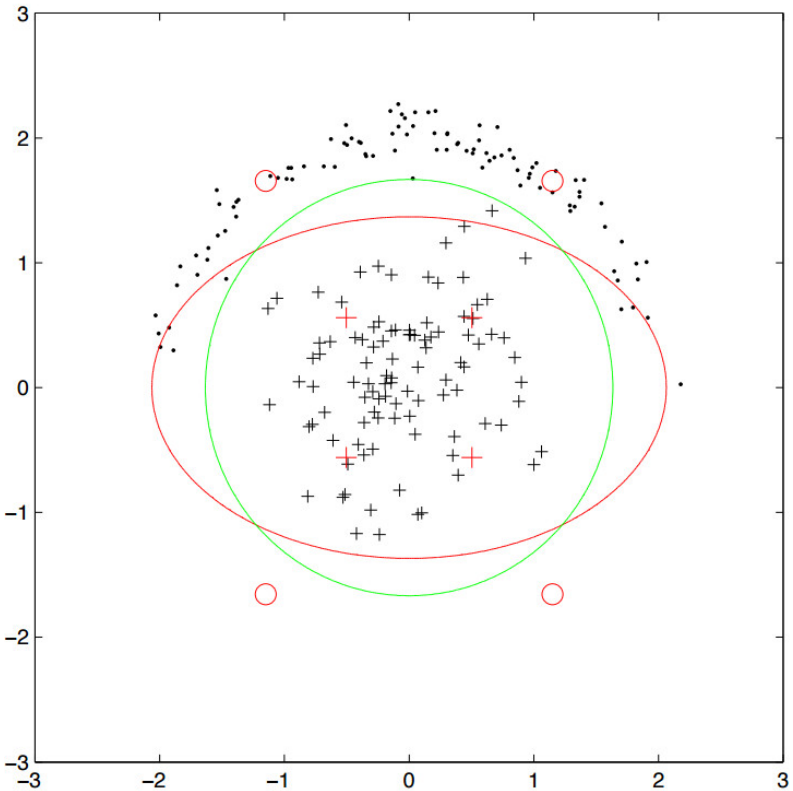
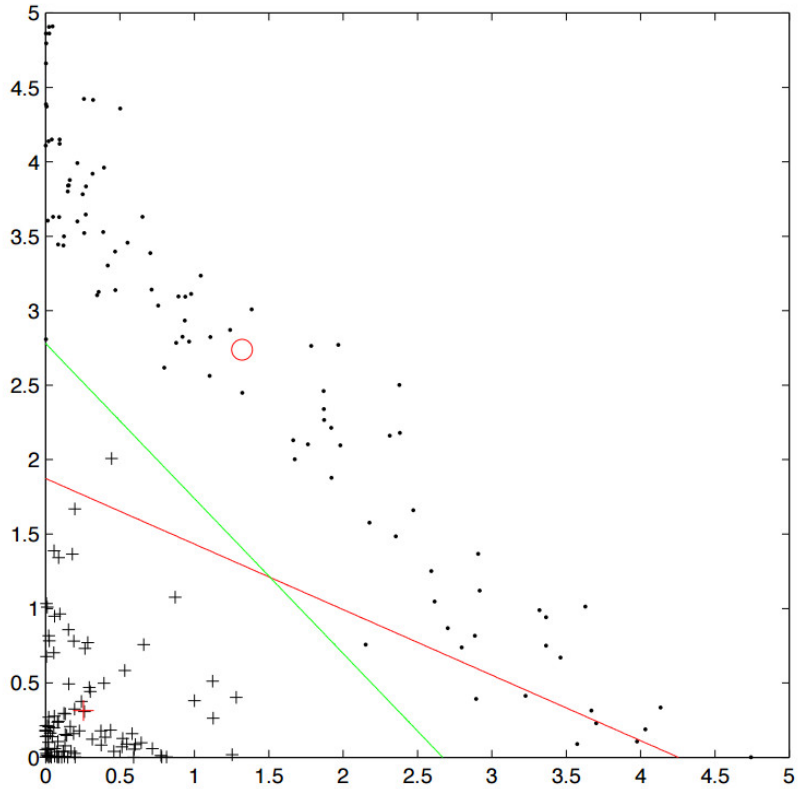
Kernel Methods

