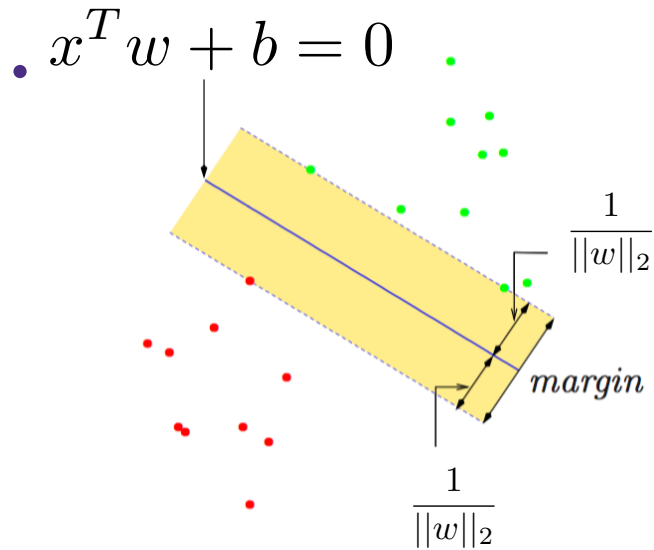


Kernels

W

What if the data is not linearly separable?



Some points do not satisfy margin constraint:

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

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

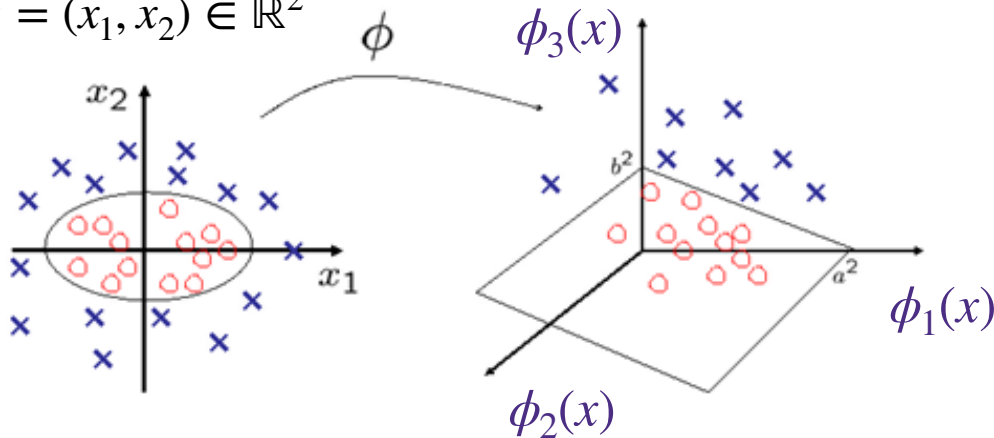
Two options:

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,

$$x = (x_1, x_2) \in \mathbb{R}^2$$



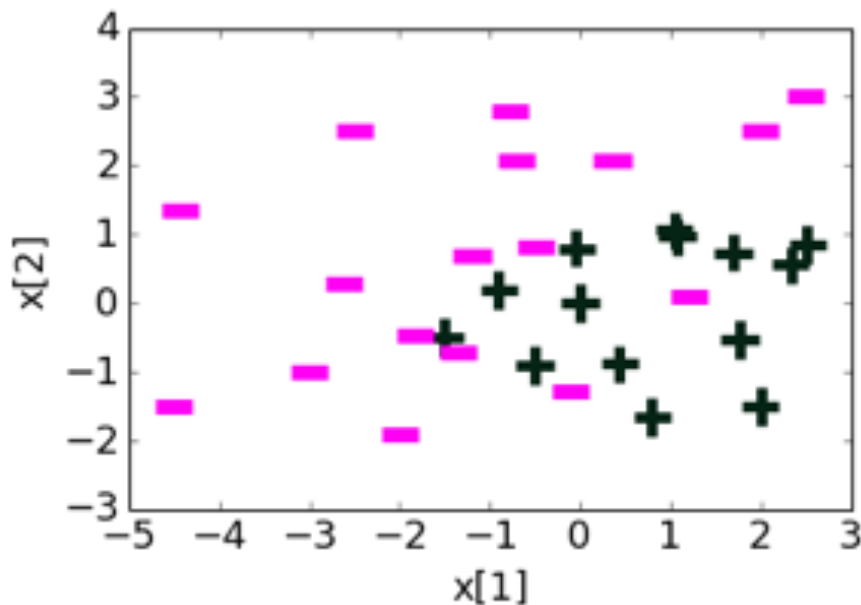
This data is not linearly separable

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



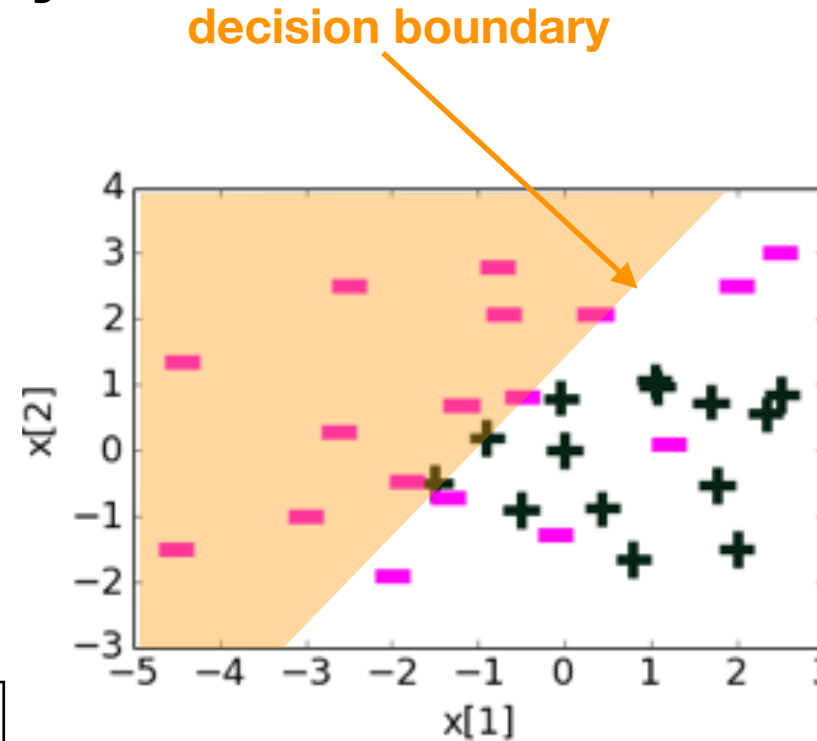
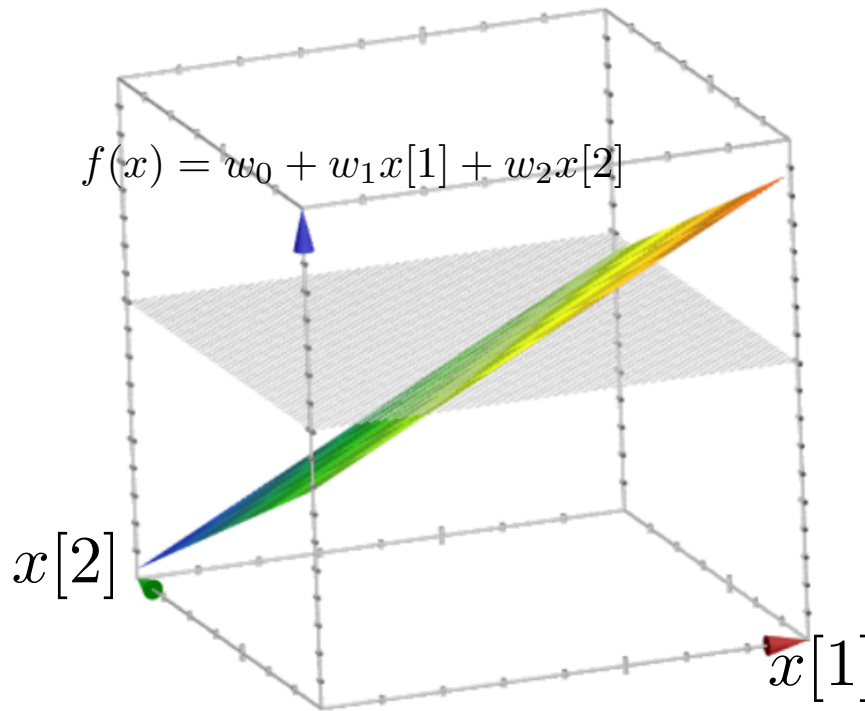
Polynomial
features

