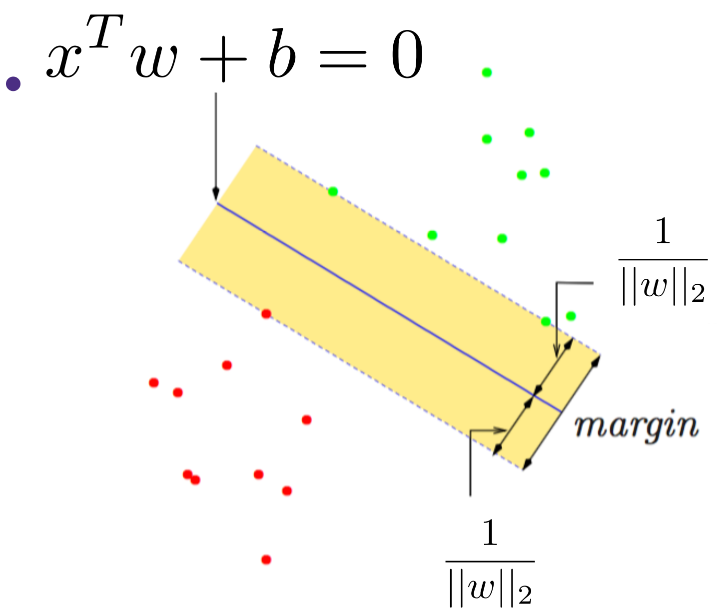
Kernels

What if the data is not linearly separable?



Some points do not satisfy margin constraint:

$$\min_{w,b} ||w||_2^2$$

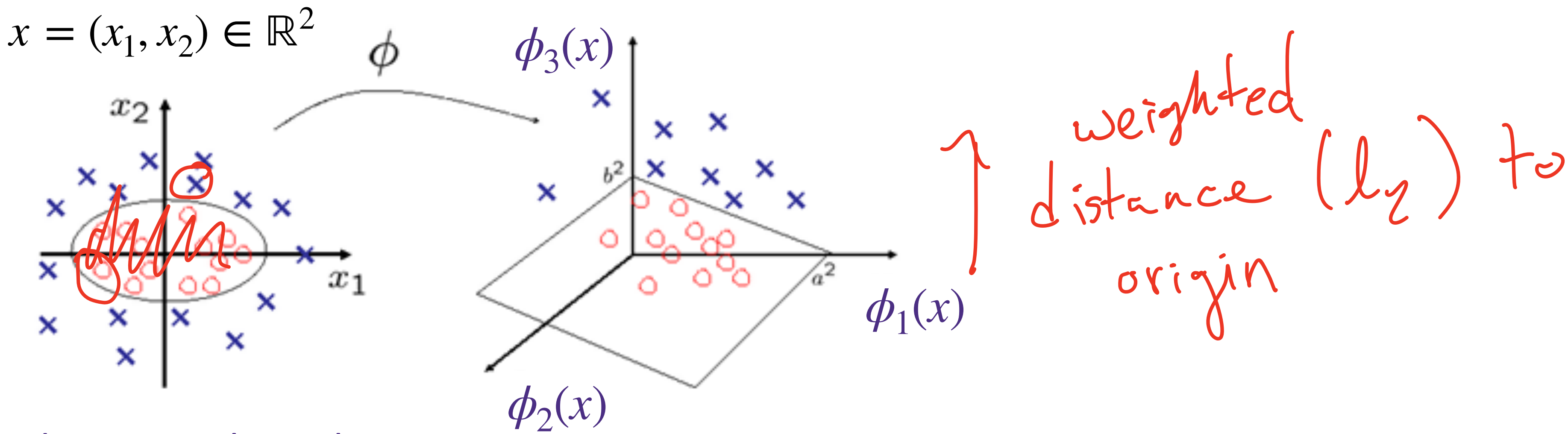
$$y_i(x_i^T w + b) \geq 1 \quad \forall i$$

Two options:

1. Introduce slack to this optimization problem (Support Vector Machine)
2. **Lift to higher dimensional space (Kernels)**

What if the data is not linearly separable?

- Use features, for example,

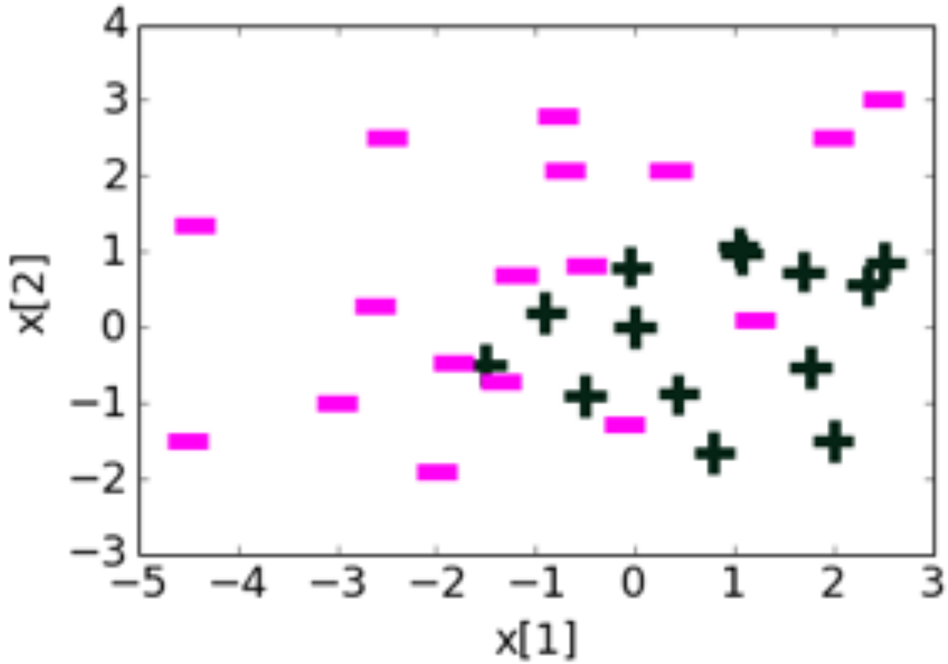


This data is not linearly separable

Can you suggest some features $\phi_1(x_1, x_2), \phi_2(x_1, x_2), \phi_3(x_1, x_2)$ such that this data is linearly separable in this 3-dimensional space?

- Generally, in high dimensional feature space, it is easier to linearly separate different classes
- However, it is hard to know which feature map will work for given data
- So the rule of thumb is to use high-dimensional features and hope that the algorithm will automatically pick the right set of features