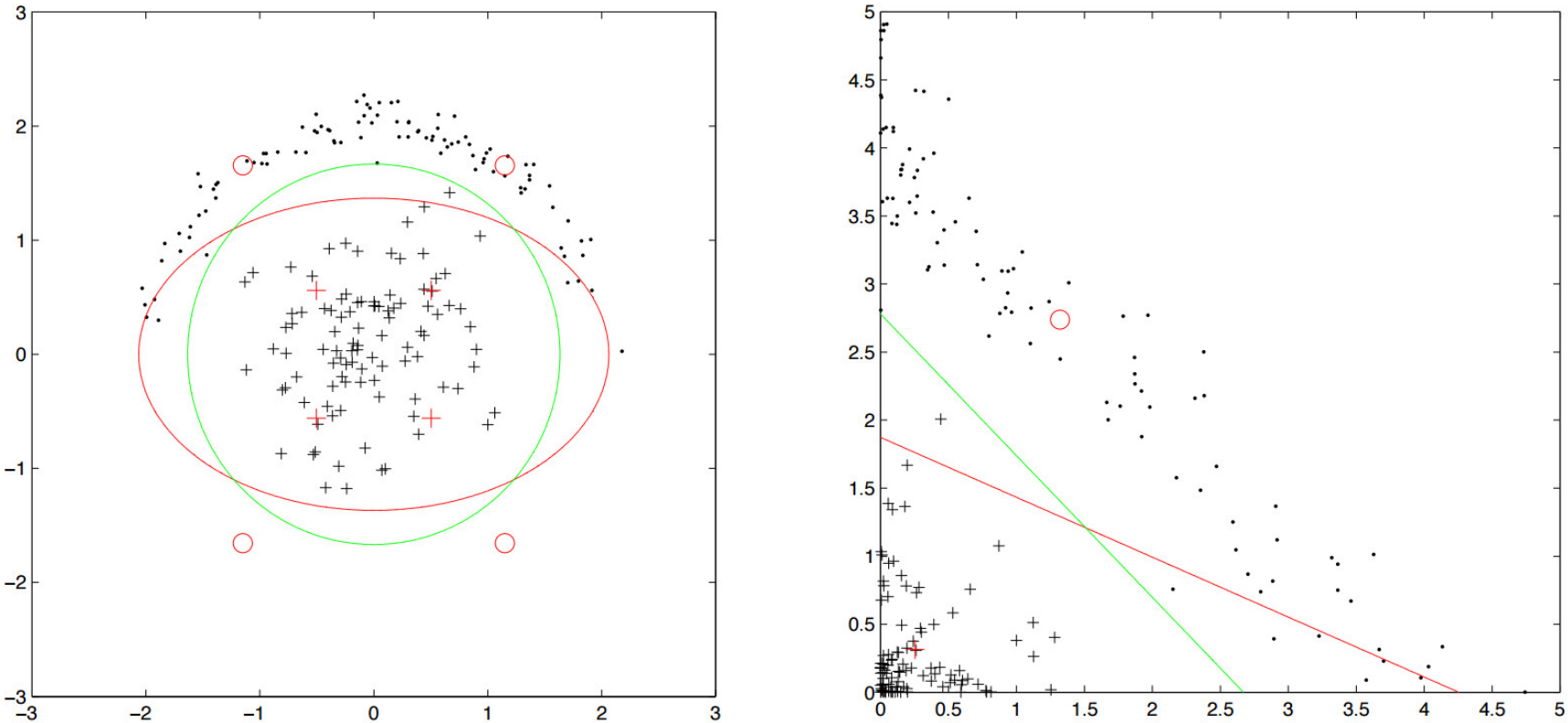
Quadratic Decision Boundary





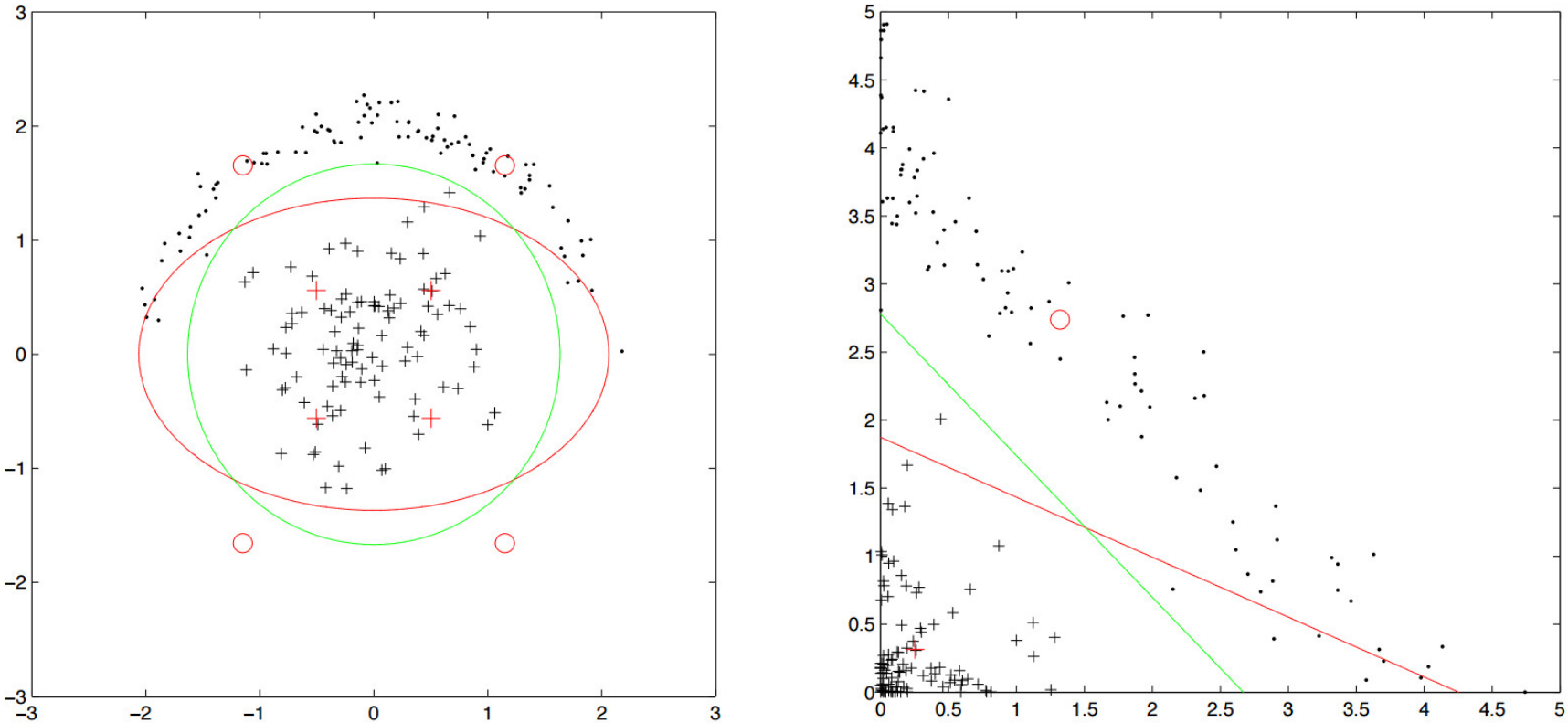
Left: The original set,

Quadratic Decision Boundary



Left: The original set, Right: Transformed using the square of features.