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

- data: \mathbf{x} in 2-dimensions, \mathbf{y} in $\{+1, -1\}$
- features: polynomials
- model: linear on polynomial features

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

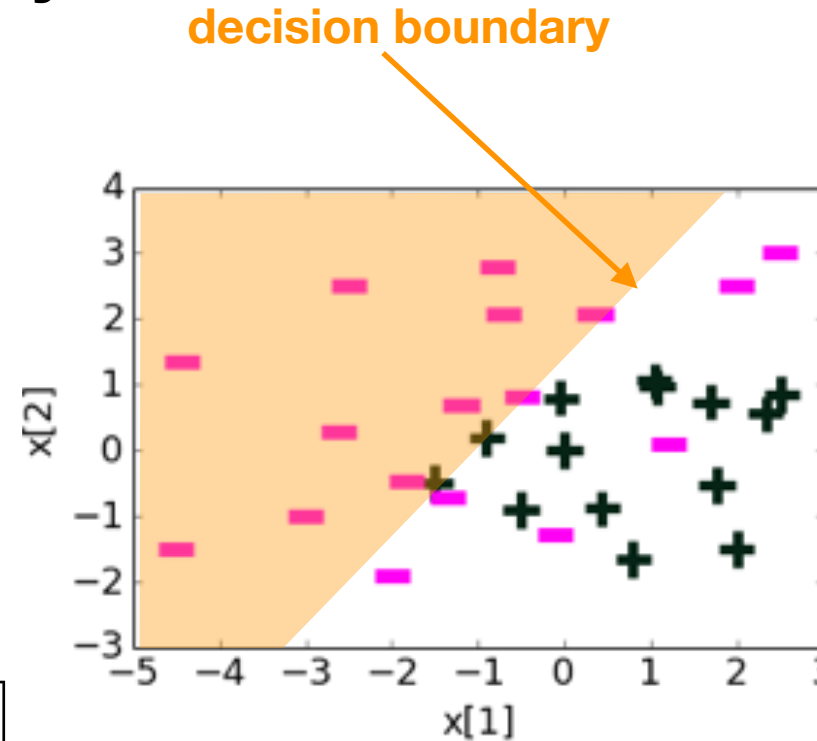
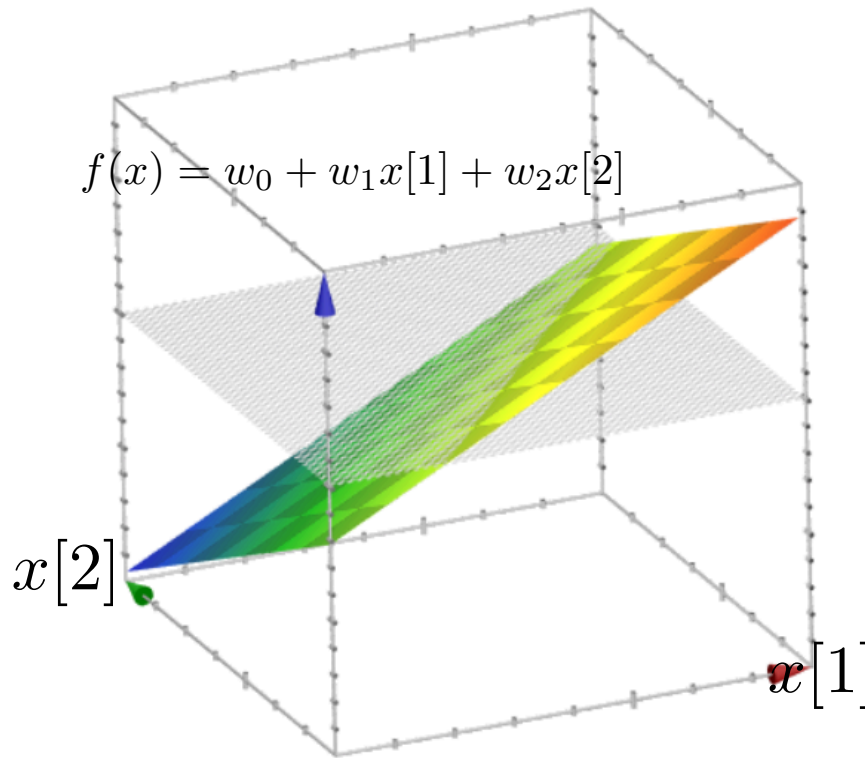
Learned decision boundary



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**

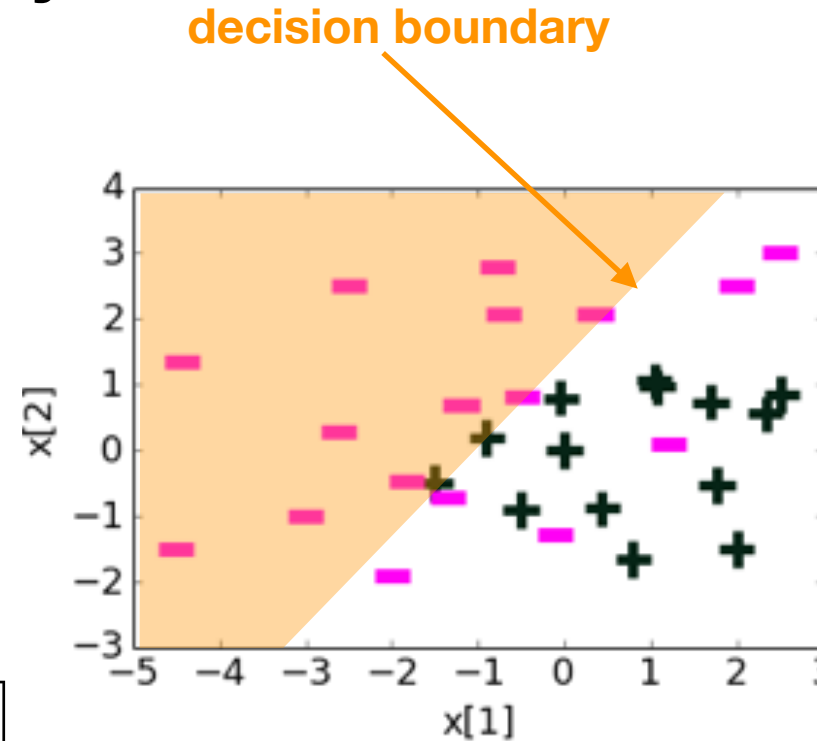
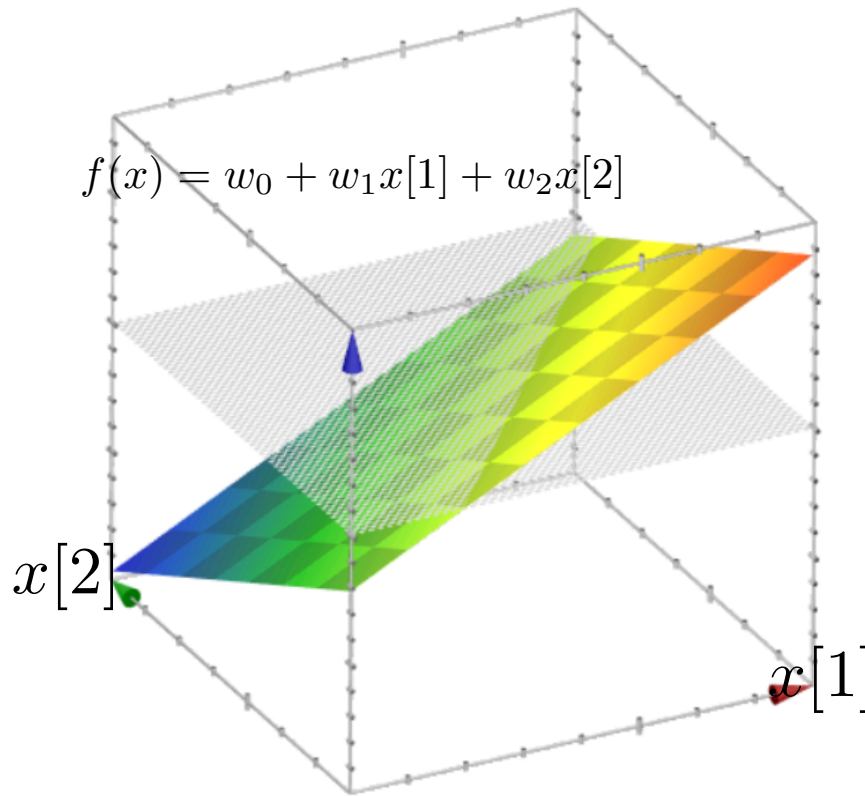
Learned decision boundary



