CS446 Introduction to Machine Learning (Fall 2015)
University of Illinois at Urbana-Champaign
http://courses.engr.illinois.edu/cs446

LECTURE 8: DUAL AND KERNELS

Prof. Julia Hockenmaier
juliahmr@illinois.edu

Admin

Reminder: Homework Late Policy

Everybody is allowed a total of two late days for the semester.

If you have exhausted your contingent of late days, we will subtract 20% per late day.

We don’t accept assignments more than two days after their due date.

Let us know if there are any special circumstances (family, health, etc.)

Convergence checks

What does it mean for $\mathbf{w}$ to have converged?

- Define a convergence threshold τ (e.g. 10^{-3})
- Compute $\Delta\mathbf{w}$, the difference between $\mathbf{w}^{old}$ and $\mathbf{w}^{new}$: $\Delta\mathbf{w} = \mathbf{w}^{old} - \mathbf{w}^{new}$
- $\mathbf{w}$ has converged when $\| \Delta\mathbf{w} \| < \tau$

Convergence checks

How often do I check for convergence?

Batch learning:
$\mathbf{w}^{\text{old}} = \mathbf{w}$ before seeing the current batch
$\mathbf{w}^{\text{new}} = \mathbf{w}$ after seeing the current batch

Assuming your batch is large enough, this works well.

Convergence checks

How often do I check for convergence?

Online learning:

– **Problem:** A single example may only lead to very small changes in $\mathbf{w}$
– **Solution:** Only check for convergence after every *k* examples (or updates, doesn't matter).

$\mathbf{w}^{\text{old}} = \mathbf{w}$ after $n \cdot k$ examples/updates
$\mathbf{w}^{\text{new}} = \mathbf{w}$ after $(n+1) \cdot k$ examples/updates

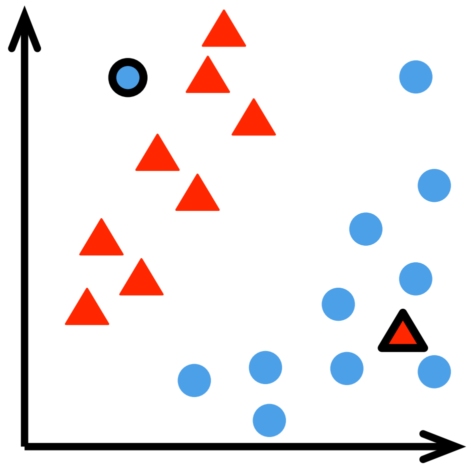
Back to linear classifiers….

Linear classifiers so far…

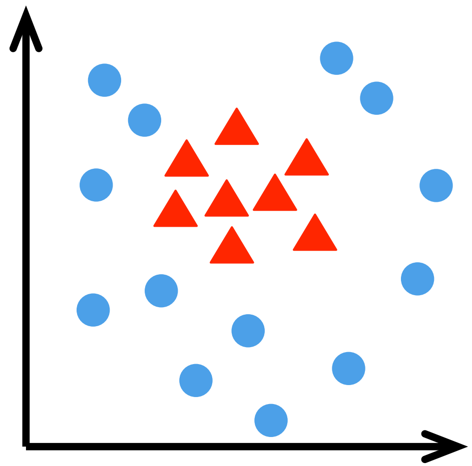
What we've seen so far is not the whole story

- We've assumed that the data are linearly separable
- We've ignored the fact that the perceptron just finds *some* decision boundary, but not necessarily *an optimal* decision boundary

Data are not linearly separability



Noise / outliers



Target function
is not linear in X

Today's key concepts

Kernel trick:

Dealing with target functions that are not linearly separable.

This requires us to move to the dual representation.

