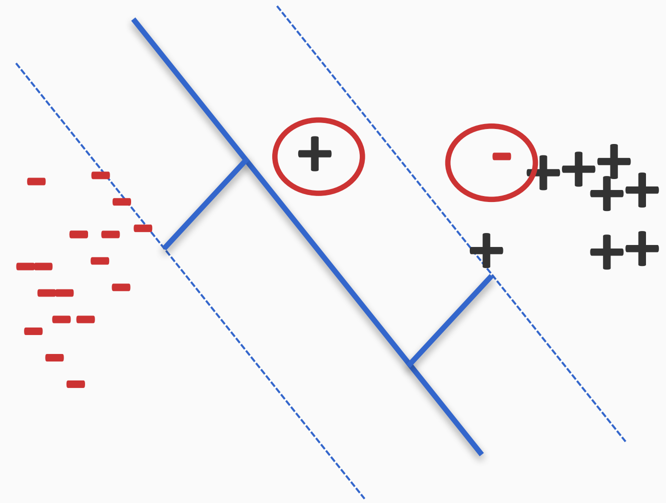
All points outside the margin

Support vector machines

Let **w** be the minimizer of the SVM problem for some dataset with m examples: {($\mathbf{x}_i$, y_i)}

Then, for i = 1…m, there exist $\alpha_i \geq 0$ such that the optimum w can be written as

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



Furthermore,

$$\alpha_i = 0 \qquad \Rightarrow y_i \mathbf{w}^T x_i \geq 1$$

$$\alpha_i = C \qquad \Rightarrow y_i \mathbf{w}^T x_i \leq 1$$

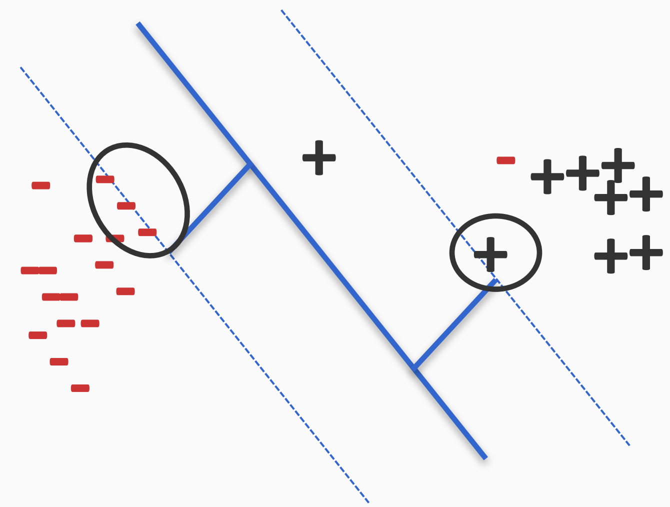
All points on the wrong side of the margin

Support vector machines

Let **w** be the minimizer of the SVM problem for some dataset with m examples: $\{(\mathbf{x}_i, y_i)\}$

Then, for i = 1…m, there exist $\alpha_i \geq 0$ such that the optimum w can be written as

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



Furthermore,

$$\alpha_i = 0 \quad \Rightarrow y_i \mathbf{w}^T x_i \geq 1$$

$$\alpha_i = C \quad \Rightarrow y_i \mathbf{w}^T x_i \leq 1$$

$$0 \leq \alpha_i \leq C \quad \Rightarrow y_i \mathbf{w}^T x_i = 1$$

All points on the margin

Support vectors

The weight vector is completely defined by training examples whose α_is are not zero

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

These examples are called the *support vectors*

This lecture

✓ Support vectors

2. **Kernels**

3. The kernel trick

4. Properties of kernels

5. Another example of the kernel trick

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- That is, we just showed that $\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \mathbf{x}_i^T \mathbf{x}$
  - We only need to compute dot products between training examples and the new example **x**