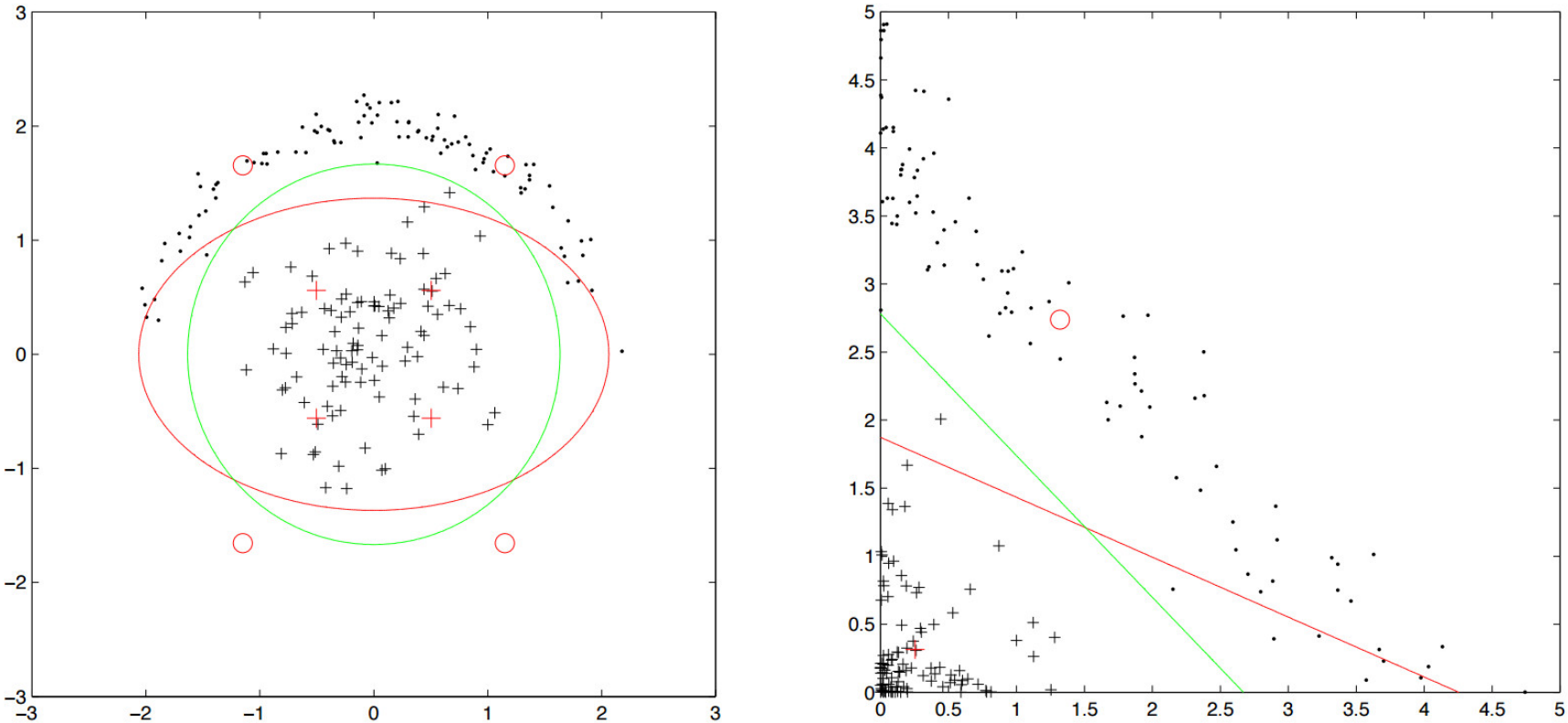
Quadratic Decision Boundary



Left: The original set, Right: Transformed using the square of features.

Right: the green line is a separating hyperplane in the transformed space.