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**

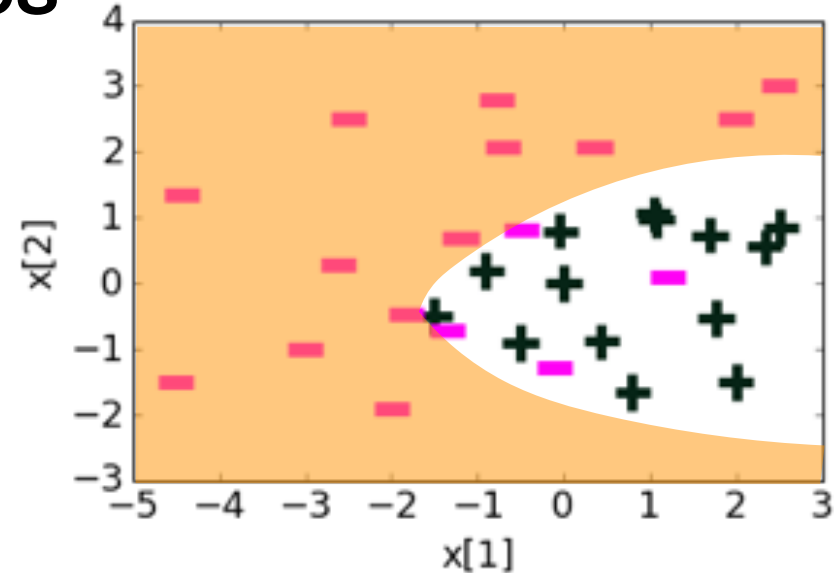
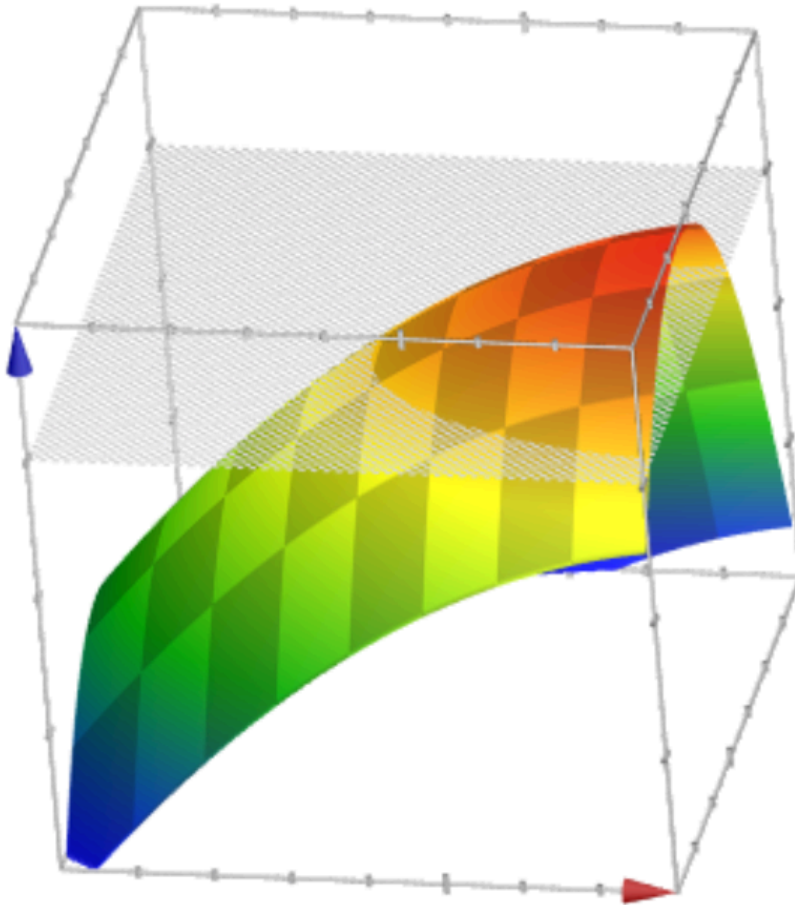
Learned decision boundary