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- That is, we just showed that $\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \boxed{\mathbf{x}_i^T \mathbf{x}}$
  - We only need to compute dot products between training examples and the new example **x**

Predicting with linear classifiers

- Prediction = $sgn(\mathbf{w}^T\mathbf{x})$ and $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$
- That is, we just showed that $\mathbf{w}^T\mathbf{x} = \sum_i \alpha_i y_i \boxed{\mathbf{x}_i^T\mathbf{x}}$
  - We only need to compute dot products between training examples and the new example **x**
- This is true even if we map examples to a high dimensional space

$$\mathbf{w}^T\phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T\phi(\mathbf{x})$$

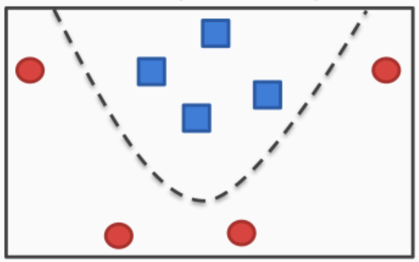
Predicting with linear classifiers

- Predic
- That i
  - Tha ... ween trai
- This is dimen

One way to learn non-linear models

Explicitly introduce non-linearity into the feature space

If the true separator is quadratic



Transform all input points as

$$\phi(x_1, x_2) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Now, we can try to find a weight vector in this higher dimensional space

That is, predict using $\mathbf{w}^T\phi(x_1, x_2) \geq b$

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

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

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

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

So prediction with this *high dimensional lifting map* is

$$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

Dot products in high dimensional spaces

Let us define a dot product in the high dimensional space

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

So prediction with this *high dimensional lifting map* is

$$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

because $\mathbf{w}^T \phi(\mathbf{x}) = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$

Kernel based methods

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

Predict using $$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

What does this new formulation give us?

If we have to compute ϕ every time anyway, we gain nothing

Kernel based methods

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

Predict using $$sgn(\mathbf{w}^T \phi(\mathbf{x})) = sgn\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x})\right)$$

What does this new formulation give us?

If we have to compute ϕ every time anyway, we gain nothing

If we can compute the value of K without explicitly writing the blown up representation, then we will have a computational advantage.

This lecture

✓ Support vectors

✓ Kernels

3. **The kernel trick**

4. Properties of kernels

5. Another example of the kernel trick

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

All degree zero terms

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, \underbrace{x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n}_{\text{All degree two terms}}]^T$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [\underbrace{1}_{\text{All degree zero terms}}, \underbrace{x_1, x_2, \cdots, x_n}_{\text{All degree one terms}}, \underbrace{x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n}_{\text{All degree two terms}}]^T$$

and compute the dot product A $= \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product $A = \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1x_2, \cdots, x_{n-1}x_n]^T$$

and compute the dot product A = $\phi(\mathbf{x})^T\phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product A $= \phi(\mathbf{x})^T \phi(\mathbf{z})$ [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Example: Polynomial Kernel

- Given two examples **x** and **z** we want to map them to a high dimensional space [for example, quadratic]

$$\phi(x_1, x_2, \cdots, x_n) = [1, x_1, x_2, \cdots, x_n, x_1^2, x_2^2, \cdots x_n^2, x_1 x_2, \cdots, x_{n-1} x_n]^T$$

and compute the dot product A = ϕ(**x**)^Tϕ (**z**) [takes time]

- Instead, in the original space, compute

$$B = K(\mathbf{x}, \mathbf{z}) = \left(1 + \mathbf{x}^T \mathbf{z}\right)^2$$

Claim: A = B (Coefficients do not really matter)

Example: Two dimensions, quadratic kernel

A = $\phi(\mathbf{x})^T\phi(\mathbf{z})$ $\quad$ $B = K(\mathbf{x},\mathbf{z}) = \left(1+\mathbf{x}^T\mathbf{z}\right)^2$

$$\phi(x_1, x_2) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$$

The Kernel Trick

Suppose we wish to compute $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$

Here ϕ maps **x** and **z** to a high dimensional space

The Kernel Trick: Save time/space by computing the value of $K(\mathbf{x}, \mathbf{z})$ by performing operations in the original space (without a feature transformation!)