Example: adding more polynomial features

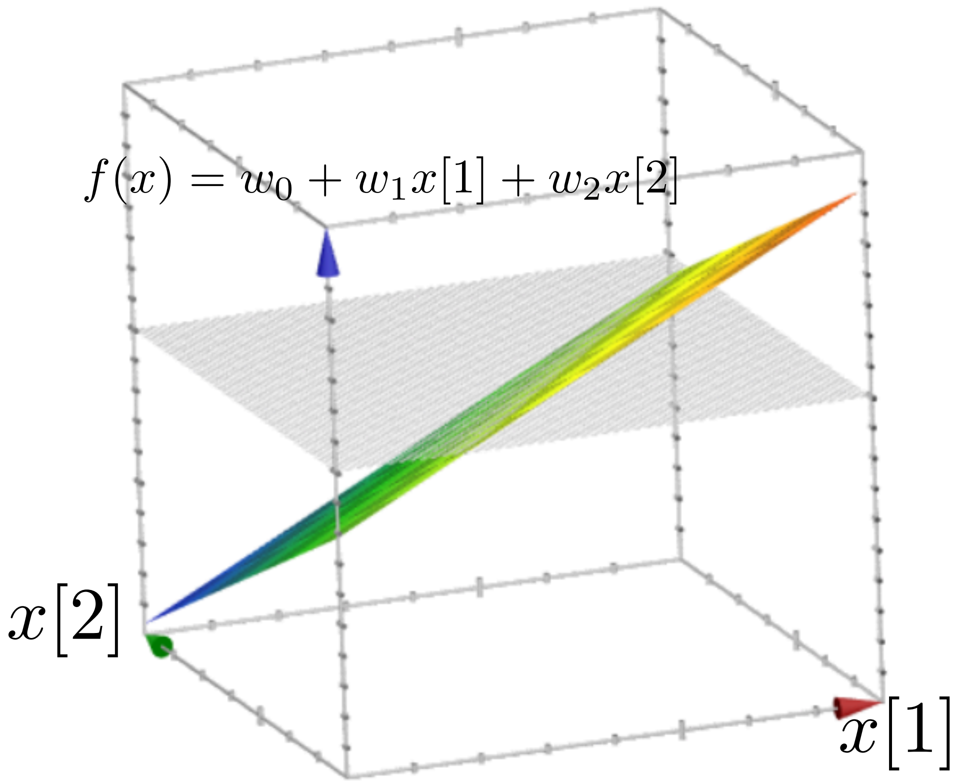


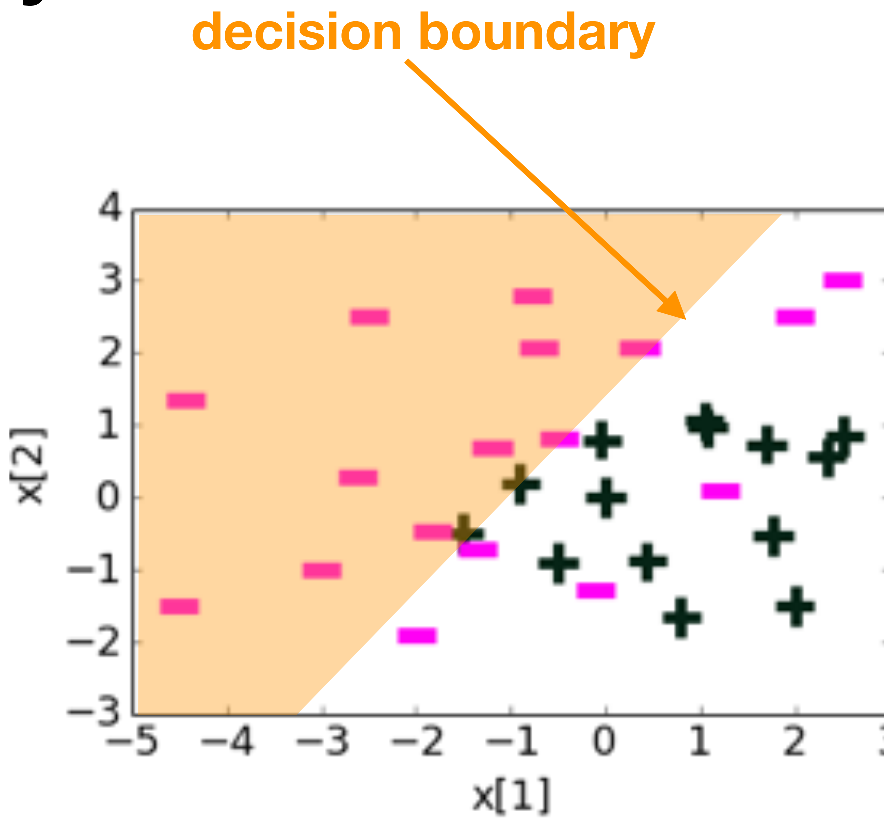
Polynomial features

$$\begin{bmatrix} h_0(x) = 1 \\ h_1(x) = x[1] \\ h_2(x) = x[2] \\ h_3(x) = x[1]^2 \\ h_4(x) = x[2]^2 \\ \vdots \end{bmatrix}$$

- data: ***x*** in 2-dimensions, ***y*** in {+1,-1}
- features: polynomials
- model: linear on polynomial features
- $$f(x) \ = \ w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

