Kernel Trick.

Note that the final classifier depends on \( x \) only by dot products.

E.g., \( x \cdot x_m \) in final classifier.

\( x_m \cdot x \) in the dual energy.

This motivates the Kernel Trick

Compute features \( \phi(x) \) and reformulate the problem in feature space.

\( \Rightarrow \) i.e. seek a classifier of form

\[
\text{sign} \left( \leq \cdot \phi(x) + b \right).
\]

Replace \( x \) by \( \phi(x) \) everywhere in the primal & dual formulation.

Then the classifier only depends on the dot product of the \( \phi(x) \)'s.

on the kernel \( \kappa(x, x') = \phi(x), \phi(x') \).
Why does this help?

First, features can make it possible to classify data by hyperplanes.

Example: logical X-OR, $\mathbf{X} = (x_1, x_2)$, $x_i \in \{\pm 1\}$

The X-OR (exclusive or) requires a decision rule $\mathbf{X}(x) \iff$

$$\mathbf{X}(+1) = \mathbf{X}(-1, -1) = 1$$
$$\mathbf{X}(+1) = \mathbf{X}(-1, +1) = -1$$

Impossible to find a linear classifier to do this.

But define a feature $\varphi(x_1, x_2) = x_1 x_2$.

Then we can classify the data by a linear classifier (i.e. a point) in 1-dimension.

$$x_1 x_2 = 1$$ for $\text{+ve's}$

$$x_1 x_2 = -1$$ for $\text{-ve's}$

classifier: if $\varphi(x_1, x_2) > 0$, +ve
$\varphi(x_1, x_2) < 0$, -ve
This example is “cheating” because we guess the right feature $x_1 x_2$.

More generally, we could set

$$\mathcal{P}(x_1, x_2) = (x_1, x_2, x_1^2, x_2^2, x_1 x_2, \ldots)$$

i.e. we specify a higher-dimensional set of features (higher than $\dim (x_1, x_2) = 2$).

$$\mathbf{z} = (\lambda_1, \lambda_2, \ldots)$$

Then $\mathbf{z} \cdot \mathcal{P}(\mathbf{x}) = \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_1 x_2 + \ldots$

The sum algorithm selects $\mathbf{z}$ to separate the positive and negative examples — maybe by setting $\lambda_2 = 1$, $\lambda_1 = \lambda_3 = \ldots = 0$

which gives us our “cheating” solution.

But there may be other equally good, or better, solutions $\hat{\mathbf{z}}$.

Note: we need more examples in this case, because we need many more examples than free parameters (e.g. dimension of $\mathcal{P}$).

So

```
Bottom-Line: By using a high-dimensional function $\mathcal{P}(x)$ (i.e. so that $\dim \mathcal{P} \gg \dim \mathbf{x}$), it is more likely that we can find a separating plane.
```
Second, we do not need to specify the features $\varphi(x)$ explicitly, we only need to specify the kernel

$$K(x, x') = \varphi(x) \cdot \varphi(x').$$

Remember:

the dual problem reduces to

$$\text{maximizing } L_d (\lambda, \alpha) = \sum \lambda \mu - \frac{1}{2} \sum \lambda \mu \alpha_{\nu} y_{\nu} y_{\nu} \varphi(x) \cdot \varphi(x')$$

$$= \sum \lambda \mu - \frac{1}{2} \sum \lambda \mu \alpha_{\nu} y_{\nu} y_{\nu} K(x, x').$$

The solution $\hat{\alpha} = \sum \lambda \mu \alpha_{\nu} y_{\nu} \varphi(x)$

$$\hat{\alpha} \cdot \varphi(x) = \sum \lambda \mu \alpha_{\nu} y_{\nu} \varphi(x) \cdot \varphi(x')$$

$$= \sum \lambda \mu y_{\nu} K(x, x').$$

Hence classifier is

$$\text{sign} \left\{ \sum \hat{\alpha}_{\nu} y_{\nu} K(x, x_{\nu}) + b \right\}$$

only depends on $K(\cdot, \cdot)$. 
What Kernels to Use?

There are many choices of Kernels. The difficulty is knowing which one to use. As always, cross-validation is useful for checking whether a kernel can generalize.

\[
\begin{align*}
K(x, x') &= \langle 1 + x \cdot x' \rangle^d \\
K(x, x') &= e^{-\frac{1}{2\sigma^2} ||x - x'||^2} \\
K(x, x') &= \tanh \langle c_1 x \cdot x' + c \rangle
\end{align*}
\]

Choice of best kernel is problem dependent.

Some kernels \( \rightarrow \) eg. \( \langle 1 + x \cdot x' \rangle^d \) naturally generalize the idea of hyperplanes.

Others \( \rightarrow \) eg. \( e^{-\frac{1}{2\sigma^2} ||x - x'||^2} \)

\( \Rightarrow \)

are similar to nearest neighbours.
(6) There are two important types of kernels.

(1) Polynomial Kernels

- Example: \( K(x_1, x_2) = x_1 x_2 \) (separating by plane, previous lecture)
- or \( K(x_1, x_2) = x_1 x_2 + (x_1^2 + 1)(x_2^2 + 1) \).

These give boundaries like these.

(2) Radial Basis Function

\[ K(x_1, x_2) = e^{-\|x_1 - x_2\|^2} \]

Note: \( K(x_1, x_2) \) decreases as \( \|x_1 - x_2\|^2 \) increases.

This kernel behaves like nearest neighbor! Only nearby data points affect classification.

Recall classifier is of form:

\[ \hat{y}(x) = \text{sign} \left\{ \sum \hat{\alpha}_n y_n K(x, x_n) + \hat{b} \right\} \]

(The \( \hat{\alpha}_n \)'s are non-zero only for support vectors)

Because \( K(x, x_n) \) decreases rapidly as \( \|x - x_n\| \) increases, it follows that the estimate \( \hat{y}(x) \) depends only on the data points near \( x \). It depends on the weighted sum of the \( y \)'s (recall \( \hat{\alpha}_n > 0 \)).