Computing dot products efficiently

Kernel Trick: You want to work with degree 2 polynomial features, ϕ(x). Then, your dot product will be operate using vectors in a space of dimensionality n(n+1)/2.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

This lecture

✓ Support vectors

✓ Kernels

✓ The kernel trick

4. Properties of kernels

5. Another example of the kernel trick

Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality n(n+1)/2.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function K(.,.)?

Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality n(n+1)/2.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function K(.,.)?
  - No! A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

Which functions are kernels?

Kernel Trick: You want to work with degree 2 polynomial features, $\phi(x)$. Then, your dot product will be operate using vectors in a space of dimensionality $n(n+1)/2$.

The kernel trick allows you to save time/space and compute dot products in an n dimensional space.

(Not just for degree 2 polynomials)

- Can we use any function K(.,.)?
  - No! A function K(x,z) is a valid kernel if it corresponds to an inner product in some (perhaps infinite dimensional) feature space.

- General condition: construct the Gram matrix $\{K(\mathbf{x}_i, \mathbf{z}_j)\}$; check that it's positive semi definite

Reminder: Positive semi-definite matrices

A symmetric matrix M is positive semi-definite if it is

- For any vector non-zero **z**, we have $\mathbf{z}^T M \mathbf{z} \geq 0$

(A useful property characterizing many interesting mathematical objects)

The Kernel Matrix

- The Gram matrix of a set of n vectors S = $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ is the $n \times n$ matrix $\mathbf{G}$ with $\mathbf{G}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
  - The kernel matrix is the Gram matrix of $\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$
  - (size depends on the # of examples, not dimensionality)

The Kernel Matrix

- The Gram matrix of a set of n vectors S = $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ is the $n \times n$ matrix $\mathbf{G}$ with $\mathbf{G}_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
  - The kernel matrix is the Gram matrix of $\{\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)\}$
  - (size depends on the # of examples, not dimensionality)

- Showing that a function K is a valid kernel
  - Direct approach: If you have the $\phi(\mathbf{x}_i)$, you have the Gram matrix (and it's easy to see that it will be positive semi-definite). *Why?*
  - Indirect: If you have the Kernel, write down the Kernel matrix K_{ij}, and show that it is a legitimate kernel, without an explicit construction of $\phi(\mathbf{x}_i)$

Mercer's condition

Let K($\mathbf{x}$, $\mathbf{z}$) be a function that maps two n dimensional vectors to a real number

K is a valid kernel if for every finite set $\{x_1, x_2, \cdots\}$, for any choice of real valued $c_1, c_2, \cdots$, we have

$$\sum_i \sum_j c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Polynomial kernels

- Linear kernel: $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z}$

- Polynomial kernel of degree *d*: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z})^d$
  - only *d*th-order interactions

- Polynomial kernel up to degree *d*: $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^d$ ($c>0$)
  - all interactions of order *d* or lower

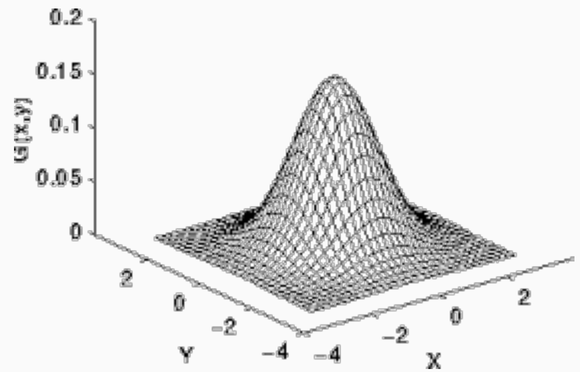
Gaussian Kernel

(or the radial basis function kernel)

$$K_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{c}\right)$$

– $(x - z)^2$: squared Euclidean distance between **x** and **z**
– $c = \sigma^2$: a free parameter
– very small c: K ≈ identity matrix (every item is different)
– very large c: K ≈ unit matrix (all items are the same)