Dual representation of linear classifiers

Dual representation

Recall the Perceptron update rule:

If $\mathbf{x_m}$ *is misclassified, add* $y_m \cdot \mathbf{x_m}$ to $\mathbf{w}$

if $y_m \cdot f(\mathbf{x_m}) = y_m \cdot \mathbf{w} \cdot \mathbf{x_m} < 0$:

$$\mathbf{w} := \mathbf{w} + y_m \cdot \mathbf{x_m}$$

Dual representation:

Write $\mathbf{w}$ as a weighted sum of training items:

$$\mathbf{w} = \sum_n \alpha_n y_n \mathbf{x}_n$$

α_n: how often was $\mathbf{x}_n$ misclassified?

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_n \alpha_n y_n \mathbf{x}_n \cdot \mathbf{x}$$

Dual representation

Primal Perceptron update rule:

If $\mathbf{x_m}$ *is misclassified, add* $y_m \cdot \mathbf{x_m}$ to $\mathbf{w}$

if $y_m \cdot f(\mathbf{x_m}) = y_m \cdot \mathbf{w} \cdot \mathbf{x_m} < 0$:

$\mathbf{w} := \mathbf{w} + y_m \cdot \mathbf{x_m}$

Dual Perceptron update rule:

If $\mathbf{x_m}$ *is misclassified, add* 1 to α_m

if $y_m \cdot \sum_d \alpha_d \mathbf{x_d} \cdot \mathbf{x_m} < 0$:

$\alpha_m := \alpha_m + 1$

Dual representation

Classifying $\mathbf{x}$ in the primal: $f(\mathbf{x}) = \mathbf{w}\,\mathbf{x}$

$\mathbf{w}$ = feature weights (to be learned)

$\mathbf{w}\mathbf{x}$ = dot product between $\mathbf{w}$ and $\mathbf{x}$

Classifying $\mathbf{x}$ in the dual: $f(\mathbf{x}) = \sum_n \alpha_n y_n \mathbf{x}_n \mathbf{x}$

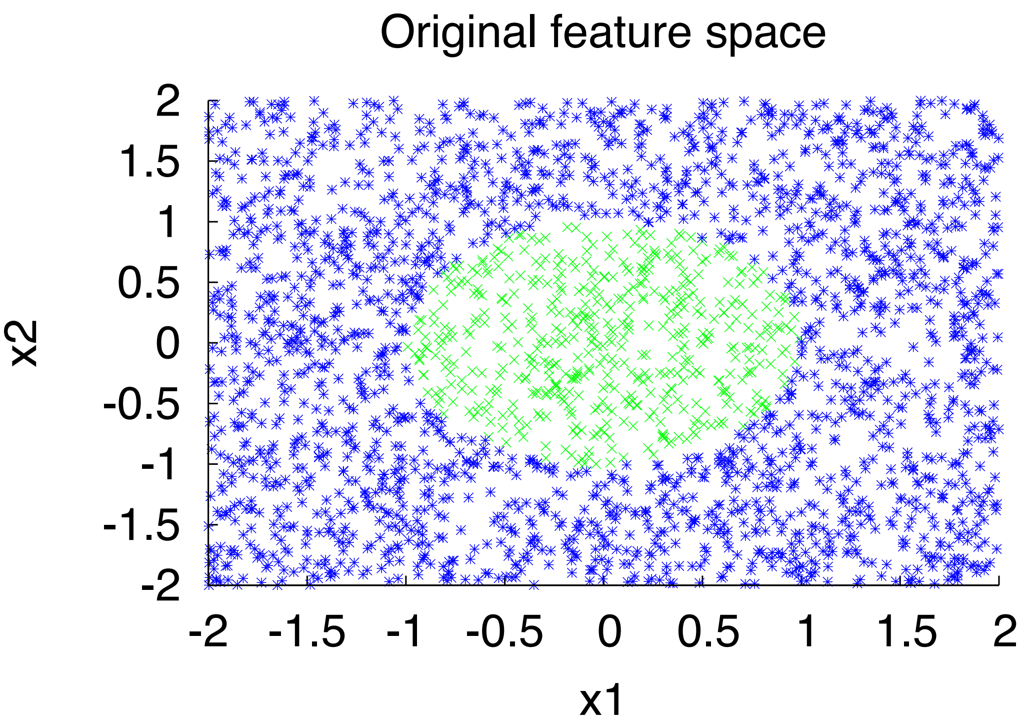
α_n = weight of n-th training example (to be learned)