Quadratic Decision Boundary

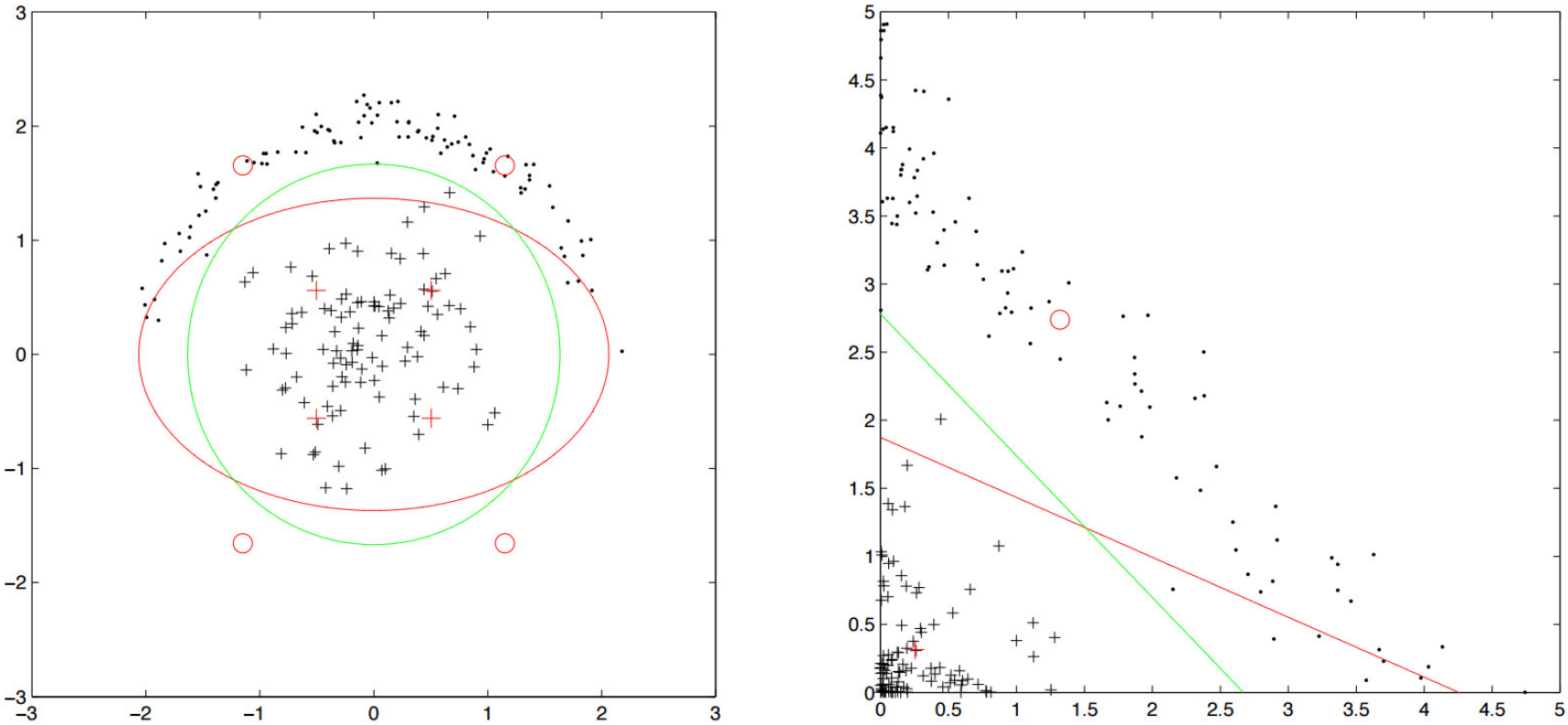


Left: The original set, Right: Transformed using the square of features.

Right: the green line is a separating hyperplane in the transformed space.

Left: the green ellipse maps exactly to the green line.

Quadratic Decision Boundary



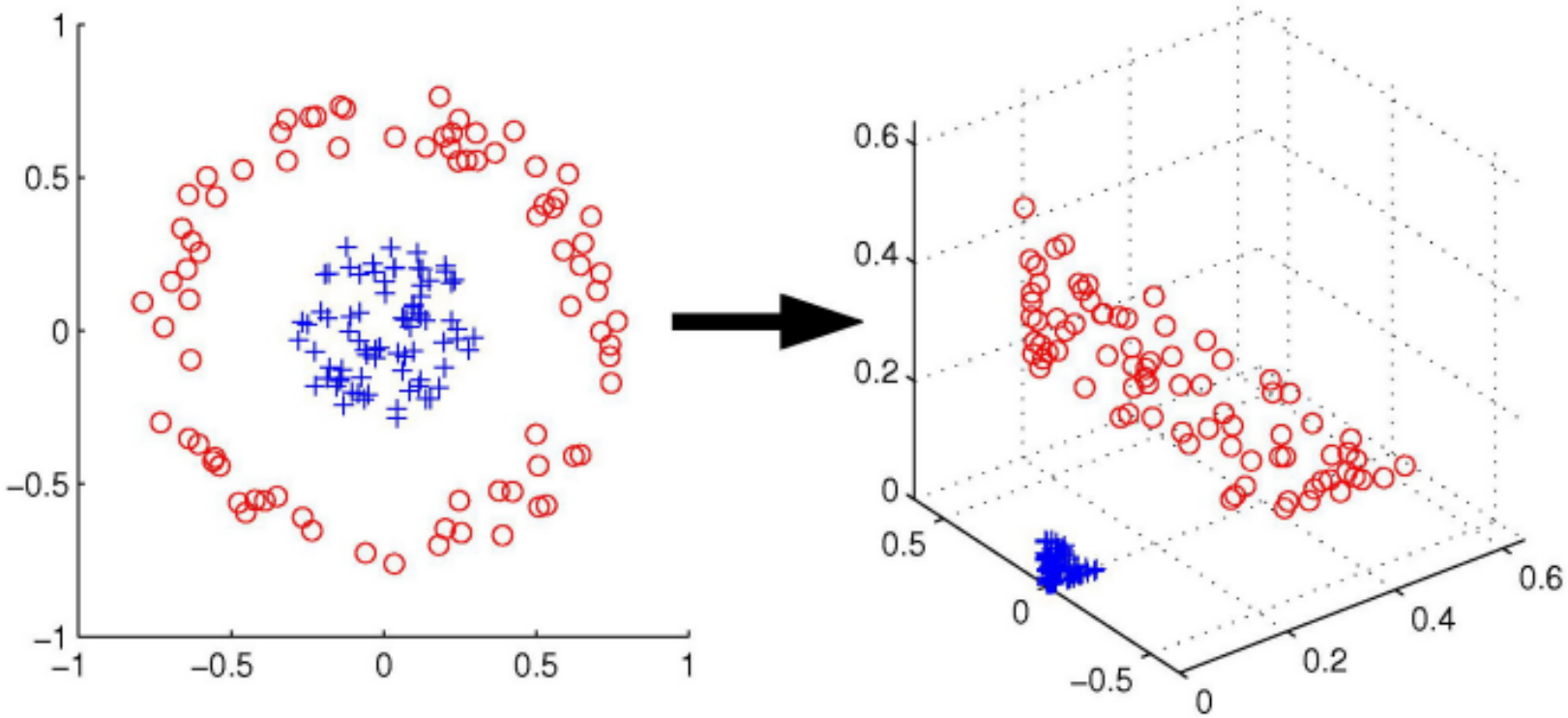
Left: The original set, Right: Transformed using the square of features.