Learned decision boundary

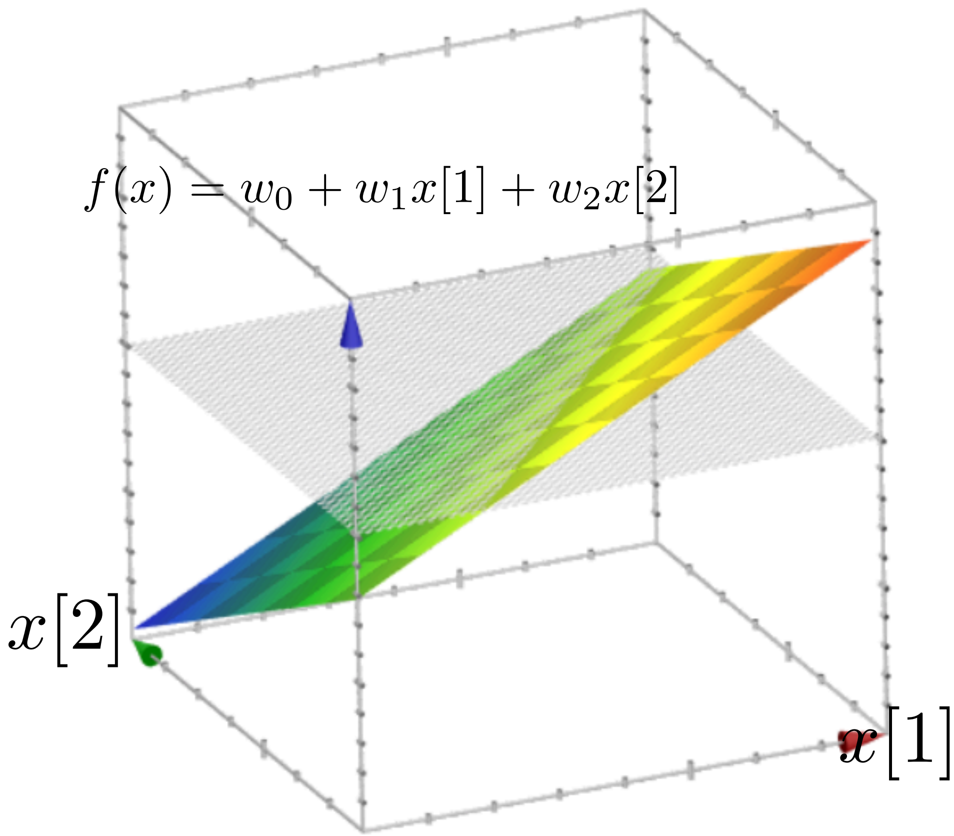


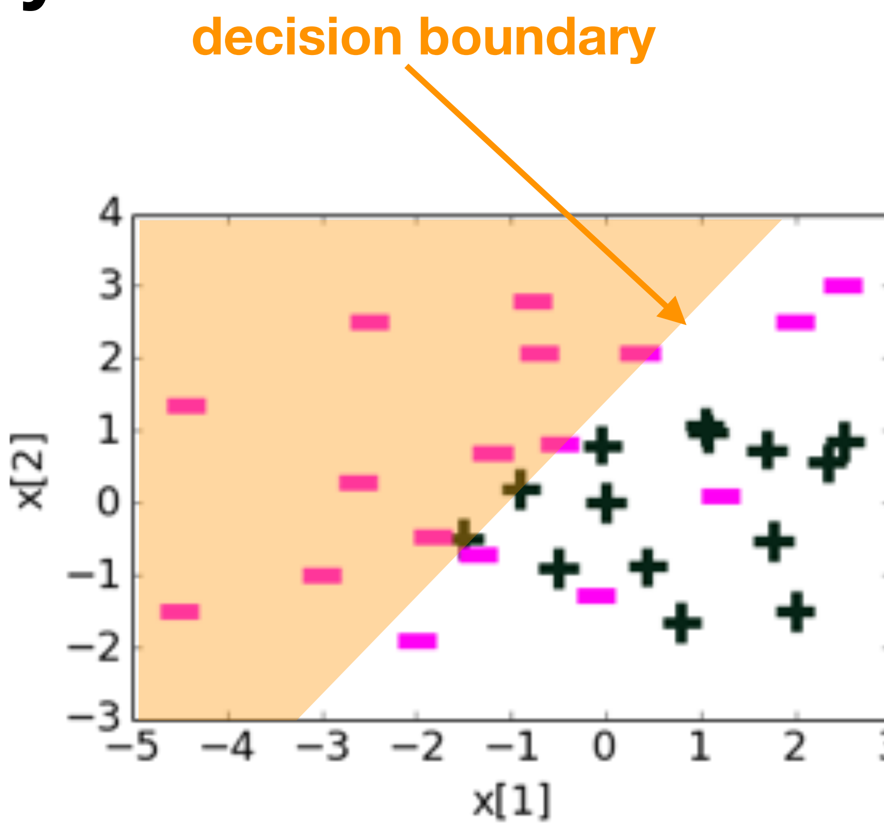


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Learned decision boundary

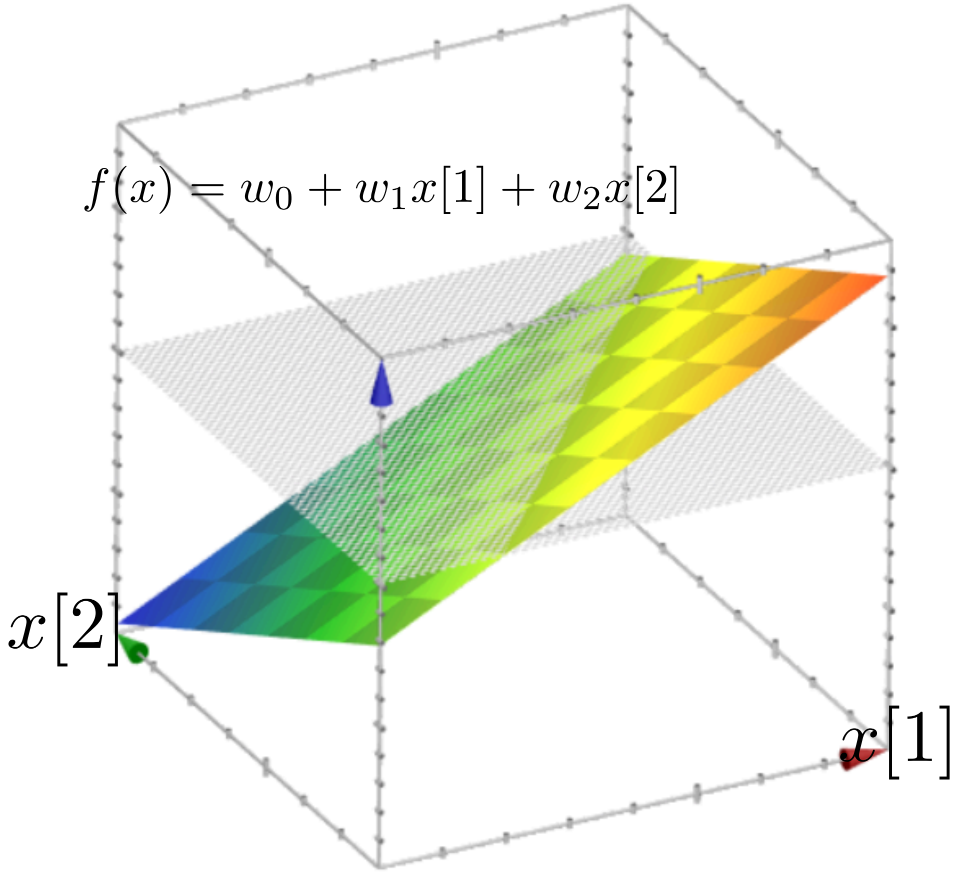


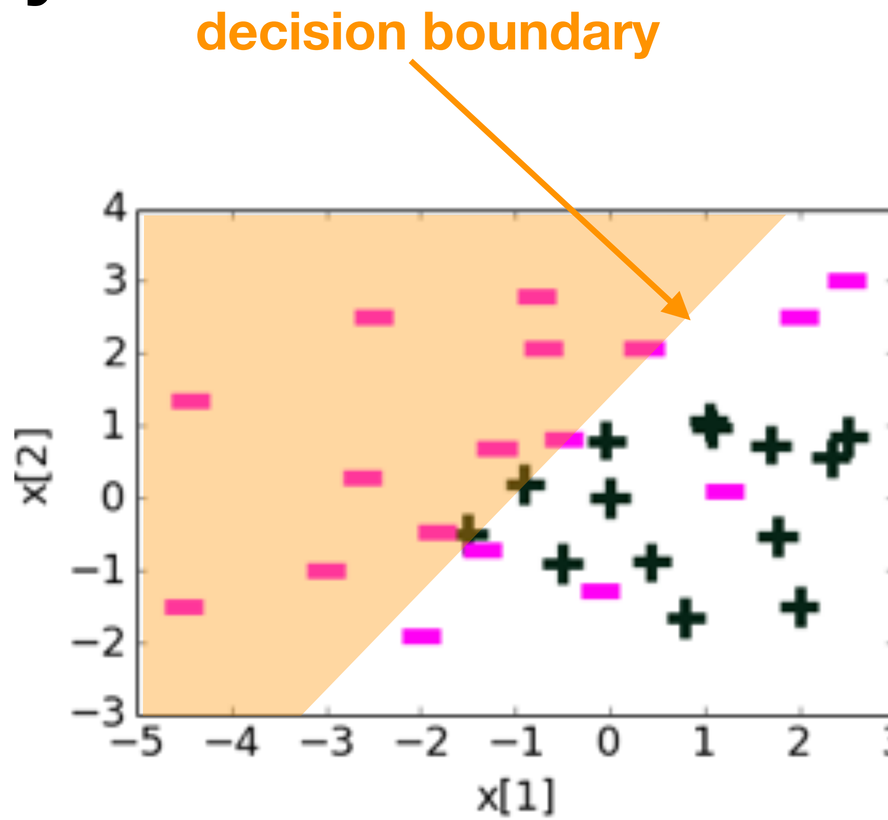


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Learned decision boundary