$\mathbf{x}_n \mathbf{x}$ = dot product between $\mathbf{x}_n$ and $\mathbf{x}$

The dual representation is advantageous when #training examples ≪ #features (requires fewer parameters to learn)

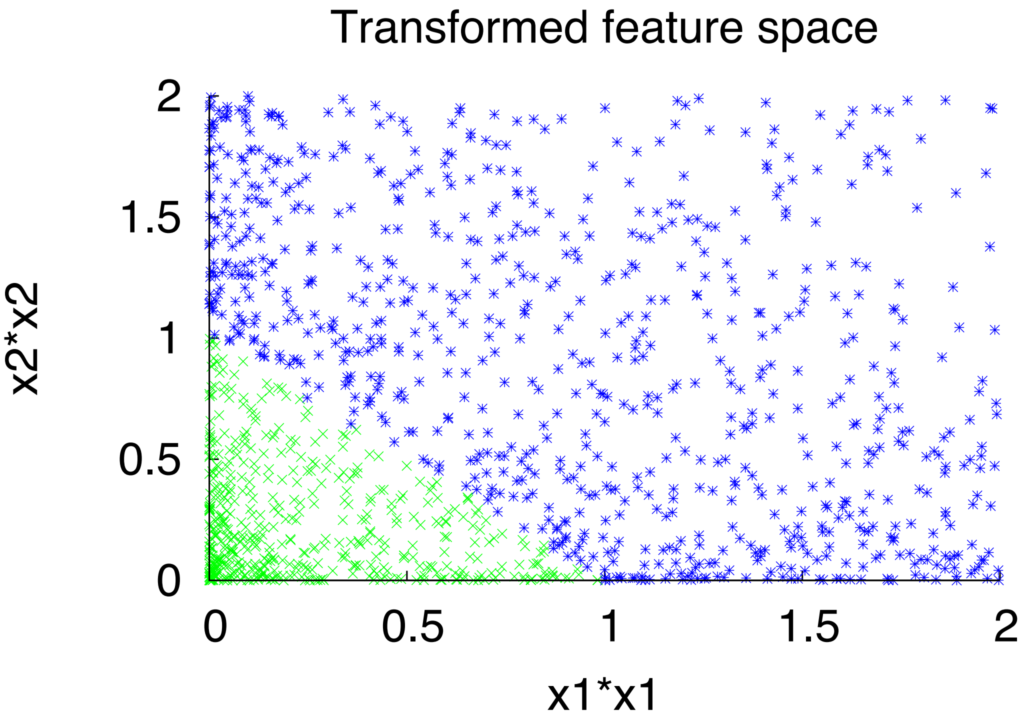
Kernels

Making data linearly separable



$$f(\mathbf{x}) = 1 \text{ iff } x_1^2 + x_2^2 \leq 1$$

Making data linearly separable

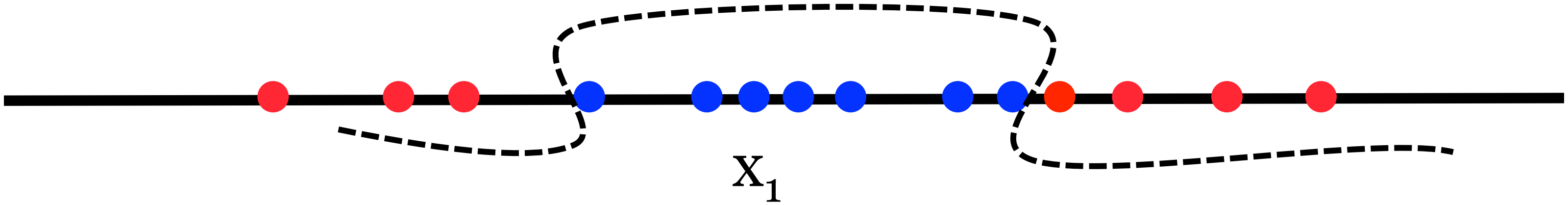


Transform data:

$\mathbf{x} = (x_1, x_2) \Rightarrow \mathbf{x'} = (x_1^2, x_2^2)$

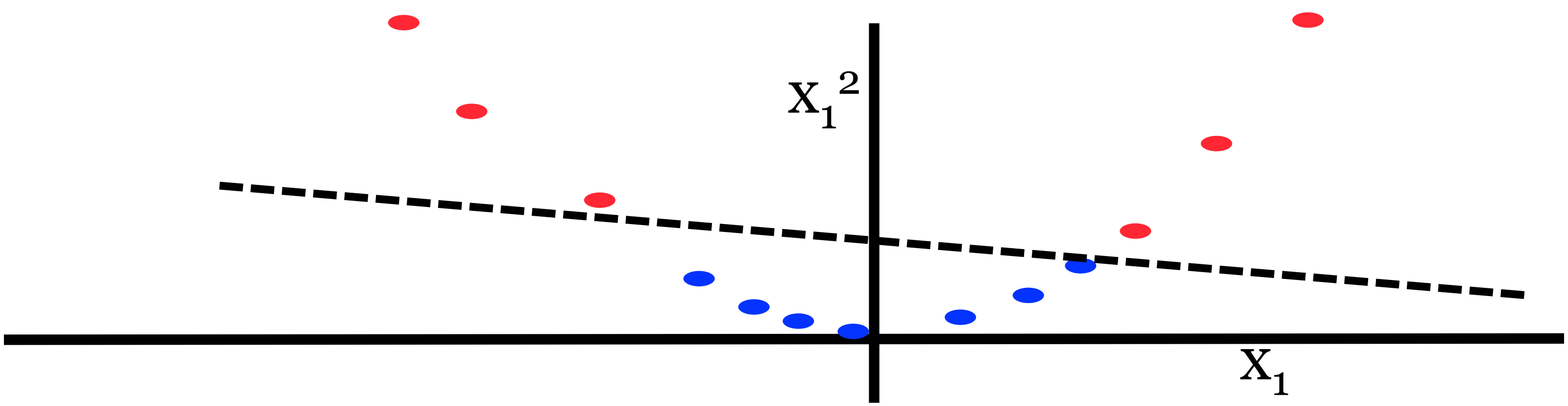
$f(\mathbf{x'}) = 1$ iff $x'_1 + x'_2 \leq 1$

Making data linearly separable



These data aren't linearly separable in the x_1 space

But adding a second dimension with $x_2 = x_1^2$ makes them linearly separable in $\langle x_1, x_2 \rangle$:



Making data linearly separable

It is common for data to be not linearly separable in the original feature space.

We can often introduce new features to make the data linearly separable in the new space:

- *transform* the original features (e.g. $x \rightarrow x^2$)
- include transformed features in addition to the original features
- capture *interactions* between features (e.g. $x_3 = x_1 x_2$)

But this may blow up the number of features

Making data linearly separable

We need to introduce a lot of new features to learn the target function.