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**

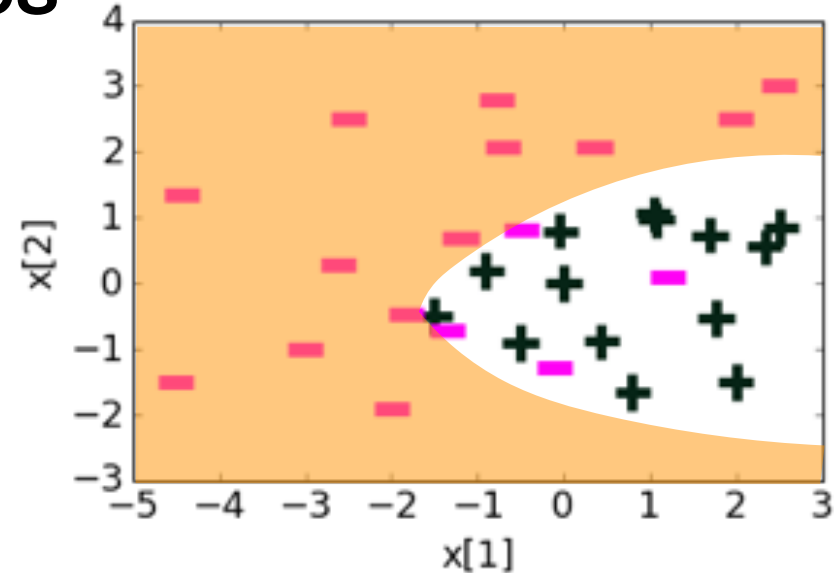
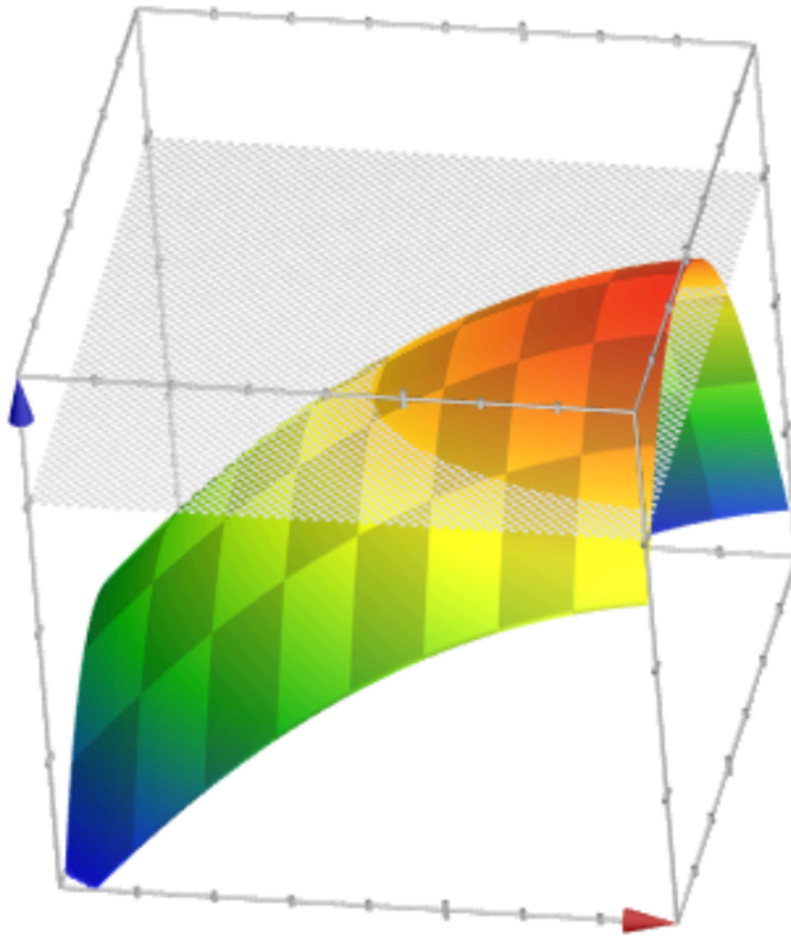
Adding quadratic features



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

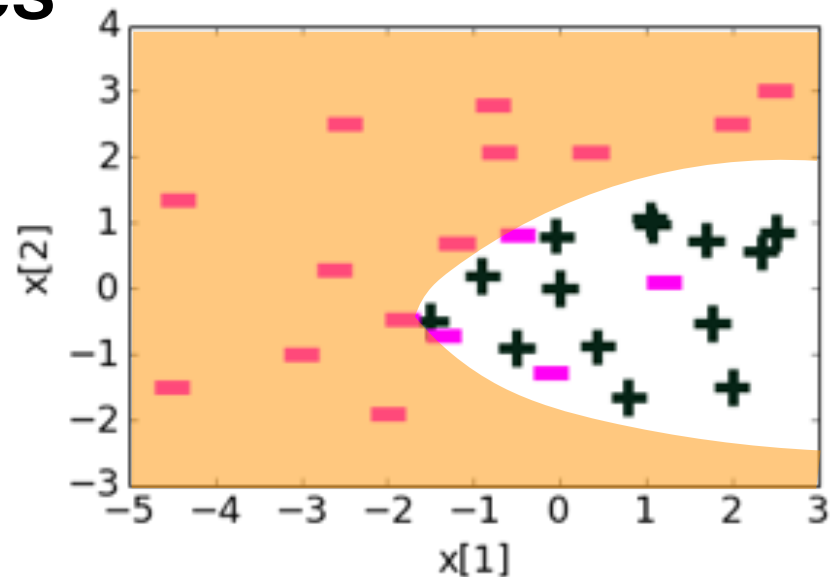
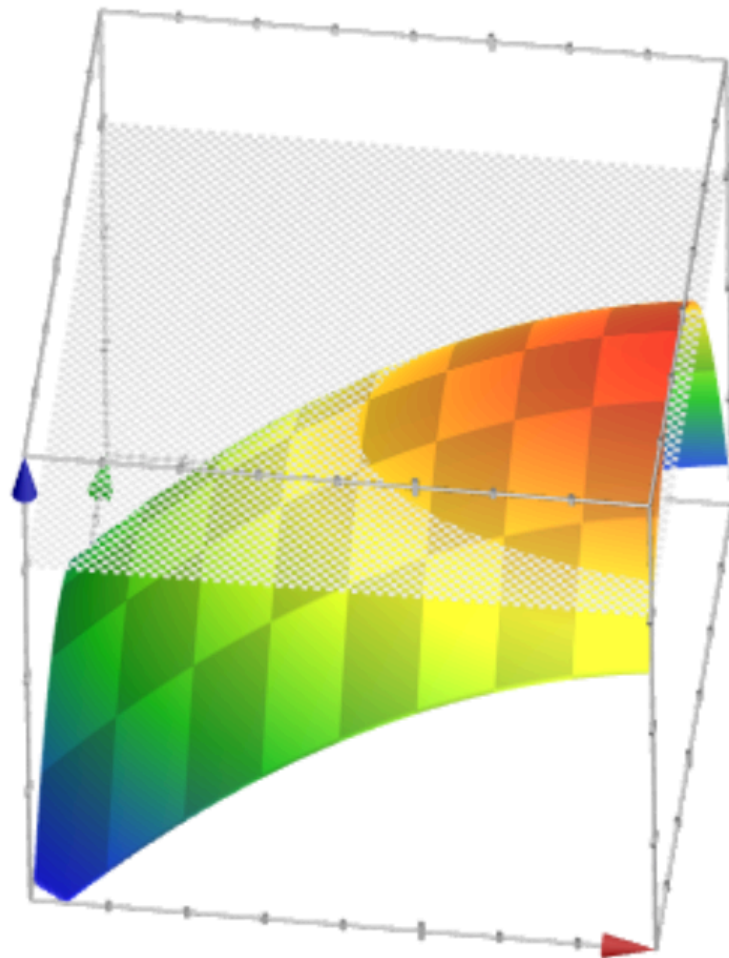
Adding quadratic features