Right: the green line is a separating hyperplane in the transformed space.

Left: the green ellipse maps exactly to the green line.

How to classify (in the original space): First, transform a given feature vector by squaring the features, then use a linear classifier.

Anothe Solution



Mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ so that there is "more space" for linear separation.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Sometimes its even beneficial to map to infinite-dimensional spaces.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Sometimes its even beneficial to map to infinite-dimensional spaces.

To avoid explicit construction of the higher dimensional feature space, we use so called *kernel trick*.

Do We Need to Map Explicitly?

In general, mapping to (much) higher feature space helps (there are more "degrees of freedom" so linear separability might get a chance).

However, complexity of learning grows (quickly) with dimension.

Sometimes its even beneficial to map to infinite-dimensional spaces.

To avoid explicit construction of the higher dimensional feature space, we use so called *kernel trick*.

But first we need to *dualize* our learning algorithm.

Linear Regression

- Given a set D of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k \in \mathbb{R}$.

- **Our goal:** Find $\vec{w}$ so that $h[\vec{w}](\vec{x_k}) = \vec{w} \cdot \tilde{\mathrm{x}}_k$ is close to f_k for every $k = 1, \ldots, p$.

Recall that $\tilde{\mathrm{x}}_k = (x_{k0}, x_{k1} \ldots, x_{kn})$ where $x_{k0} = 1$.

- **Squared Error Function:**

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}(\vec{w} \cdot \tilde{\mathrm{x}}_k - f_k)^2 = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=0}^{n} w_i x_{ki} - f_k\right)^2$$

Regularized Linear Regression

Regularized Squared Error Function:

$$E(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}(\vec{w}\cdot\tilde{\mathbf{x}}_k - f_k)^2 + \vec{w}\cdot\vec{w}$$

Intuition: the added term $\vec{w}\cdot\vec{w}$ prevents growth of weights.

Regularized Linear Regression

Regularized Squared Error Function:

$$E(\vec{w}) \;=\; \frac{1}{2}\sum_{k=1}^{p}(\vec{w}\cdot\tilde{x}_k - f_k)^2 + \vec{w}\cdot\vec{w}$$

Intuition: the added term $\vec{w}\cdot\vec{w}$ prevents growth of weights.

The Representer Theorem: The weight vector $\vec{w}^*$ minimizing the regularized squared error function can be written as

$$\vec{w}^* = \sum_{i=1}^{p}\alpha_i f_i \tilde{x}_i$$

Here $\alpha_1, \ldots, \alpha_p$ are suitable coefficients.

Regularized Linear Regression

Regularized Squared Error Function:

$$E(\vec{w}) \;=\; \frac{1}{2}\sum_{k=1}^{p}(\vec{w}\cdot\tilde{\mathrm{x}}_k - f_k)^2 + \vec{w}\cdot\vec{w}$$

Intuition: the added term $\vec{w}\cdot\vec{w}$ prevents growth of weights.

The Representer Theorem: The weight vector $\vec{w}^*$ minimizing the regularized squared error function can be written as

$$\vec{w}^* = \sum_{i=1}^{p}\alpha_i f_i\,\tilde{\mathrm{x}}_i \qquad \text{Here } \alpha_1,\ldots,\alpha_p \text{ are suitable coefficients.}$$

Substituting this expression for weights in E gives

$$E'(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\,\tilde{\mathrm{x}}_i\cdot\tilde{\mathrm{x}}_k - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\,\tilde{\mathrm{x}}_i\cdot\tilde{\mathrm{x}}_j$$

and we minimize E' w.r.t. $\alpha_1,\ldots,\alpha_p$. What is this good for??