Note: Can also specify a set of kernels \( K_1(\cdot), \ldots, K_m(\cdot) \) parameters \( (a_1, \ldots, a_m) \), where \( \text{Kend} = \sum_{i=1}^{m} a_i K_i(\cdot, \cdot) \)

Use cross-validation to select the best \( a \)'s.
Geometric Interpretation

The kernel can be used to specify a distance measure. Recall, Euclidean distance:

$$\| \mathbf{x}_1 - \mathbf{x}_2 \|^2 = \mathbf{x}_1 \cdot \mathbf{x}_1 - 2 \mathbf{x}_1 \cdot \mathbf{x}_2 + \mathbf{x}_2 \cdot \mathbf{x}_2$$

Similarly,

$$1 \left[ \Phi (\mathbf{x}_1) - \Phi (\mathbf{x}_2) \right]^2 = \Phi (\mathbf{x}_1) \cdot \Phi (\mathbf{x}_1) + \Phi (\mathbf{x}_2) \cdot \Phi (\mathbf{x}_2) - 2 \Phi (\mathbf{x}_1) \cdot \Phi (\mathbf{x}_2)$$

$$= K (\mathbf{x}_1, \mathbf{x}_1) + K (\mathbf{x}_2, \mathbf{x}_2) - 2 K (\mathbf{x}_1, \mathbf{x}_2)$$

**Euclidean distance in \( \Phi \)-space**

**Special case**, if \( K (\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1 \cdot \mathbf{x}_2 \), then we recover \( \| \mathbf{x}_1 - \mathbf{x}_2 \| \).

**Bottom line**: think of \( \mathbf{x} \mapsto \Phi (\mathbf{x}) \) as a transformation on the geometry, which is the same as specifying a new distance \(- K (\mathbf{x}_1, \mathbf{x}_1) + K (\mathbf{x}_2, \mathbf{x}_2) - 2 K (\mathbf{x}_1, \mathbf{x}_2)\) - in the original \( \mathbf{x} \)-space.

**Note**: the RBF kernel has interesting geometry.

$$K (\mathbf{x}_1, \mathbf{x}_2) + K (\mathbf{x}_2, \mathbf{x}_2) - 2 K (\mathbf{x}_1, \mathbf{x}_2) = 2 \left( 1 - e^{-\| \mathbf{x}_1 - \mathbf{x}_2 \|^2} \right)$$

So as \( \| \mathbf{x}_1 - \mathbf{x}_2 \| \) increases, the distance between data points in \( \Phi \)-space tends to a maximum value of 2.
When do kernels correspond to features?

I.e., if we specify \( K(x, x') \), is it equal to \( \Phi(x) \cdot \Phi(x') \) for some feature \( \Phi(x) \)?

There are theoretical results \( \rightarrow \) Mercer's Theorem.

Here we do two things:

(i) give some partial understanding, by showing that this relates to the Spectral Theorem in Linear Algebra.

(ii) explaining why this result matters in practice, giving alternatives to SVM.

Consider \( K(x, y) \)

allow \( x, y \) to take a finite set of possible values \( \{ x_1, x_2, \ldots, x_n \} \) evenly spaced, \( \frac{x_{i+1} - x_i}{x_{i-1} - x_i} \)

Then consider \( K(x_i, x_j) = K_{ij} \)

This is an \( n \times n \) matrix \( K \) (with positive terms \( K_{ij} > 0 \))

By the Spectral Theorem (linear analysis)

we can express it as \( K_{ij} = \sum_{m=1}^{\infty} \lambda_m e_m^i e_m^j \)

where \( \lambda, e \) are the eigenvalues and eigenvectors of \( K \),

i.e. \( \sum_{i} K_{ij} e_m^i = \lambda_m e_m^j \)
The eigenvalues \( \lambda^m > 0 \) (because \( K_{ij} > 0 \) for all \( i,j \)) so

\[ K_{ij} = \sum_{m=1}^{\infty} (\lambda_i^m e_i^m)(\lambda_j^m e_j^m) \]

\[ K(\Xi, \Xi_0) = \varphi(\Xi) \cdot \varphi(\Xi_0) \]

where \( \varphi(\Xi) = (\gamma_1^i e_i, \gamma_2^i e_i, \ldots, \gamma_n e_i) \)

This answers the question - when can we find the \( \varphi \)?

If we restrict the \( x, y \) to only take values \( \Xi, \ldots, \Xi_n \)

Instead, we have to consider \( K(x, y) \) for continuous values of \( x \) and \( y \).

This requires functional analysis, finding eigenfunctions and eigenvalues.

\[ \int K(x, y) \varphi(y) \, dy = \lambda \varphi(x) \]

Functional analysis extends linear algebra to functions.

Out of scope of this course. But Mercer's theorem is like a generalization of the spectral theorem to functional analysis.

Why does this matter? New Algorithms

Consider the spectral theorem again. Order the eigenvalues so that \( \lambda_1 > \lambda_2 > \ldots > \lambda_n \)

Then - depending on \( K(\Xi, \Xi_0) \) - we may get a very good approximation to \( K \) by using only a few terms in the spectral theorem. Those with the biggest \( \lambda \)'s.

\[ \text{i.e. replace } K(\Xi, \Xi_0) \text{ by } \sum_{m=1}^{n} \lambda^m \varphi_i e_i^m \]

This approximation can be very useful for some kernel methods. It means we need only evaluate \[ \sum_{m=1}^{n} \lambda^m \varphi_i e_i^m \] (full method requires functional analysis)