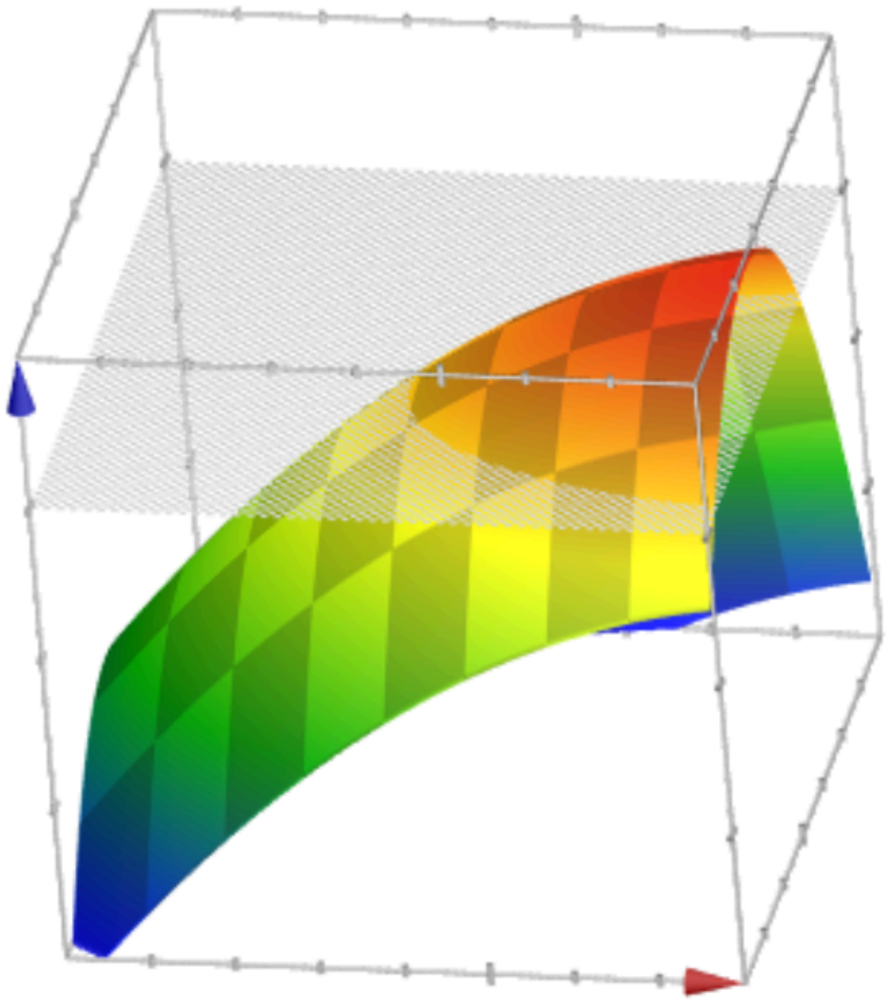


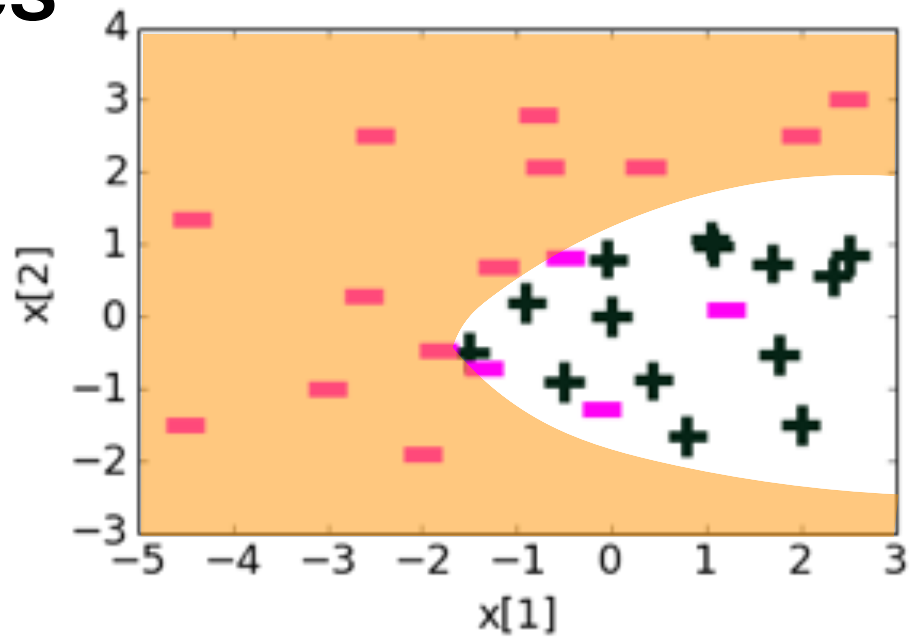


Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Adding quadratic features

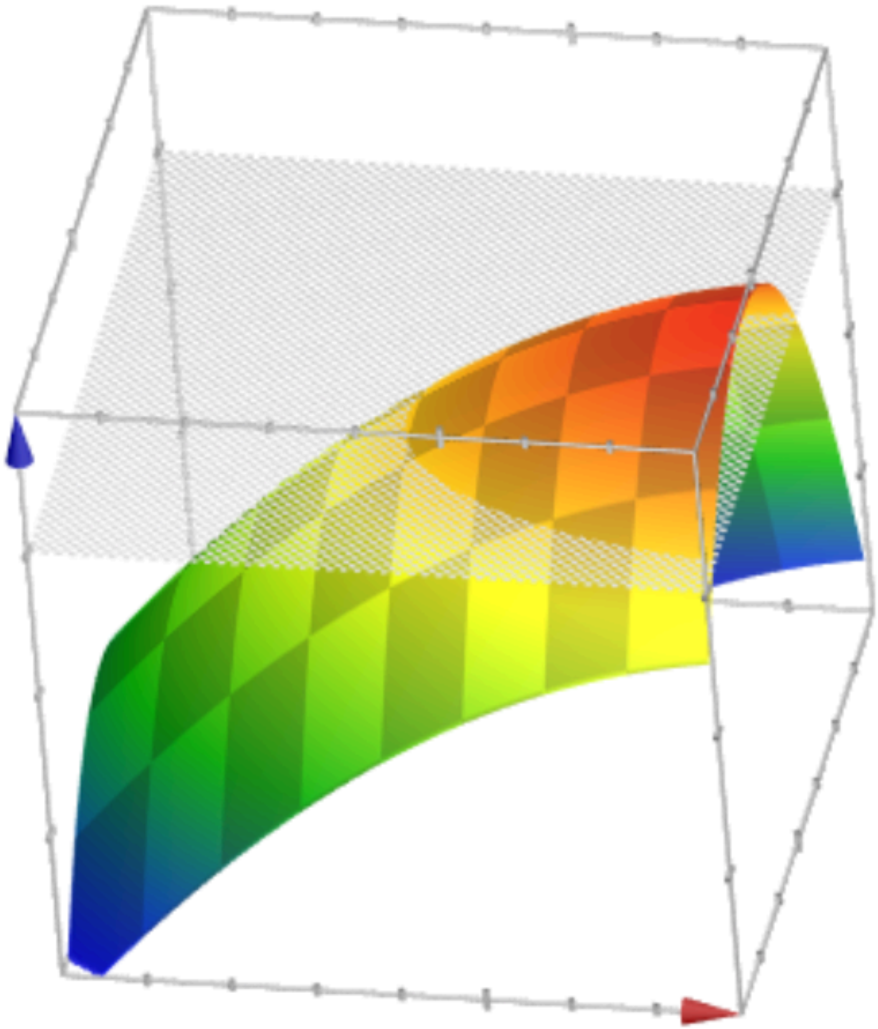


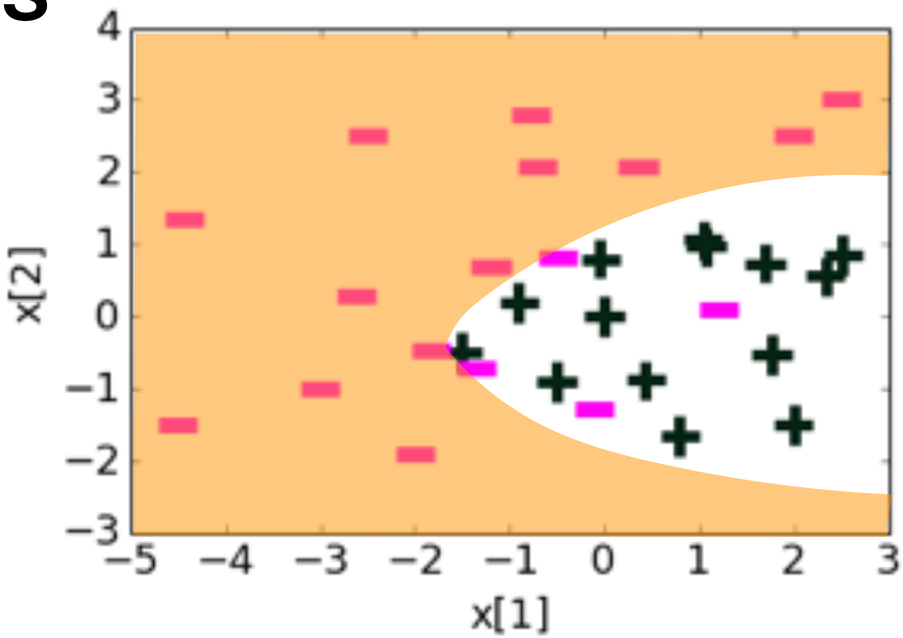


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding quadratic features