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

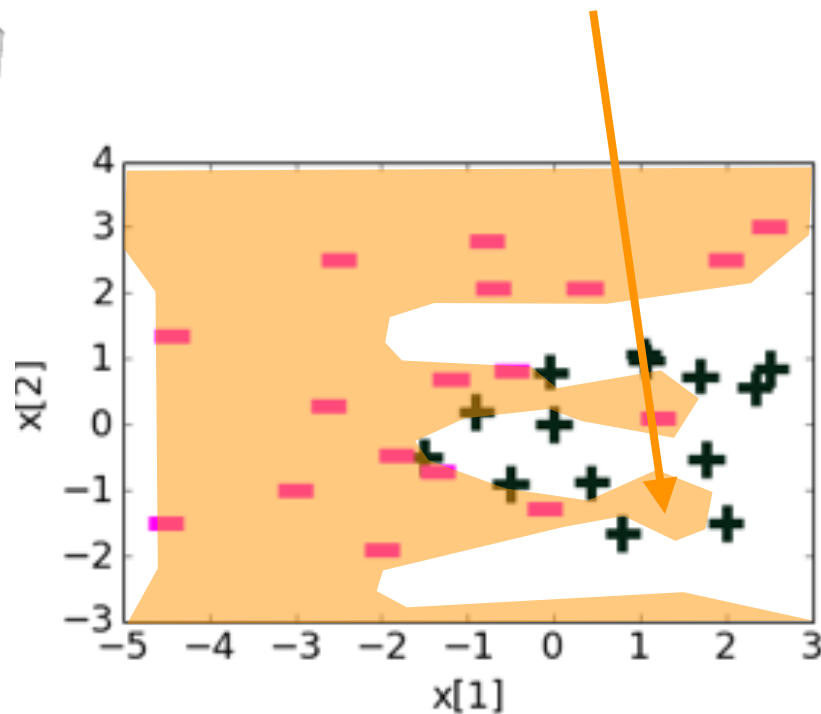
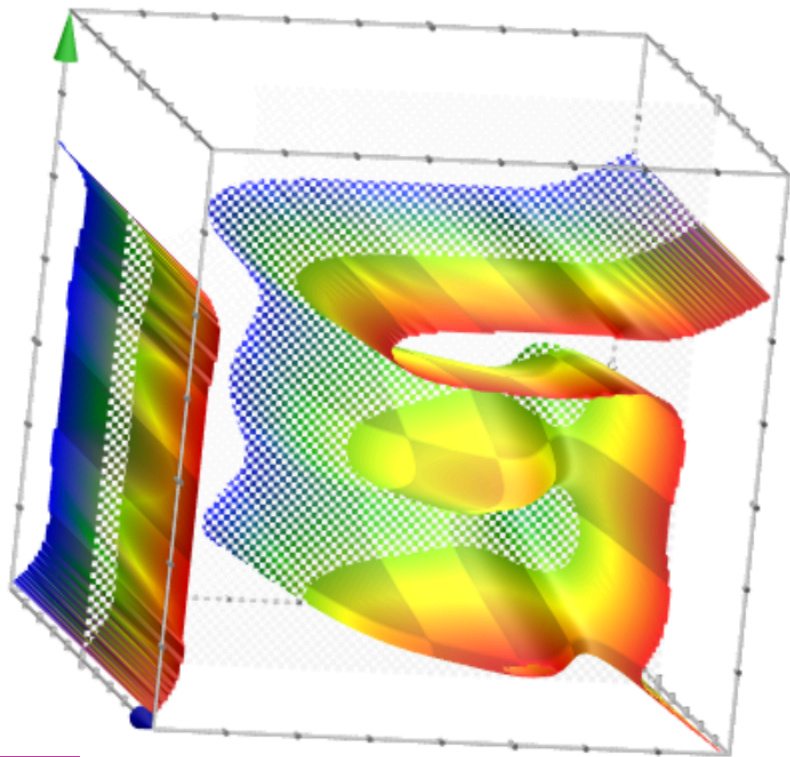


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	$x[1]$	1.39
$h_2(x)$	$x[2]$	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	$x[1]x[2]$	Omitted

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

Adding higher degree polynomial features

Overfitting leads to non-generalization

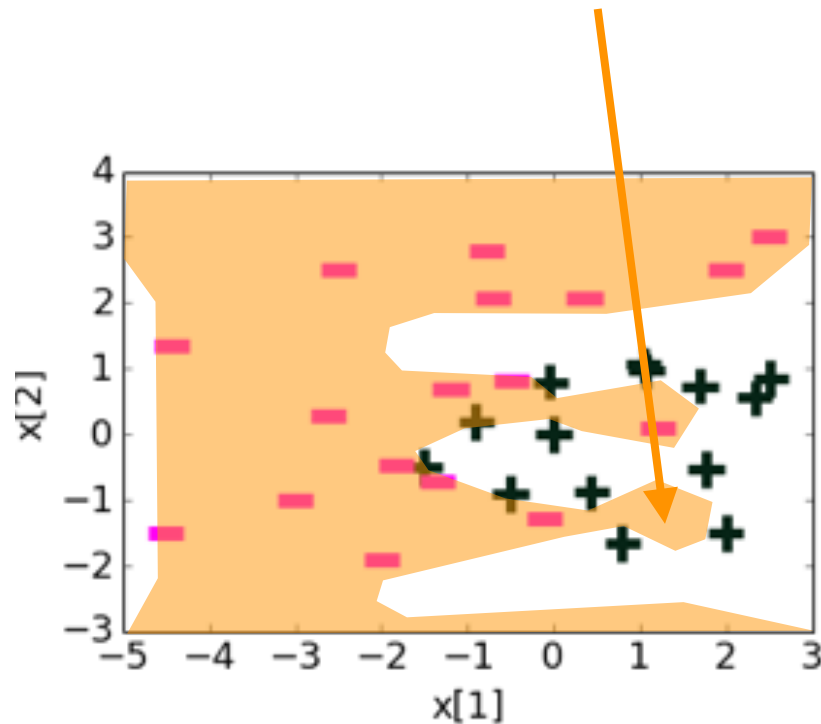
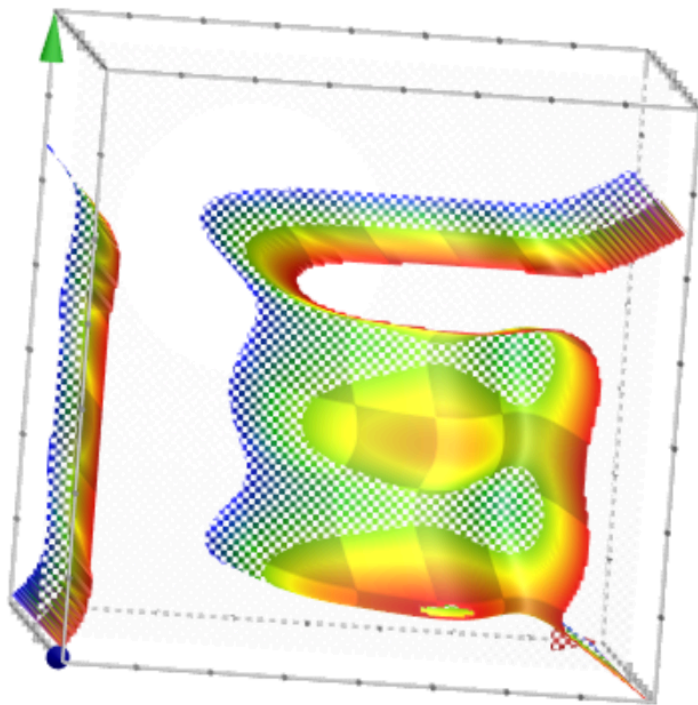


Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	$x[1]$	5.3
$h_2(x)$	$x[2]$	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$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