Problem for the primal representation:
$\mathbf{w}$ now has a lot of elements, and we might not have enough data to learn $\mathbf{w}$

The dual representation is not affected

The kernel trick

- Define a feature function $\varphi(\mathbf{x})$ which maps items $\mathbf{x}$ into a higher-dimensional space.
- The kernel function $K(\mathbf{x}^i, \mathbf{x}^j)$ computes the inner product between the $\varphi(\mathbf{x}^i)$ and $\varphi(\mathbf{x}^j)$

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

- Dual representation: We don't need to learn $\mathbf{w}$ in this higher-dimensional space. It is sufficient to evaluate $K(\mathbf{x}^i, \mathbf{x}^j)$

Quadratic kernel

Original features: $\mathbf{x} = (a, b)$

Transformed features: $\varphi(\mathbf{x}) = (a^2, b^2, \sqrt{2} \cdot ab)$

Dot product in transformed space:

$$\begin{aligned}\varphi(\mathbf{x_1}) \cdot \varphi(\mathbf{x_2}) &= a_1^2 a_2^2 + b_1^2 b_2^2, 2 \cdot a_1 b_1 a_2 b_2 \\ &= (\mathbf{x_1} \cdot \mathbf{x_2})^2\end{aligned}$$

Kernel:

$$K(\mathbf{x_1}, \mathbf{x_2}) = (\mathbf{x_1} \cdot \mathbf{x_2})^2 = \varphi(\mathbf{x_1}) \cdot \varphi(\mathbf{x_2})$$

Polynomial kernels

Polynomial kernel of degree p:

- Basic form $K(\mathbf{x_i},\mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j})^p$
- Standard form (captures all lower order terms):
  $K(\mathbf{x_i},\mathbf{x_j}) = (\mathbf{x_i} \cdot \mathbf{x_j} + 1)^p$

From dual to kernel perceptron

Dual Perceptron: $f(\mathbf{x}) = \sum_d \alpha_d \mathbf{x}_d \cdot \mathbf{x}_\mathbf{m}$

Update: *If* $\mathbf{x}_\mathbf{m}$ *is misclassified, add* 1 to α_m

$$\textbf{if } y_m \cdot \sum_d \alpha_d \mathbf{x}_d \cdot \mathbf{x}_\mathbf{m} < 0: \quad \alpha_m := \alpha_m + 1$$

Kernel Perceptron: $f(\mathbf{x}) = \sum_d \alpha_d \varphi(\mathbf{x}_d) \cdot \varphi(\mathbf{x}_\mathbf{m}) = \sum_d \alpha_d K(\mathbf{x}_d \cdot \mathbf{x}_\mathbf{m})$

Update: *If* $\mathbf{x}_\mathbf{m}$ *is misclassified, add* 1 to α_m

$$\textbf{if } y_m \cdot \sum_d \alpha_d K(\mathbf{x}_d \cdot \mathbf{x}_\mathbf{m}) < 0: \quad \alpha_m := \alpha_m + 1$$

Primal and dual representation

Linear classifier (primal representation):

$\mathbf{w}$ defines weights of features of $\mathbf{x}$

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

Linear classifier (dual representation):

Rewrite $\mathbf{w}$ as a (weighted) sum of training items:

$$\mathbf{w} = \sum_n \alpha_n y_n \mathbf{x}_n$$

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_n \alpha_n y_n \mathbf{x}_n \cdot \mathbf{x}$$

The kernel trick

– Define a feature function $\varphi(\mathbf{x})$ which maps items $\mathbf{x}$ into a higher-dimensional space.
– The kernel function $K(\mathbf{x}^i, \mathbf{x}^j)$ computes the inner product between the $\varphi(\mathbf{x}^i)$ and $\varphi(\mathbf{x}^j)$

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

– Dual representation: We don't need to learn $\mathbf{w}$ in this higher-dimensional space. It is sufficient to evaluate $K(\mathbf{x}^i, \mathbf{x}^j)$

The kernel matrix

The kernel matrix of a data set D = $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ defined by a kernel function $k(\mathbf{x}, \mathbf{z}) = \varphi(\mathbf{x})\varphi(\mathbf{z})$ is the $n \times n$ matrix $\mathbf{K}$ with $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

You'll also find the term 'Gram matrix' used:

- 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 $\mathbf{G}_{ij} = \mathbf{x}_i \mathbf{x}_j$
- The kernel matrix is the Gram matrix of $\{\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_n)\}$