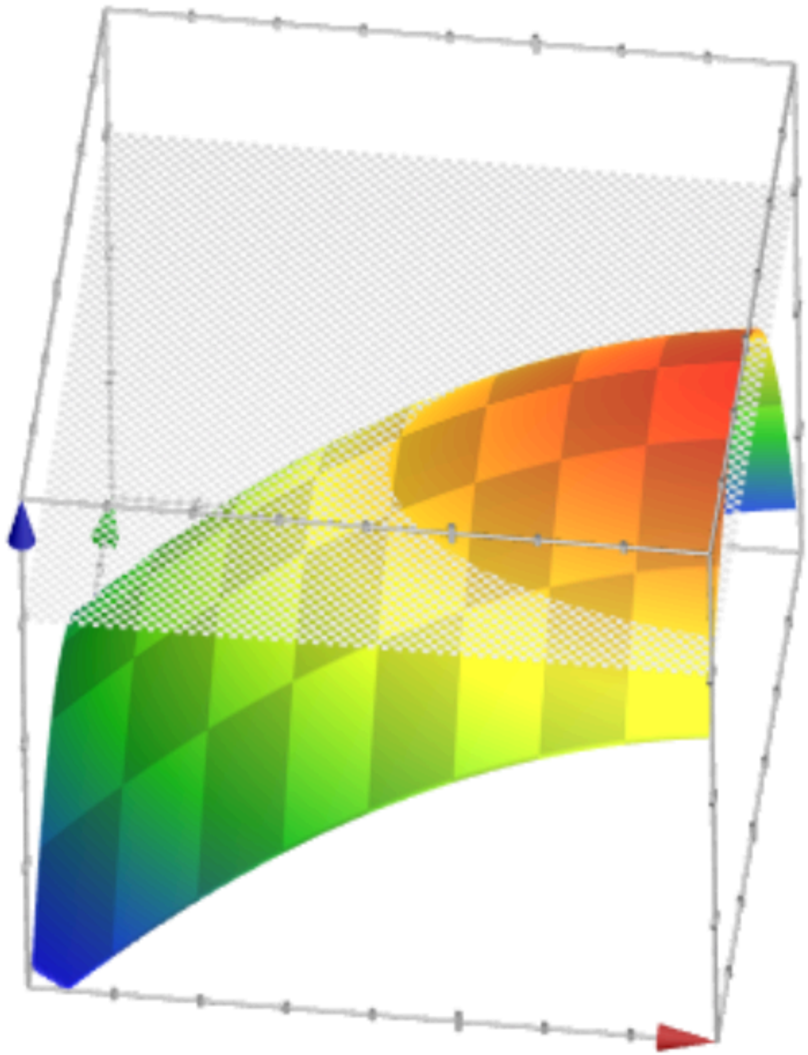


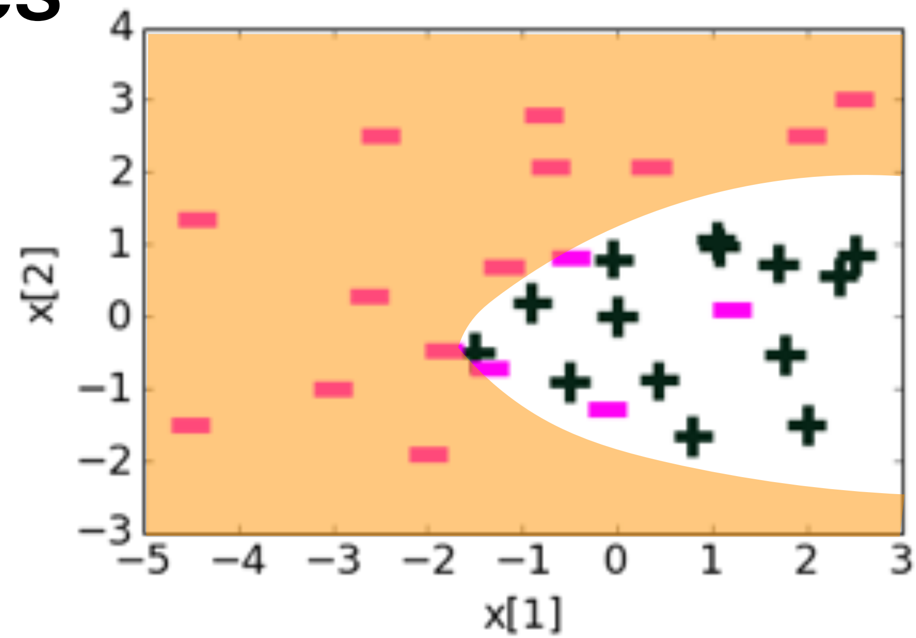


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding quadratic features

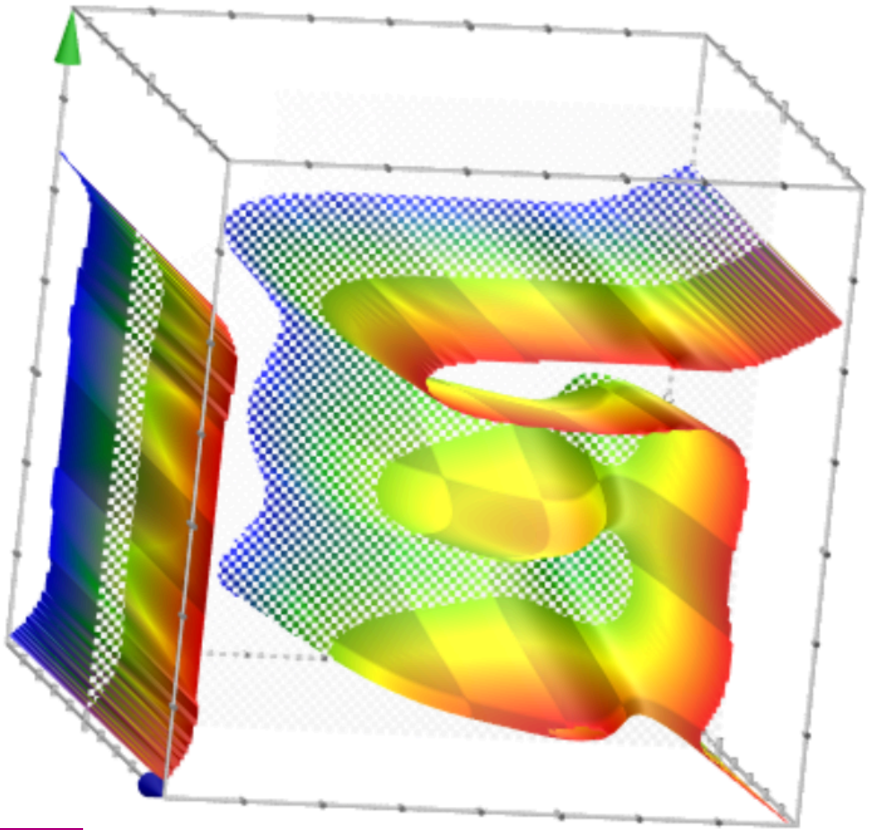




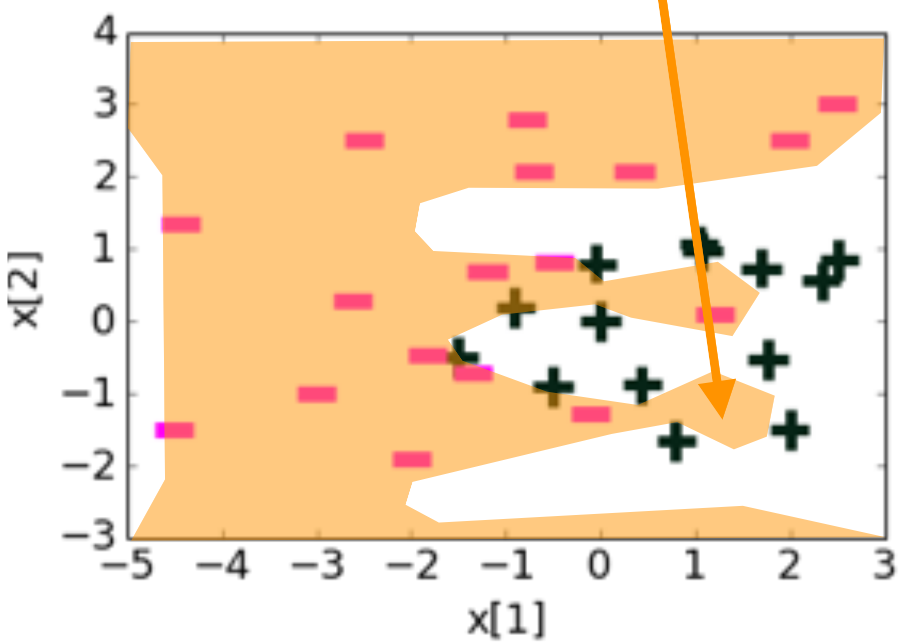
Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding higher degree polynomial features