Coefficient values getting large

Adding higher degree polynomial features

Overfitting leads to non-generalization

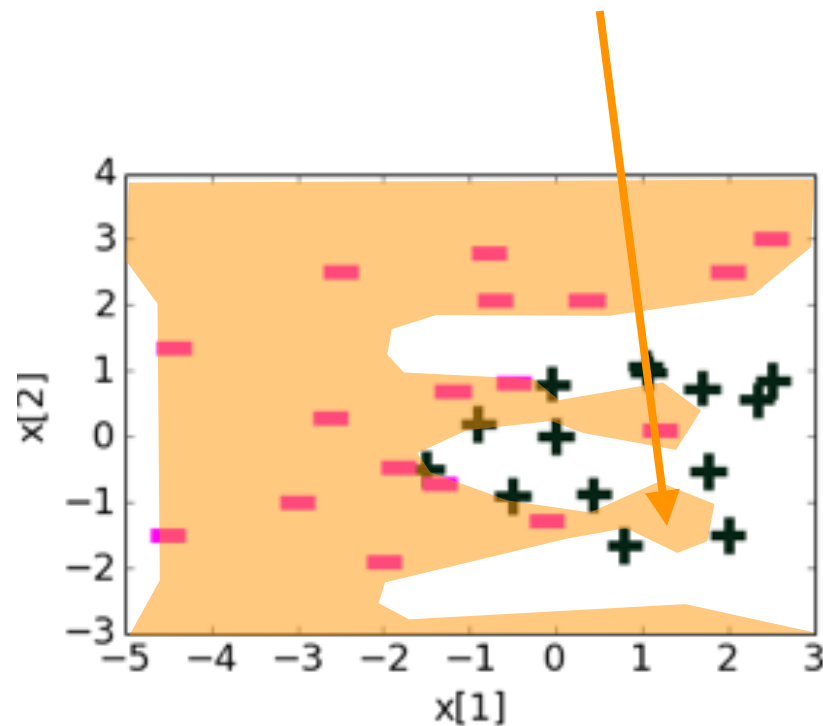
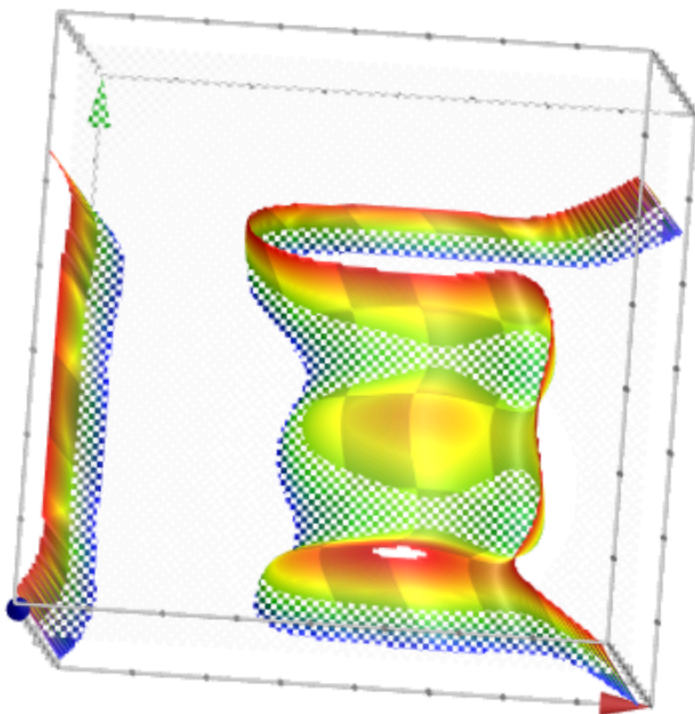


Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	$x[1]$	5.3
$h_2(x)$	$x[2]$	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	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 $f(x) = w_0h_0(x) + w_1h_1(x) + w_2h_2(x) + \dots$

Creating Features

- Feature mapping $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ 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 $x = (x_1, \dots, x_d) \in \mathbb{R}^d$?
- 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 $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$ 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 $x = (x_1, \dots, x_d) \in \mathbb{R}^d$?
- 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ 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

$$\{K(x_i, x_j)\}_{i,j \in \{1, \dots, n\}}$$

An example of a kernel

An example of a kernel

- Suppose we have the map $\phi(x) = (x_1^3, \sqrt{3}x_1^2x_2, \sqrt{3}x_1x_2^2, x_2^3)$

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

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?

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 exist 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 exist 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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 exist 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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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).

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 exist a $K : \mathbb{R}^2 \times \mathbb{R}^2$ such that $K(x, x') = \phi(x) \cdot \phi(x')$ for all x, x' ?

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

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 $K(x, x') = x_1x'_1 + x_2x'_2$

- $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \\ x_2x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

then $K(x, x') = x_1^2(x'_1)^2 + x_2^2(x'_2)^2 + 2x_1x_2x'_1x'_2 = (x_1x'_1 + x_2x'_2)^2$

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 $K(x, x') = x_1x'_1 + x_2x'_2$

- $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \\ x_2x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

then $K(x, x') = x_1^2(x'_1)^2 + x_2^2(x'_2)^2 + 2x_1x_2x'_1x'_2 = (x_1x'_1 + x_2x'_2)^2$

- 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 $K(x, x') = x_1x'_1 + x_2x'_2$

- $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \\ x_2x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

then $K(x, x') = x_1^2(x'_1)^2 + x_2^2(x'_2)^2 + 2x_1x_2x'_1x'_2 = (x_1x'_1 + x_2x'_2)^2$

- 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 $K(x, x') = x_1x'_1 + x_2x'_2$

- $\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1x_2 \\ x_2x_1 \end{bmatrix}$ for $k = 2$ and $d = 2$,

then $K(x, x') = x_1^2(x'_1)^2 + x_2^2(x'_2)^2 + 2x_1x_2x'_1x'_2 = (x_1x'_1 + x_2x'_2)^2$

- 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 \hat{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 \hat{w} using linear kernel defined as $K(x, x') = x^T x'$
- Training:
$$\begin{aligned} \hat{w} &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_{d \times d})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y} \end{aligned} \quad (\text{when } n < d \text{ via linear algebra})$$

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 \hat{w} using linear kernel defined as $K(x, x') = x^T x'$
- Training:
$$\begin{aligned}\hat{w} &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_{d \times d})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y} \quad (\text{when } n < d \text{ via linear algebra})\end{aligned}$$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$
$$\begin{aligned}\hat{y}_{\text{new}} &= \hat{w}^T x_{\text{new}} \\ &= \mathbf{y}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{X} x_{\text{new}}\end{aligned}$$

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 \hat{w} using linear kernel defined as $K(x, x') = x^T x'$
- Training:
$$\begin{aligned}\hat{w} &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_{d \times d})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y} \quad (\text{when } n < d \text{ via linear algebra})\end{aligned}$$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$
$$\begin{aligned}\hat{y}_{\text{new}} &= \hat{w}^T x_{\text{new}} \\ &= \mathbf{y}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{X} x_{\text{new}}\end{aligned}$$
- Hence, to make prediction on any future data points, all we need to know is

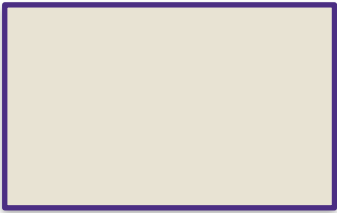
Ridge Linear Regression as Kernels

- 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 \hat{w} using linear kernel defined as $K(x, x') = x^T x'$
- Training:
$$\begin{aligned}\hat{w} &= (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_{d \times d})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{y} \quad (\text{when } n < d \text{ via linear algebra})\end{aligned}$$
- Prediction: $x_{\text{new}} \in \mathbb{R}^d$
$$\begin{aligned}\hat{y}_{\text{new}} &= \hat{w}^T x_{\text{new}} \\ &= \mathbf{y}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I}_{n \times n})^{-1} \mathbf{X} x_{\text{new}}\end{aligned}$$
- Hence, to make prediction on any future data points, all we need to know is
 - $\mathbf{X} x_{\text{new}} = \begin{bmatrix} K(x_1, x_{\text{new}}) \\ \vdots \\ K(x_n, x_{\text{new}}) \end{bmatrix} \in \mathbb{R}^n$, 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

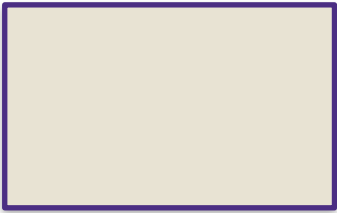
- Consider Ridge regression: $\hat{w} = \arg \min_{w \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}w\|_2^2 + \lambda \|w\|_2^2$
- As an exercise, we will represent prediction with \hat{w} using linear kernel defined as $K(x, x') = x^T x'$
- Training: $\hat{w} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}_{d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