Given a set D of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

Here $\vec{x}_k = (x_{k1} \ldots, x_{kn}) \in \mathbb{R}^n$ and $f_k \in \mathbb{R}$.

Find $\alpha_1, \ldots, \alpha_p$ *minimizing dual regularized squared error*

$$E'(\vec{w}) = \frac{1}{2} \sum_{k=1}^{p} \left(\sum_{i=1}^{p} \alpha_i f_i \, \tilde{\mathrm{x}}_i \cdot \tilde{\mathrm{x}}_k - f_k \right)^2 + \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j f_i f_j \, \tilde{\mathrm{x}}_i \cdot \tilde{\mathrm{x}}_j$$

The resulting coefficients $\alpha_1, \ldots, \alpha_p$ give a weight vector

$$\vec{w}^* = \sum_{i=1}^{p} \alpha_i f_i \, \tilde{\mathrm{x}}_i$$

which in turn gives a linear model

$$h[\vec{w}^*](\vec{x}) = \vec{w}^* \tilde{\mathrm{x}} = \sum_{i=1}^{p} \alpha_i f_i \, \tilde{\mathrm{x}}_i \cdot \tilde{\mathrm{x}}$$

Note that all $\tilde{\mathrm{x}}, \tilde{\mathrm{x}}_i, \tilde{\mathrm{x}}_j, \tilde{\mathrm{x}}_k$ occur in dot products with themselves!

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing dual regularized squared error*

$$E'(\vec{w}) = \frac{1}{2} \sum_{k=1}^{p} \left(\sum_{i=1}^{p} \alpha_i f_i \, \tilde{x}_i \cdot \tilde{x}_k - f_k \right)^2 + \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j f_i f_j \, \tilde{x}_i \cdot \tilde{x}_j$$

Linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p} \alpha_i f_i \, \tilde{x}_i \cdot \tilde{x}$

Do we need to use the dot product in the above procedure? NO!

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing dual regularized squared error*

$$E'(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\, \tilde{x}_i \cdot \tilde{x}_k - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\, \tilde{x}_i \cdot \tilde{x}_j$$

Linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p}\alpha_i f_i\, \tilde{x}_i \cdot \tilde{x}$

Do we need to use the dot product in the above procedure? NO!

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing kernel dual regularized squared error*

$$E'(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\, \kappa(\tilde{x}_i, \tilde{x}_k)) - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\, \kappa(\tilde{x}_i, \tilde{x}_j)$$

Non-linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p}\alpha_i f_i\, \kappa(\tilde{x}_i, \tilde{x})$

Here κ is a **kernel function**. But now what is the trick?

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing dual regularized squared error*

$$E'(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\, \tilde{\mathrm{x}}_i \cdot \tilde{\mathrm{x}}_k - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\, \tilde{\mathrm{x}}_i \cdot \tilde{\mathrm{x}}_j$$

Linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p}\alpha_i f_i\, \tilde{\mathrm{x}}_i \cdot \tilde{\mathrm{x}}$

Do we need to use the dot product in the above procedure? NO!

Find $\vec{\alpha} = (\alpha_1, \ldots, \alpha_p)$ *minimizing kernel dual regularized squared error*