Overfitting leads to non-generalization

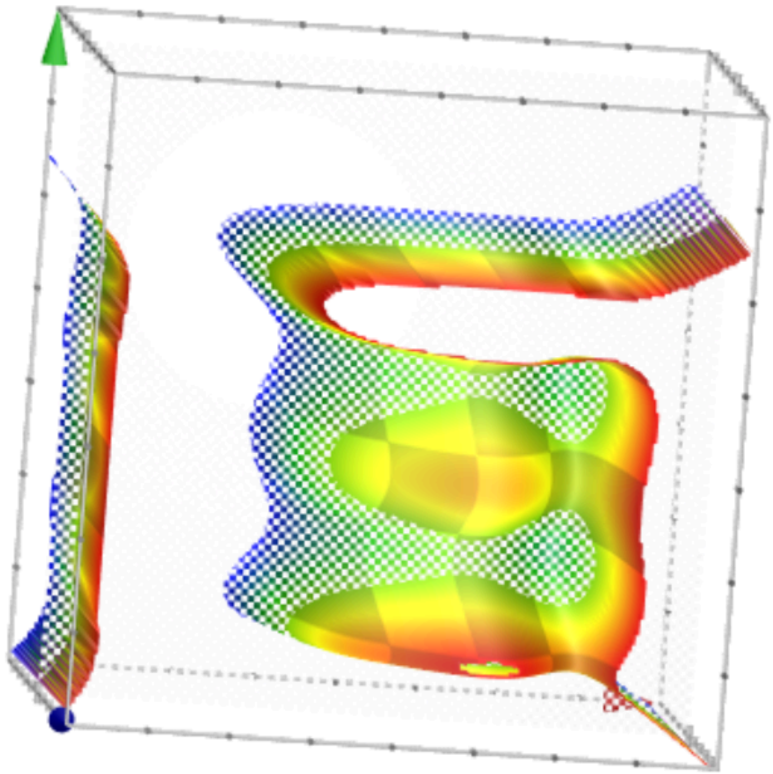


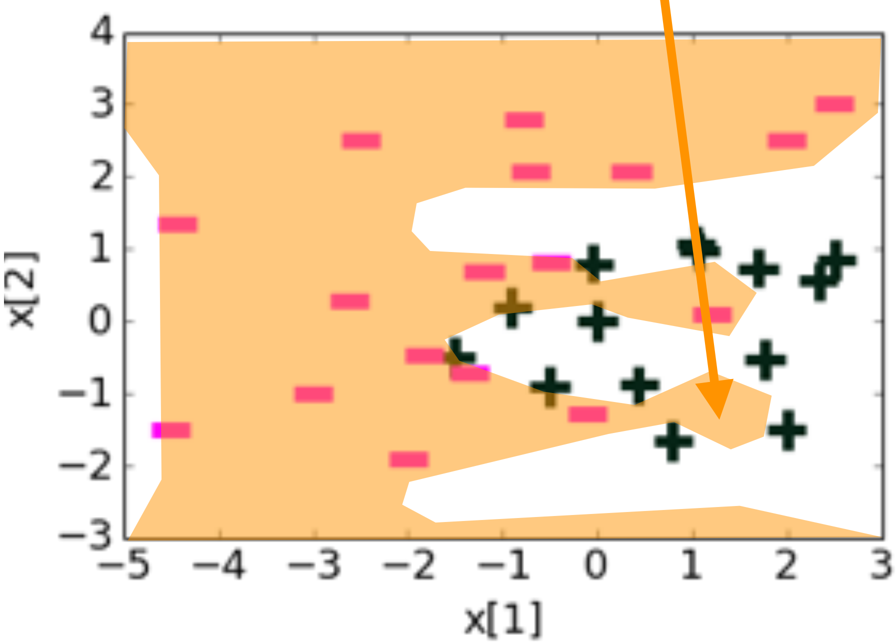
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

Adding higher degree polynomial features

Overfitting leads to non-generalization

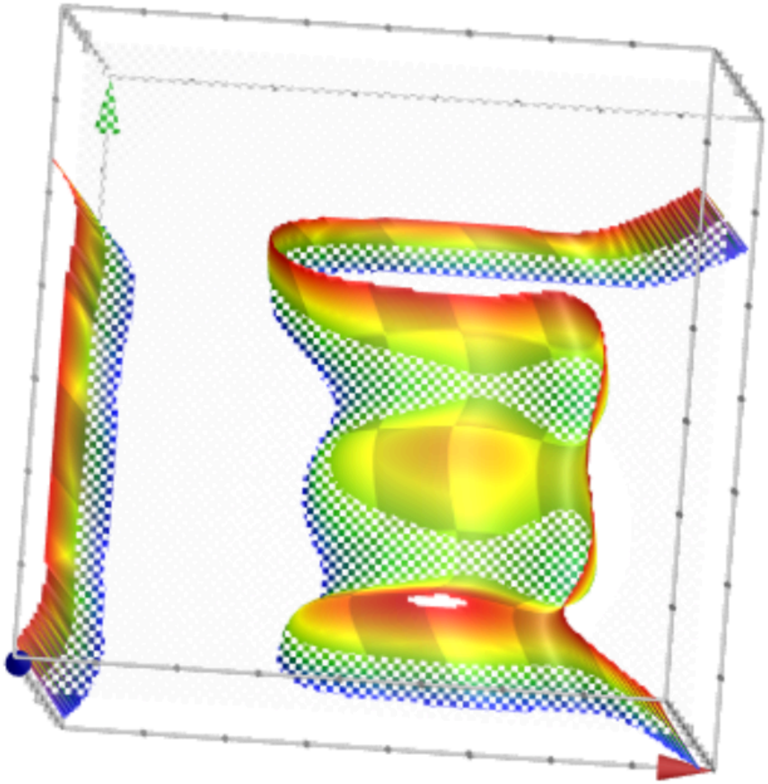




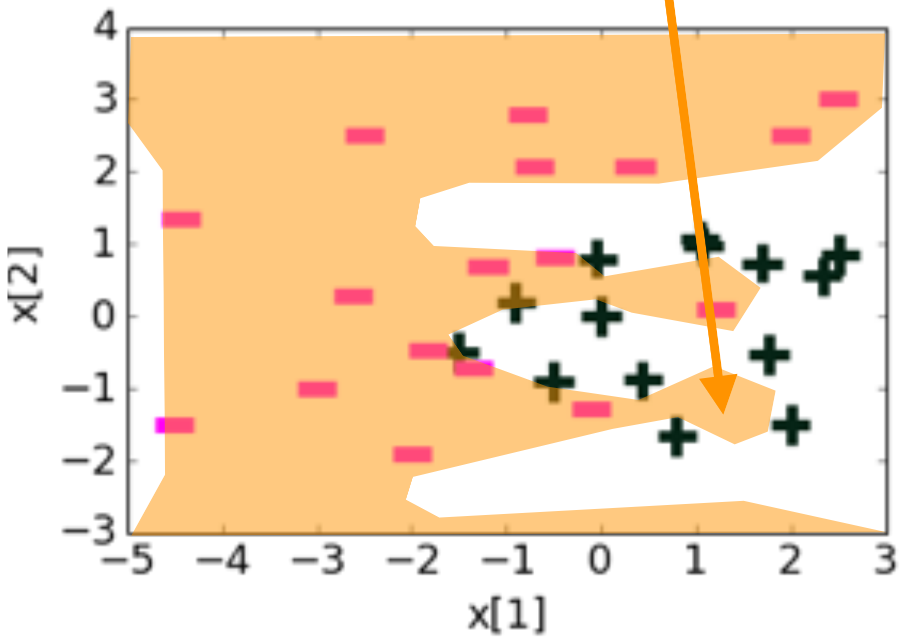
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