The Kernel Trick

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



The Kernel Trick

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

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

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

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



The Kernel Trick

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

- For a choice of a loss, use a linear predictor of the form

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

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

- 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

- For a choice of a loss, use a linear predictor of the form

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

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

- Design an algorithm that finds α while accessing the data only via $\{x_i^T x_j\}$
- 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 $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$

- For a choice of a loss, use a linear predictor of the form

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

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

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

- Make prediction with $\hat{y}_{\text{new}} = \sum_{i=1}^n \alpha_i K(x_i, x_{\text{new}})$

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

The Kernel Trick for regularized least squares

The Kernel Trick for regularized least squares

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

The Kernel Trick for regularized least squares

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

(Step 1. Use a linear predictor)

The Kernel Trick for regularized least squares

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

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

(Step 1. Use a linear predictor)

The Kernel Trick for regularized least squares

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

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

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

The Kernel Trick for regularized least squares

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

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

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

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

The Kernel Trick for regularized least squares

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

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

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

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

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

The Kernel Trick for regularized least squares

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

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

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

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

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

The Kernel Trick for regularized least squares

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

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

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

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

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

(Step 3. Switch inner product with kernel)

The Kernel Trick for regularized least squares

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

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

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

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

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

(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

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

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

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

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

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

(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 $\hat{\alpha}_{\text{kernel}}$)

The Kernel Trick for regularized least squares

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

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

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

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

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

(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 $\hat{\alpha}_{\text{kernel}}$)

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

The Kernel Trick for regularized least squares

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

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

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

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

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

(Step 3. Switch inner product with kernel)

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

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

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

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

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)

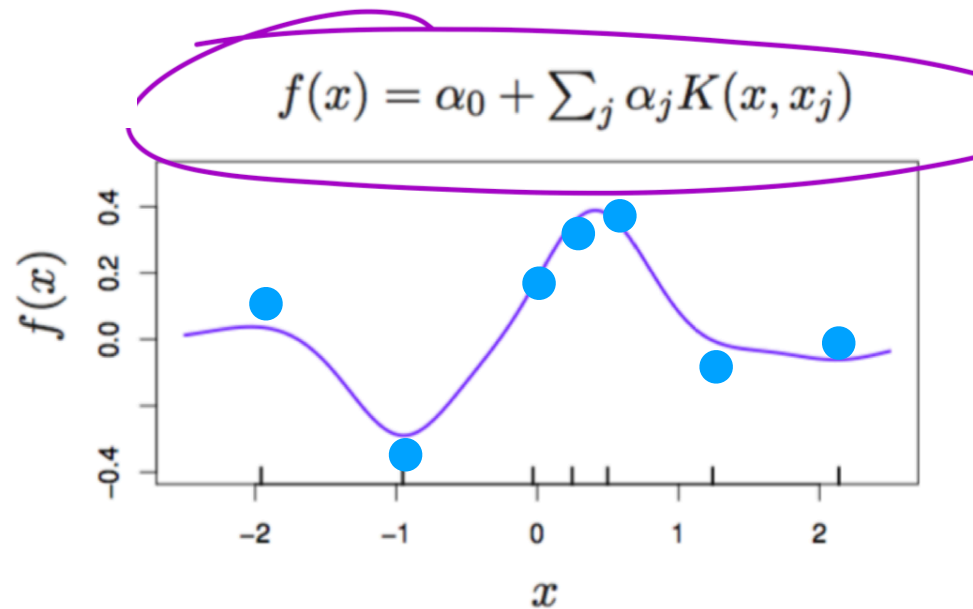
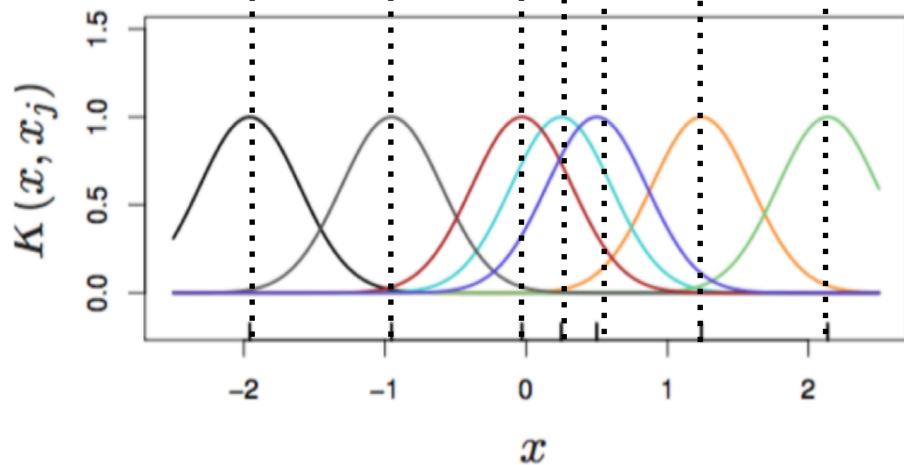
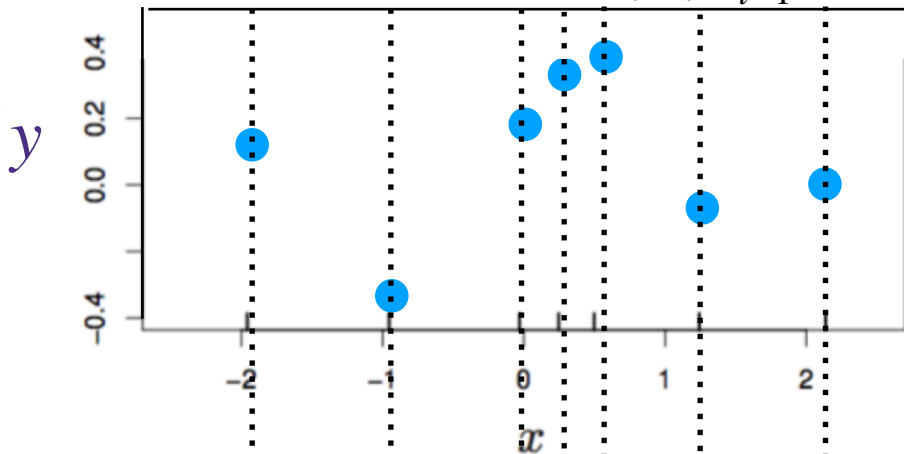
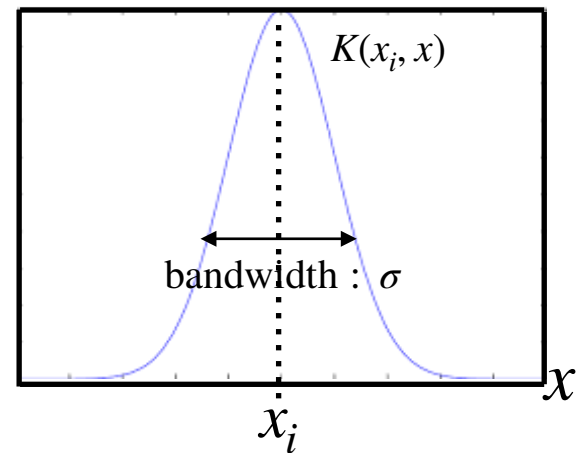
$$K(x, x') = \exp\left(-\frac{\|x - x'\|_2^2}{2\sigma^2}\right)$$

- **Sigmoid**

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

RBF kernel $k(x_i, x) = \exp\left\{-\frac{\|x_i - x\|_2^2}{2\sigma^2}\right\}$

samples $\{(x_i, y_i)\}_{i=1}^n$



- predictor $f(x) = \sum_{i=1}^n \alpha_i K(x_i, x)$ is taking weighted sum of n kernel functions centered at each sample points