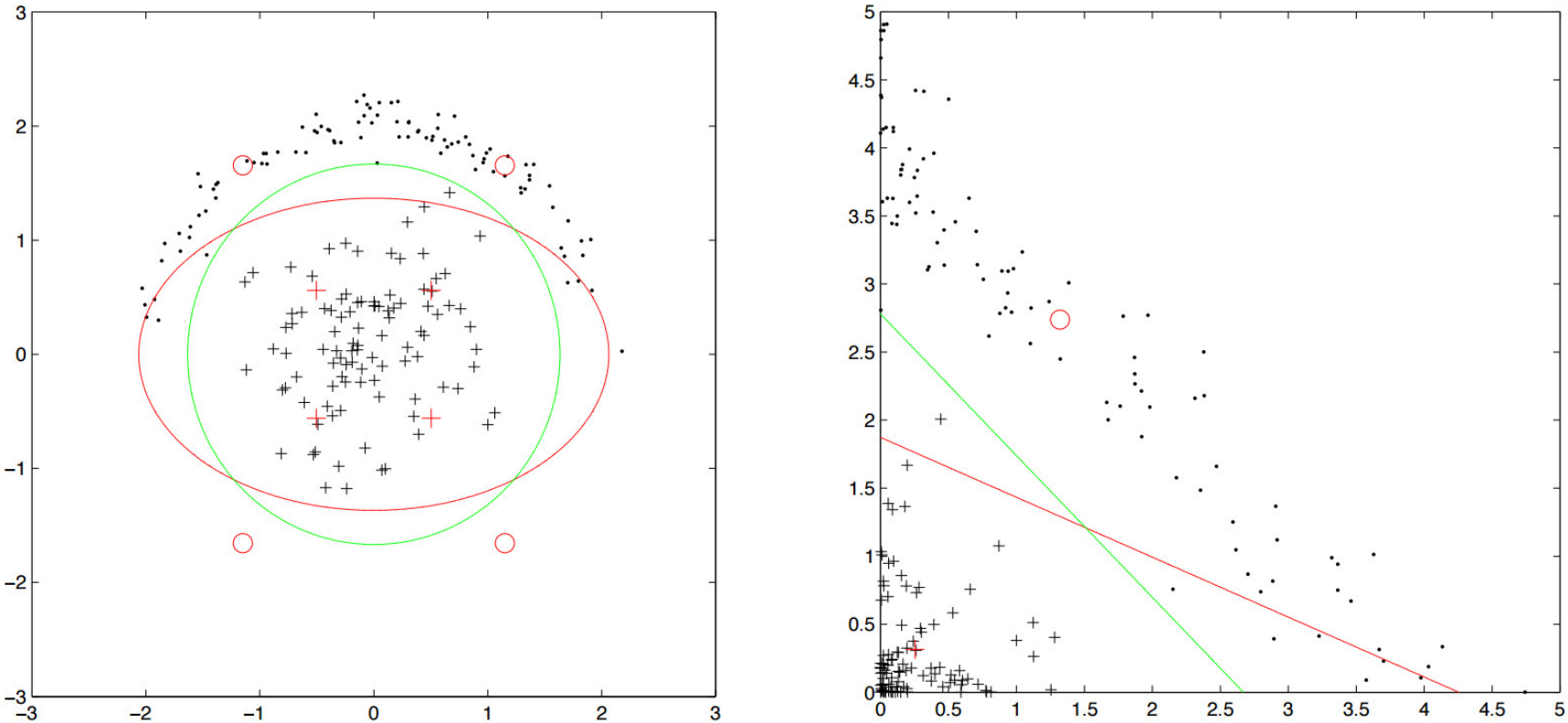
$$E'(\vec{w}) = \frac{1}{2}\sum_{k=1}^{p}\left(\sum_{i=1}^{p}\alpha_i f_i\, \kappa(\tilde{\mathrm{x}}_i, \tilde{\mathrm{x}}_k)) - f_k\right)^2 + \sum_{i=1}^{p}\sum_{j=1}^{p}\alpha_i\alpha_j f_i f_j\, \kappa(\tilde{\mathrm{x}}_i, \tilde{\mathrm{x}}_j)$$

Non-linear model: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p}\alpha_i f_i\, \kappa(\tilde{\mathrm{x}}_i, \tilde{\mathrm{x}})$

Here κ is a **kernel function**. But now what is the trick?

The trick is that suitable kernel functions κ correspond to dot products in transformed spaces!

Recall the Quadratic Decision Boundary



Left: The original set, Right: Transformed using the square of features.

Right: the green line is a separating hyperplane in the transformed space.

Left: the green ellipse maps exactly to the green line.

How to classify (in the original space): Transform a given feature vector by squaring the features, then use a linear classifier.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathbf{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathrm{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$.

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathrm{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathrm{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2 = (1 + x_{k1} x_{\ell 1} + x_{k2} x_{\ell 2})^2 =$$

$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2 x_{k1} x_{\ell 1} x_{k2} x_{\ell 2} + 2 x_{k1} x_{\ell 1} + 2 x_{k2} x_{\ell 2}$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathrm{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$

$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping ϕ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathrm{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathrm{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$

$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping ϕ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathrm{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Then $\phi(\tilde{\mathrm{x}}_k) \cdot \phi(\tilde{\mathrm{x}}_\ell) = (\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2$

Kernel Trick

For simplicity, assume bivariate data: $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$.

The *corresponding instance* in the quadratic feature space is $(1, x_{k1}^2, x_{k2}^2)$.

Consider two instances $\tilde{\mathrm{x}}_k = (1, x_{k1}, x_{k2})$ and $\tilde{\mathrm{x}}_\ell = (1, x_{\ell 1}, x_{\ell 2})$. Then the scalar product of their corresponding instances $(1, x_{k1}^2, x_{k2}^2)$ and $(1, x_{\ell 1}^2, x_{\ell 2}^2)$, resp., in the quadratic feature space is

$$1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2$$

which resembles (but is not equal to)

$$(\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2 = (1 + x_{k1}x_{\ell 1} + x_{k2}x_{\ell 2})^2 =$$

$$= 1 + x_{k1}^2 x_{\ell 1}^2 + x_{k2}^2 x_{\ell 2}^2 + 2x_{k1}x_{\ell 1}x_{k2}x_{\ell 2} + 2x_{k1}x_{\ell 1} + 2x_{k2}x_{\ell 2}$$

But now consider a mapping ϕ to $\mathbb{R}^6$ defined by

$$\phi(\tilde{\mathrm{x}}_k) = (1, x_{k1}^2, x_{k2}^2, \sqrt{2}x_{k1}x_{k2}, \sqrt{2}x_{k1}, \sqrt{2}x_{k2})$$

Then $\phi(\tilde{\mathrm{x}}_k) \cdot \phi(\tilde{\mathrm{x}}_\ell) = (\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2$

THE Idea: Using the kernel $\kappa(\tilde{\mathrm{x}}_k, \tilde{\mathrm{x}}_\ell) = (\tilde{\mathrm{x}}_k \cdot \tilde{\mathrm{x}}_\ell)^2$ in the kernel dual regularized squared error corredponds to using the regularized squared error after the transformation ϕ.

Quadratic Decision Boundary

Given a set D of training examples:

$$D = \{(\vec{x}_1, f_1), (\vec{x}_2, f_2), \ldots, (\vec{x}_p, f_p)\}$$

Assume that $f_i \in \{1, -1\}$ indicates the class of $\vec{x}_i$.