Properties of the kernel matrix **K**

K is symmetric:

$\mathbf{K}_{\mathrm{ij}} = k(\mathbf{x}_{\mathrm{i}}, \mathbf{x}_{\mathrm{j}}) = \varphi(\mathbf{x}_{\mathrm{i}})\varphi(\mathbf{x}_{\mathrm{j}}) = k(\mathbf{x}_{\mathrm{j}}, \mathbf{x}_{\mathrm{i}}) = \mathbf{K}_{\mathrm{ji}}$

K is positive semi-definite ($\forall$ vectors $\mathbf{v}$: $\mathbf{v}^{\mathrm{T}}\mathbf{K}\mathbf{v} \geq 0$):

Proof:
$$\begin{aligned}
\mathbf{v}^T\mathbf{K}\mathbf{v} &= \sum_{i=1}^{D}\sum_{j=1}^{D} v_i v_j K_{ij} = \sum_{i=1}^{D}\sum_{j=1}^{D} v_i v_j \left\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j)\right\rangle \\
&= \sum_{i=1}^{D}\sum_{j=1}^{D} v_i v_j \sum_{k=1}^{N} \phi_k(\mathbf{x}_i)\cdot\phi_k(\mathbf{x}_j) = \sum_{k=1}^{N}\sum_{i=1}^{D}\sum_{j=1}^{D} v_i\phi_k(\mathbf{x}_i)\cdot v_j\phi_k(\mathbf{x}_j) \\
&= \sum_{k=1}^{N}\left(\sum_{i=1}^{D} v_i\phi_k(\mathbf{x}_i)\right)^2 \geq 0
\end{aligned}$$

Quadratic kernel (1)

$K(\mathbf{x}, \mathbf{z}) = (\mathbf{xz})^2$

This corresponds to a feature space which contains only terms of degree 2 (products of two features)

(for $\mathbf{x} = (x_1, x_2)$ in R^2, these are x_1x_1, x_1x_2, x_2x_2)

For $\mathbf{x} = (x_1, x_2)$, $\mathbf{z} = (z_1, z_2)$:

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{z}) &= (\mathbf{xz})^2 \\
&= x_1^2 z_1^2 + 2x_1z_1x_2z_2 + x_2^2 z_2^2 \\
&= \varphi(\mathbf{x}) \cdot \varphi(\mathbf{z})
\end{aligned}
$$

Hence, $\varphi(\mathbf{x}) = (x_1^2, \sqrt{2} \cdot x_1x_2, x_2^2)$

Quadratic kernel (2)

$K(\mathbf{x}, \mathbf{z}) = (\mathbf{xz} + c)^2$

This corresponds to a feature space which contains constants, linear terms (original features), as well as terms of degree 2 (products of two features)

(for $\mathbf{x} = (x_1, x_2)$ in R^2: $x_1, x_2, x_1x_1, x_1x_2, x_2x_2$)

Polynomial kernels

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

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

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

Constructing new kernels from one existing kernel $k(\mathbf{x}, \mathbf{x}')$

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

- Multiplying $k(\mathbf{x}, \mathbf{x}')$ by a constant c:
  $k'(\mathbf{x}, \mathbf{x}') = ck(\mathbf{x}, \mathbf{x}')$
- Multiplying $k(\mathbf{x}, \mathbf{x}')$ by a function f applied to $\mathbf{x}$ and $\mathbf{x}'$:
  $k'(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$
- Applying a polynomial (with non-negative coefficients) to $k(\mathbf{x}, \mathbf{x}')$:
  $k'(\mathbf{x}, \mathbf{x}') = P(\, k(\mathbf{x}, \mathbf{x}')\,)$ with $P(z) = \sum_i a_i z^i$ and $a_i \geq 0$
- Exponentiating $k(\mathbf{x}, \mathbf{x}')$:
  $k'(\mathbf{x}, \mathbf{x}') = \exp(k(\mathbf{x}, \mathbf{x}'))$