– k(**x**, **z**) ≈ 1 when **x**, **z** close
– k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar



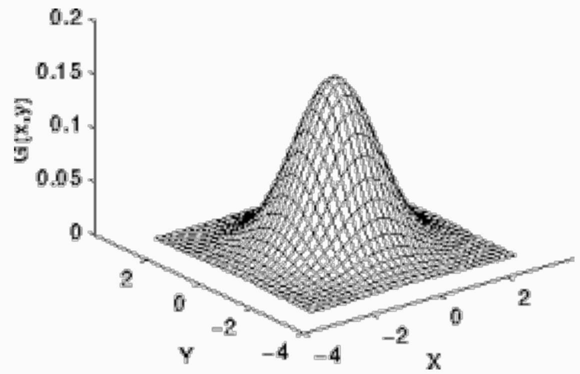
Gaussian Kernel

(or the radial basis function kernel)

$$K_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{c}\right)$$

– $(x - z)^2$: squared Euclidean distance between **x** and **z**
– $c = \sigma^2$: a free parameter
– very small c: K ≈ identity matrix (every item is different)
– very large c: K ≈ unit matrix (all items are the same)

– k(**x**, **z**) ≈ 1 when **x**, **z** close
– k(**x**, **z**) ≈ 0 when **x**, **z** dissimilar



Exercises:
1. Prove that this is a kernel.
2. What is the "blown up" feature space for this kernel?

Constructing New Kernels

You can construct new kernels $k'(\mathbf{x}, \mathbf{x'})$ from existing ones:

- Multiplying $k(\mathbf{x}, \mathbf{x'})$ by a constant c

  $ck(\mathbf{x}, \mathbf{x'})$

- Multiplying $k(\mathbf{x}, \mathbf{x'})$ by a function f applied to $\mathbf{x}$ and $\mathbf{x'}$

  $f(\mathbf{x})k(\mathbf{x}, \mathbf{x'})f(\mathbf{x'})$

- Applying a polynomial (with non-negative coefficients) to $k(\mathbf{x}, \mathbf{x'})$

  $P(\ k(\mathbf{x}, \mathbf{x'})\ )$ with $P(z) = \sum_i a_i z^i$ and $a_i \geq 0$

- Exponentiating $k(\mathbf{x}, \mathbf{x'})$

  $\exp(k(\mathbf{x}, \mathbf{x'}))$

Constructing New Kernels (2)

- You can construct $k'(\mathbf{x}, \mathbf{x'})$ from $k_1(\mathbf{x}, \mathbf{x'})$, $k_2(\mathbf{x}, \mathbf{x'})$ by:
  - Adding $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'}) + k_2(\mathbf{x}, \mathbf{x'})$
  - Multiplying $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'})k_2(\mathbf{x}, \mathbf{x'})$

Constructing New Kernels (2)

- You can construct $k'(\mathbf{x}, \mathbf{x'})$ from $k_1(\mathbf{x}, \mathbf{x'})$, $k_2(\mathbf{x}, \mathbf{x'})$ by:
  - Adding $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'}) + k_2(\mathbf{x}, \mathbf{x'})$
  - Multiplying $k_1(\mathbf{x}, \mathbf{x'})$ and $k_2(\mathbf{x}, \mathbf{x'})$:
    $k_1(\mathbf{x}, \mathbf{x'})k_2(\mathbf{x}, \mathbf{x'})$
- Also:
  - If $\phi(\mathbf{x}) \in R^m$ and $k_m(\mathbf{z}, \mathbf{z'})$ a valid kernel in R^m,
    $k(\mathbf{x}, \mathbf{x}') = k_m(\phi(\mathbf{x}), \phi(\mathbf{x}'))$ is also a valid kernel
  - If $\mathbf{A}$ is a symmetric positive semi-definite matrix,
    $k(\mathbf{x}, \mathbf{x}') = \mathbf{xAx}'$ is also a valid kernel