Yes, I know that squared error regression should not be used for classification!

Considering $\kappa(\tilde{x}_k, \tilde{x}_\ell) = (\tilde{x}_k \cdot \tilde{x}_\ell)^2$ in our kernel dual regularized squared error we obtain

Find $\vec{\alpha} = \alpha_1, \ldots, \alpha_p$ *minimizing*

$$E'(\vec{w}) = \frac{1}{2} \sum_{k=1}^{p} \left(\sum_{i=1}^{p} \alpha_i f_i \, (\tilde{x}_i \cdot \tilde{x}_k)^2 - f_k \right)^2 + \sum_{i=1}^{p} \sum_{j=1}^{p} \alpha_i \alpha_j f_i f_j \, (\tilde{x}_i \cdot \tilde{x}_j)^2$$

Non-linear classifier: $h[\vec{\alpha}](\vec{x}) = \sum_{i=1}^{p} \alpha_i f_i \, (\tilde{x}_i \cdot \tilde{x})^2$

Intuitively, minimizing E' in $\mathbb{R}^2$ gives a separating hyperplane for the input vectors transformed into $\mathbb{R}^5$. This means, that in $\mathbb{R}^2$ it searches for a quadratic (i.e., non-linear) boundary.

Examples of Kernels

- Linear: $\kappa(\tilde{\mathsf{x}}_\ell, \tilde{\mathsf{x}}_\mathsf{k}) = \tilde{\mathsf{x}}_\ell \cdot \tilde{\mathsf{x}}_\mathsf{k}$

  The corresponding mapping $\phi(\tilde{\mathsf{x}}) = \tilde{\mathsf{x}}$ is identity (no transformation).

Examples of Kernels

- Linear: $\kappa(\tilde{x}_\ell, \tilde{x}_k) = \tilde{x}_\ell \cdot \tilde{x}_k$

  The corresponding mapping $\phi(\tilde{x}) = \tilde{x}$ is identity (no transformation).

- Polynomial of power m: $\kappa(\tilde{x}_\ell, \tilde{x}_k) = (\tilde{x}_\ell \cdot \tilde{x}_k)^m$

  The corresponding mapping assigns to $\tilde{x} \in \mathbb{R}^{n+1}$ the vector $\phi(\tilde{x})$ in $\mathbb{R}^{\binom{n+m}{m}+1}$.

Examples of Kernels

- Linear: $\kappa(\tilde{\mathsf{x}}_\ell, \tilde{\mathsf{x}}_\mathsf{k}) = \tilde{\mathsf{x}}_\ell \cdot \tilde{\mathsf{x}}_\mathsf{k}$

  The corresponding mapping $\phi(\tilde{\mathsf{x}}) = \tilde{\mathsf{x}}$ is identity (no transformation).

- Polynomial of power m: $\kappa(\tilde{\mathsf{x}}_\ell, \tilde{\mathsf{x}}_\mathsf{k}) = (\tilde{\mathsf{x}}_\ell \cdot \tilde{\mathsf{x}}_\mathsf{k})^m$

  The corresponding mapping assigns to $\tilde{\mathsf{x}} \in \mathbb{R}^{n+1}$ the vector $\phi(\tilde{\mathsf{x}})$ in $\mathbb{R}^{\binom{n+m}{m}+1}$.

- Gaussian (radial-basis function): $\kappa(\tilde{\mathsf{x}}_\ell, \tilde{\mathsf{x}}_\mathsf{k}) = e^{-\frac{\|\tilde{\mathsf{x}}_\ell - \tilde{\mathsf{x}}_\mathsf{k}\|^2}{2\sigma^2}}$

  The corresponding mapping ϕ maps $\tilde{\mathsf{x}}$ to an *infinite-dimensional* vector $\phi(\tilde{\mathsf{x}})$ which is, in fact, a Gaussian function; combination of such functions for support vectors is then the separating hypersurface.

- ...

Examples of Kernels

- Linear: $\kappa(\tilde{x}_\ell, \tilde{x}_k) = \tilde{x}_\ell \cdot \tilde{x}_k$

  The corresponding mapping $\phi(\tilde{x}) = \tilde{x}$ is identity (no transformation).

- Polynomial of power m: $\kappa(\tilde{x}_\ell, \tilde{x}_k) = (\tilde{x}_\ell \cdot \tilde{x}_k)^m$

  The corresponding mapping assigns to $\tilde{x} \in \mathbb{R}^{n+1}$ the vector $\phi(\tilde{x})$ in $\mathbb{R}^{\binom{n+m}{m}+1}$.

- Gaussian (radial-basis function): $\kappa(\tilde{x}_\ell, \tilde{x}_k) = e^{-\frac{\|\tilde{x}_\ell - \tilde{x}_k\|^2}{2\sigma^2}}$

  The corresponding mapping ϕ maps $\tilde{x}$ to an *infinite-dimensional* vector $\phi(\tilde{x})$ which is, in fact, a Gaussian function; combination of such functions for support vectors is then the separating hypersurface.

- ...

Choosing kernels remains to be black magic of kernel methods. They are usually chosen based on trial and error (of course, experience and additional insight into data helps).

Similar trick can be done with (soft-margin) support vector machines.