Constructing new kernels by combining two kernels $k_1(\mathbf{x}, \mathbf{x}')$, $k_2(\mathbf{x}, \mathbf{x}')$

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

– Multiplying $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$:
  $k'(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$

Constructing new kernels

- If $\varphi(\mathbf{x}) \in R^m$ and $k_m(\mathbf{z}, \mathbf{z'})$ a valid kernel in R^m, $k(\mathbf{x}, \mathbf{x}') = k_m(\varphi(\mathbf{x}), \varphi(\mathbf{x}'))$ 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

Normalizing a kernel

Recall: you can normalize any vector $\mathbf{x}$ (transform it into a unit vector that has the same direction as $\mathbf{x}$) by

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|} = \frac{\mathbf{x}}{\sqrt{\mathbf{x}_1^2 + \ldots + \mathbf{x}_N^2}}$$

$$\begin{aligned} k'(\mathbf{x},\mathbf{z}) &= \frac{k(\mathbf{x},\mathbf{z})}{\sqrt{k(\mathbf{x},\mathbf{x})k(\mathbf{z},\mathbf{z})}} \\ &= \frac{\phi(\mathbf{x})\phi(\mathbf{z})}{\sqrt{\phi(\mathbf{x})\phi(\mathbf{x})\phi(\mathbf{z})\phi(\mathbf{z})}} \\ &= \frac{\phi(\mathbf{x})\phi(\mathbf{z})}{\|\phi(\mathbf{x})\|\|\phi(\mathbf{z})\|} \\ &= \psi(\mathbf{x})\psi(\mathbf{z}) \text{ with } \psi(\mathbf{x}) = \frac{\phi(\mathbf{x})}{\|\phi(\mathbf{x})\|} \end{aligned}$$

Gaussian kernel (aka radial basis function kernel)

$k(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/c)$

$\|\mathbf{x} - \mathbf{z}\|^2$: squared Euclidean distance between **x** and **z**

c (often called σ^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)

- $k(\mathbf{x}, \mathbf{z}) \approx 1$ when **x**, **z** close
- $k(\mathbf{x}, \mathbf{z}) \approx 0$ when **x**, **z** dissimilar

Gaussian kernel (aka radial basis function kernel)

$k(\mathbf{x}, \mathbf{z}) = \exp(- \| \mathbf{x} - \mathbf{z} \|^2/c)$

This is a valid kernel because:

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= \exp(- \| \mathbf{x} - \mathbf{z} \|^2/2\sigma^2) \\ &= \exp(-(\mathbf{xx} + \mathbf{zz} - 2\mathbf{xz})/2\sigma^2) \\ &= \exp(-\mathbf{xx}/2\sigma^2) \exp(\mathbf{xz}/\sigma^2) \exp(-\mathbf{zz}/2\sigma^2) \\ &= f(\mathbf{x}) \exp(\mathbf{xz}/\sigma^2) f(\mathbf{z}) \end{aligned}$$

$\exp(\mathbf{xz}/\sigma^2)$ is a valid kernel:

- $\mathbf{xz}$ is the linear kernel;
- we can multiply kernels by constants ($1/\sigma^2$)
- we can exponentiate kernels

Kernels over (finite) sets

X, Z: subsets of a finite set D with $|D|$ elements

$k(X, Z) = |X \cap Z|$ (the number of elements in X and Z)
is a valid kernel:

$k(X, Z) = \varphi(X)\varphi(Z)$ where $\varphi(X)$ maps X to a bit vector of length $|D|$
(*i*-th bit: does X contains the *i*-th element of D?).

$k(X, Z) = 2^{|X \cap Z|}$ (the number of subsets shared by X and Z)
is a valid kernel:

$\varphi(X)$ maps X to a bit vector of length $2^{|D|}$
(*i*-th bit: does X contains the *i*-th subset of D?)