Adding higher degree polynomial features



Overfitting leads to non-generalization



Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

- Overfitting leads to very large values of

$$f(x) = w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1, one can generate vectors

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

- Feature space can get really large really quickly!
- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?
- At a first glance, it seems inevitable that we need memory (to store the features $\{\phi(x_i) \in \mathbb{R}^p\}_{i=1}^n$) and run-time that increases with p where $d < n \ll p$

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \to \mathbb{R}^p$ maps original data into a rich and high-dimensional feature space (usually $d \ll p$)

For example, in d=1, one can use

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_k(x) \end{bmatrix} = \begin{bmatrix} x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}$$

For example, for d>1,
one can generate vectors $\{u_j\}_{j=1}^p \subset \mathbb{R}^d$

and define features:

$$\phi_j(x) = \cos(u_j^T x)$$

$$\phi_j(x) = (u_j^T x)^2$$

$$\phi_j(x) = \frac{1}{1 + \exp(u_j^T x)}$$

- Feature space can get really large really quickly!
- How many coefficients/parameters are there for degree-k polynomials for $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$?
- At a first glance, it seems inevitable that we need memory (to store the features $\{\phi(x_i) \in \mathbb{R}^p\}_{i=1}^n$) and run-time that increases with p where $d < n \ll p$

How do we deal with high-dimensional lifts/data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

This notation is for dot product (which is the same as inner product)

How do we deal with high-dimensional lifts/data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our

How do we deal with high-dimensional lifts/data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and

How do we deal with high-dimensional lifts/data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction

How do we deal with high-dimensional lifts/data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction
- as functions of dot products of feature maps (i.e. $\{\phi(x) \cdot \phi(x')\}$)

  and if we can find a kernel for our feature map such that

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

How do we deal with high-dimensional lifts/data?

A fundamental trick in ML: use kernels

A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'.

This notation is for dot product (which is the same as inner product)

- So, if we can represent our
  - training algorithms and
  - decision rules for prediction
- as functions of dot products of feature maps (i.e. $\{\phi(x) \cdot \phi(x')\}$)
  and if we can find a kernel for our feature map such that

  $$K(x, x') = \phi(x) \cdot \phi(x')$$

then we can avoid explicitly computing and storing (high-dimensional) $\{\phi(x_i)\}_{i=1}^n$ and instead only work with the kernel matrix of the training data $\{K(x_i, x_j)\}_{i,j\in\{1,\ldots,n\}}$

An example of a kernel

$x \in \mathbb{R}^2 \rightarrow \phi(x) \in \mathbb{R}^3$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2 x_2', \sqrt{3}x_1' x_2'^2, x_2'^3)$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3) \cdot ({x'_1}^3, \sqrt{3}{x'_1}^2x'_2, \sqrt{3}x'_1{x'_2}^2, {x'_2}^3)$

$$= x_1^3 \cdot {x'_1}^3 + \sqrt{3}x_1^2x_2 \cdot \sqrt{3}{x'_1}^2x'_2 + \sqrt{3}x_1x_2^2 \cdot \sqrt{3}x'_1{x'_2}^2 + x_2^3 \cdot {x'_2}^3$$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2x_2', \sqrt{3}x_1'x_2'^2, x_2'^3)$

$$= x_1^3 \cdot x_1'^3 + \sqrt{3}x_1^2x_2 \cdot \sqrt{3}x_1'^2x_2' + \sqrt{3}x_1x_2^2 \cdot \sqrt{3}x_1'x_2'^2 + x_2^3 \cdot x_2'^3$$

$$= x_1^3 \cdot x_1'^3 + 3x_1^2x_2x_1'^2x_2' + 3x_1x_2^2x_1'x_2'^2 + x_2^3 \cdot x_2'^3$$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3) \cdot ({x'_1}^3, \sqrt{3}{x'_1}^2x'_2, \sqrt{3}x'_1{x'_2}^2, {x'_2}^3)$

$$= x_1^3 \cdot {x'_1}^3 + \sqrt{3}x_1^2x_2 \cdot \sqrt{3}{x'_1}^2x'_2 + \sqrt{3}x_1x_2^2 \cdot \sqrt{3}x'_1{x'_2}^2 + x_2^3 \cdot {x'_2}^3$$

$$= x_1^3 \cdot {x'_1}^3 + 3x_1^2x_2{x'_1}^2x'_2 + 3x_1x_2^2x'_1{x'_2}^2 + x_2^3 \cdot {x'_2}^3$$

$$= ((x_1, x_2)^\top(x'_1, x'_2))^3$$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2x_2', \sqrt{3}x_1'x_2'^2, x_2'^3)$

$$= x_1^3 \cdot x_1'^3 + \sqrt{3}x_1^2x_2 \cdot \sqrt{3}x_1'^2x_2' + \sqrt{3}x_1x_2^2 \cdot \sqrt{3}x_1'x_2'^2 + x_2^3 \cdot x_2'^3$$

$$= x_1^3 \cdot x_1'^3 + 3x_1^2x_2x_1'^2x_2' + 3x_1x_2^2x_1'x_2'^2 + x_2^3 \cdot x_2'^3$$

$$= ((x_1, x_2)^\top(x_1', x_2'))^3$$

$$= (x^\top x')^3 \quad = \quad k(x, x')$$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2 x_2', \sqrt{3}x_1' x_2'^2, x_2'^3)$

$$
\begin{aligned}
&= x_1^3 \cdot x_1'^3 + \sqrt{3}x_1^2 x_2 \cdot \sqrt{3}x_1'^2 x_2' + \sqrt{3}x_1 x_2^2 \cdot \sqrt{3}x_1' x_2'^2 + x_2^3 \cdot x_2'^3 \\
&= x_1^3 \cdot x_1'^3 + 3x_1^2 x_2 x_1'^2 x_2' + 3x_1 x_2^2 x_1' x_2'^2 + x_2^3 \cdot x_2'^3 \\
&= ((x_1, x_2)^\top (x_1', x_2'))^3 \\
&= (x^\top x')^3
\end{aligned}
$$

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
    - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2x_2', \sqrt{3}x_1'x_2'^2, x_2'^3)$

$$
\begin{aligned}
&= x_1^3 \cdot x_1'^3 + \sqrt{3}x_1^2x_2 \cdot \sqrt{3}x_1'^2x_2' + \sqrt{3}x_1x_2^2 \cdot \sqrt{3}x_1'x_2'^2 + x_2^3 \cdot x_2'^3 \\
&= x_1^3 \cdot x_1'^3 + 3x_1^2x_2x_1'^2x_2' + 3x_1x_2^2x_1'x_2'^2 + x_2^3 \cdot x_2'^3 \\
&= ((x_1, x_2)^\top(x_1', x_2'))^3 \\
&= (x^\top x')^3
\end{aligned}
$$

So yes, there is a kernel, which is efficiently computable!

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2x_2', \sqrt{3}x_1'x_2'^2, x_2'^3)$

$$= x_1^3 \cdot x_1'^3 + \sqrt{3}x_1^2x_2 \cdot \sqrt{3}x_1'^2x_2' + \sqrt{3}x_1x_2^2 \cdot \sqrt{3}x_1'x_2'^2 + x_2^3 \cdot x_2'^3$$

$$= x_1^3 \cdot x_1'^3 + 3x_1^2x_2x_1'^2x_2' + 3x_1x_2^2x_1'x_2'^2 + x_2^3 \cdot x_2'^3$$

$$= ((x_1, x_2)^\top(x_1', x_2'))^3$$

$$= (x^\top x')^3$$

So yes, there is a kernel, which is efficiently computable!
(There is always a kernel, the question is whether it can be computed more efficiently than explicitly going through this dot product calculation).