Kernel Trick: An example

Let the blown up feature space represent the space of all 3^n conjunctions. Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

This lecture

- ✓ Support vectors
- ✓ Kernels
- ✓ The kernel trick
- ✓ Properties of kernels

5. Another example of the kernel trick

Kernel Trick: An example

Let the blown up feature space represent the space of all 3^n conjunctions. Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

Example: Take n=3; **x**=(001), **z**=(011), we have conjunctions of size 0,1,2,3

Kernel Trick: An example

Let the blown up feature space represent the space of all 3^n conjunctions. Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

Example: Take n=3; **x**=(001), **z**=(011), we have conjunctions of size 0,1,2,3

Proof: let m=same(**x**, **z**); construct “surviving” conjunctions by

Kernel Trick: An example

Let the blown up feature space represent the space of all 3^n conjunctions. Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

Example: Take n=3; **x**=(001), **z**=(011), we have conjunctions of size 0,1,2,3

Proof: let m=same(**x**, **z**); construct “surviving” conjunctions by

1. choosing to include one of these k literals with the right polarity in the conjunctions, or

Kernel Trick: An example

Let the blown up feature space represent the space of all 3^n conjunctions. Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

Example: Take n=3; **x**=(001), **z**=(011), we have conjunctions of size 0,1,2,3

Proof: let m=same(**x**, **z**); construct "surviving" conjunctions by

1. choosing to include one of these k literals with the right polarity in the conjunctions, or
2. choosing to not include it at all.

Kernel Trick: An example

Let the blown up feature space represent the space of all 3^n conjunctions. Then,

$$K(\mathbf{x}, \mathbf{z}) = \sum_i \phi_i(\mathbf{x})\phi_i(\mathbf{z}) = 2^{same(\mathbf{x},\mathbf{z})}$$

where same(x,z) is the number of features that have the same value for both x and z

Example: Take n=3; **x**=(001), **z**=(011), we have conjunctions of size 0,1,2,3

Proof: let m=same(**x**, **z**); construct "surviving" conjunctions by

1. choosing to include one of these k literals with the right polarity in the conjunctions, or
2. choosing to not include it at all.

Conjunctions with literals outside this set disappear.

Exercises

1. Show that this argument works for a specific example
   - Take $X=\{x_1, x_2, x_3, x_4\}$
   - $\phi(\mathbf{x})$ = The space of all 3^n conjunctions ; $|\phi(\mathbf{x})| = 81$
   - Consider $\mathbf{x}=(1100)$, $\mathbf{z}=(1101)$
   - Write $\phi(\mathbf{x})$, $\phi(\mathbf{z})$, the representation of **x**, **z** in the ϕ space
   - Compute $\phi(\mathbf{x})^T\phi(\mathbf{z})$
   - Show that

     $$K(x,z) = \phi(\mathbf{x})^T\phi(\mathbf{z}) = \sum_i \phi_i(z)\, \phi_i(x) = 2^{same(\mathbf{x},\mathbf{z})} = 8$$

2. Try to develop another kernel, e.g., where the space of all conjunctions of size 3 (exactly)

Summary: Kernel trick

- To make the final prediction, we are computing dot products

- The kernel trick is a computational trick to compute dot products in higher dimensional spaces

- This is applicable not just to SVMs. The same idea can be extended to Perceptron too: the Kernel Perceptron

- Important: All the bounds we have seen (eg: Perceptron bound, etc) depend on the underlying dimensionality
  - By moving to a higher dimensional space, we are incurring a penalty on sample complexity