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3)$
  - Mapping 2-d vector into 3-d, with a degree-3 polynomial
- Does this map have a kernel?
- IE, does there exists a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x'?

Well, $\phi(x) \cdot \phi(x') = (x_1^3, \sqrt{3}x_1^2 x_2, \sqrt{3}x_1 x_2^2, x_2^3) \cdot (x_1'^3, \sqrt{3}x_1'^2 x_2', \sqrt{3}x_1' x_2'^2, x_2'^3)$

$$
\begin{aligned}
&= x_1^3 \cdot x_1'^3 + \sqrt{3}x_1^2 x_2 \cdot \sqrt{3}x_1'^2 x_2' + \sqrt{3}x_1 x_2^2 \cdot \sqrt{3}x_1' x_2'^2 + x_2^3 \cdot x_2'^3 \\
&= x_1^3 \cdot x_1'^3 + 3x_1^2 x_2 x_1'^2 x_2' + 3x_1 x_2^2 x_1' x_2'^2 + x_2^3 \cdot x_2'^3 \\
&= ((x_1, x_2)^\top (x_1', x_2'))^3 \\
&= (x^\top x')^3
\end{aligned}
$$

So yes, there is a kernel, which is efficiently computable!
(There is always a kernel, the question is whether it can be computed more efficiently than explicitly going through this dot product calculation).

Kernel (i.e., dot-product) of polynomial features

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly k

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point x_i, if we can make predictions (as we saw in the previous slide) by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time dn

Kernel (i.e., dot-product) of polynomial features

- Recall kernel is defined as $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle = \phi(x)^T \phi(x')$
- As illustrating examples, consider polynomial features of degree exactly k
  - $\phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ for $k = 1$ and $d = 2$, then $K(x, x') = x_1 x_1' + x_2 x_2'$
  - $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_2 x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

    then $K(x, x') = x_1^2 (x_1')^2 + x_2^2 (x_2')^2 + 2 x_1 x_2 x_1' x_2' = (x_1 x_1' + x_2 x_2')^2$
- Note that for a data point x_i, **explicitly** computing the feature $\phi(x_i)$ takes memory/time $p = d^k$
- For a data point x_i, if we can make predictions (as we saw in the previous slide) by only computing the kernel, then computing $\{K(x_i, x_j)\}_{j=1}^n$ takes memory/time dn
  - The features are **implicit** and accessed only via kernels, making it efficient

Ridge Linear Regression as Kernels

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$ → closed form solution

$$= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$$

(when $n < d$ via linear algebra)

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d \times d})^{-1}\mathbf{X}^T\mathbf{y}$

$$= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1}\mathbf{y}$$

(when $n < d$ via linear algebra)

- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

$$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$

$$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1}\mathbf{X}x_{\text{new}}$$

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

  $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$

  $= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$
- Hence, to make prediction on any future data points, all we need to know is

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$
  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$
  $$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$
  $$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$
- Hence, to make prediction on any future data points, all we need to know is
- $$\mathbf{X}x_{\text{new}} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n, \text{ and } \mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$$

Ridge Linear Regression as Kernels

- Consider Ridge regression: $\hat{w} = \arg\min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with $\widehat{w}$ using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\widehat{w} = (\mathbf{X}^T\mathbf{X} + \lambda \mathbf{I}_{d\times d})^{-1}\mathbf{X}^T\mathbf{y}$

  $= \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{y}$ (when $n < d$ via linear algebra)
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$

  $$\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}}$$

  $$= \mathbf{y}^T(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I}_{n\times n})^{-1}\mathbf{X}x_{\text{new}}$$
- Hence, to make prediction on any future data points, all we need to know is
- $$\mathbf{X}x_{\text{new}} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n, \text{ and } \mathbf{X}\mathbf{X}^T = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ \vdots & \vdots & \\ K(x_n, x_1) & K(x_n, x_2) & \cdots \end{bmatrix} \in \mathbb{R}^{n\times n}$$
- Even if we run ridge linear regression on feature map $\phi(x) \in \mathbb{R}^p$, we only need to access the features via kernel $K(x_i, x_j)$ and $K(x_i, x_{\text{new}})$ and not the features $\phi(x_i)$

The Kernel Trick

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\widehat{w} = \sum_{i=1}^n \alpha_i x_i \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\widehat{w} = \sum_{i=1}^n \alpha_i x_i \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\widehat{w} = \sum_{i=1}^n \alpha_i x_i \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^n \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$
3. Substitute $x_i^T x_j$ with $K(x_i, x_j)$, and find α using the above algorithm from step 2.

The Kernel Trick

- Given data $\{(x_i, y_i)\}_{i=1}^n$, pick a kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$

1. For a choice of a loss, use a linear predictor of the form

$$\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i \quad \text{for some } \alpha = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} \in \mathbb{R}^n \text{ to be learned}$$

   Prediction is $\widehat{y}_{\text{new}} = \widehat{w}^T x_{\text{new}} = \sum_{i=1}^{n} \alpha_i \, x_i^T x_{\text{new}}$

2. Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$
3. Substitute $x_i^T x_j$ with $K(x_i, x_j)$, and find α using the above algorithm from step 2.
4. Make prediction with $\widehat{y}_{\text{new}} = \sum_{i=1}^{n} \alpha_i K(x_i, x_{\text{new}})$

   (replacing $x_i^T x_{\text{new}}$ with $K(x_i, x_{\text{new}})$)

The Kernel Trick for regularized least squares

ℓ^2

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

(Step 1. Use a linear predictor)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

(Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} \Big(y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle\Big)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(transformed optimization)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$= \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j)$$

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j)$$

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_w \sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^n \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_\alpha \sum_{i=1}^n (y_i - \sum_{j=1}^n \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

Where $\mathbf{P}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$

(Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

Where $\mathbf{P}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

(Solve for $\widehat{\alpha}_{\text{kernel}}$)

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

Where $\mathbf{P}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

(Solve for $\widehat{\alpha}_{\text{kernel}}$)

Thus, $\widehat{\alpha}_{\text{kernel}} = (\mathbf{P} + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y}$

The Kernel Trick for regularized least squares

$$\widehat{w} = \arg\min_{w} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \lambda \|w\|_2^2$$

There exists an $\alpha \in \mathbb{R}^n$: $\widehat{w} = \sum_{i=1}^{n} \alpha_i x_i$ (Step 1. Use a linear predictor)

$$\widehat{\alpha} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j \langle x_j, x_i \rangle)^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle x_i, x_j \rangle$$

(Step 2. Write an algorithm in terms of $\widehat{\alpha}$)

$$\widehat{\alpha}_{\text{kernel}} = \arg\min_{\alpha} \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} \alpha_j K(x_i, x_j))^2 + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j)$$

(Step 3. Switch inner product with kernel)

$$= \arg\min_{\alpha} ||\mathbf{y} - \mathbf{P}\alpha||_2^2 + \lambda \alpha^{\top} \mathbf{P} \alpha$$

Where $\mathbf{P}_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$

(Solve for $\widehat{\alpha}_{\text{kernel}}$)

Thus, $\widehat{\alpha}_{\text{kernel}} = (\mathbf{P} + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y}$

Examples of popular Kernels

- **Polynomials of degree exactly k**

$$K(x, x') = (x^T x')^k$$

- **Polynomials of degree up to k**

$$K(x, x') = (1 + x^T x')^k$$

- **Gaussian (squared exponential) kernel**
  (a.k.a RBF kernel for Radial Basis Function)

$$K(x, x') = \exp\left(- \frac{\|x - x'\|_2^2}{2\sigma^2} \right)$$

- **Sigmoid**

$$K(x, x') = \tanh(\gamma x^T x' + r)$$