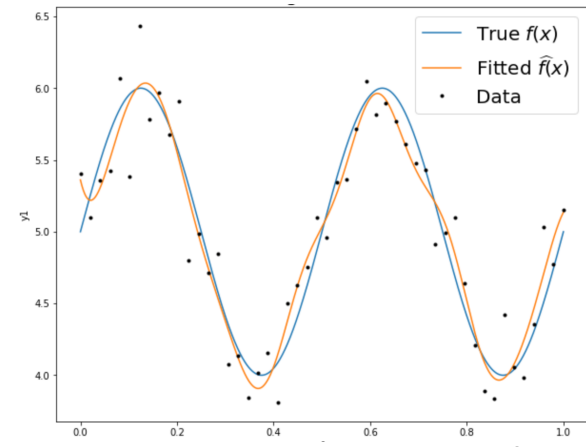
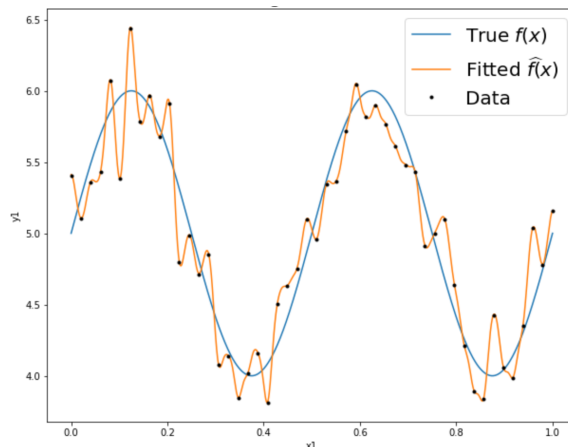
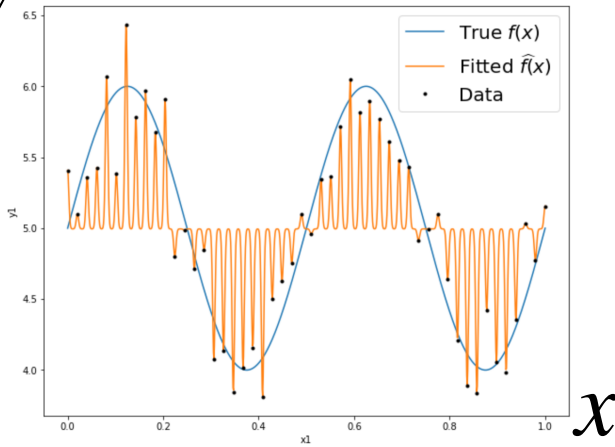
RBF kernel $k(x_i, x) = \exp\left\{-\frac{\|x_i - x\|_2^2}{2\sigma^2}\right\}$

- $\mathcal{L}(\alpha) = \|\mathbf{P}\alpha - \mathbf{y}\|_2^2 + \lambda\alpha^T\mathbf{P}\alpha$
- The bandwidth σ^2 of the kernel regularizes the predictor, and the regularization coefficient λ also regularizes the predictor

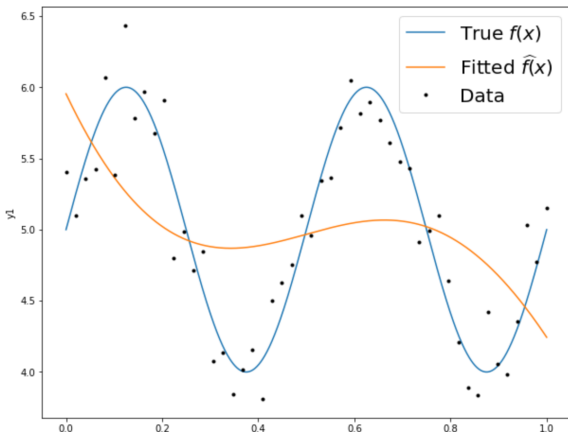
$\sigma = 10^{-3} \quad \lambda = 10^{-4}$

$\sigma = 10^{-2} \quad \lambda = 10^{-4}$

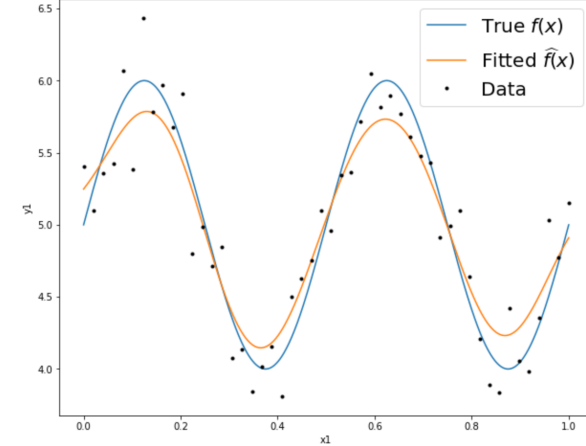
$\sigma = 10^{-1} \quad \lambda = 10^{-4}$



$\sigma = 10^{-0} \quad \lambda = 10^{-4}$



$\sigma = 10^{-1} \quad \lambda = 10^{-0}$



$$\hat{f}(x) = \sum_{i=1}^n \hat{\alpha}_i K(x_i, x)$$

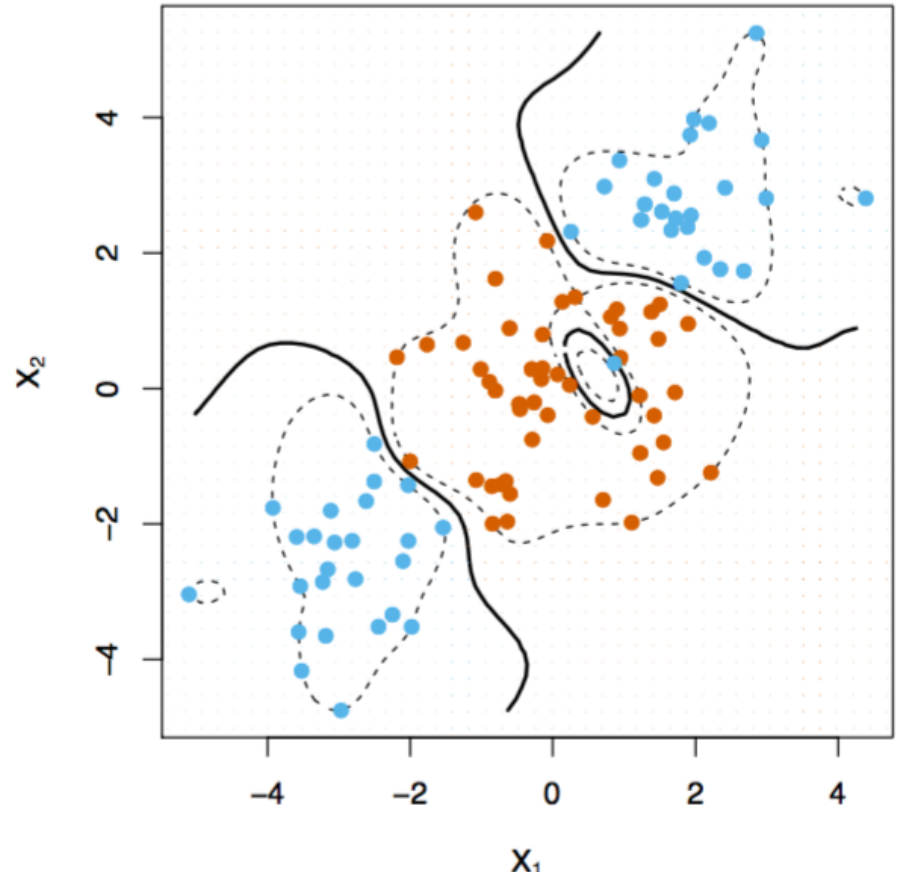
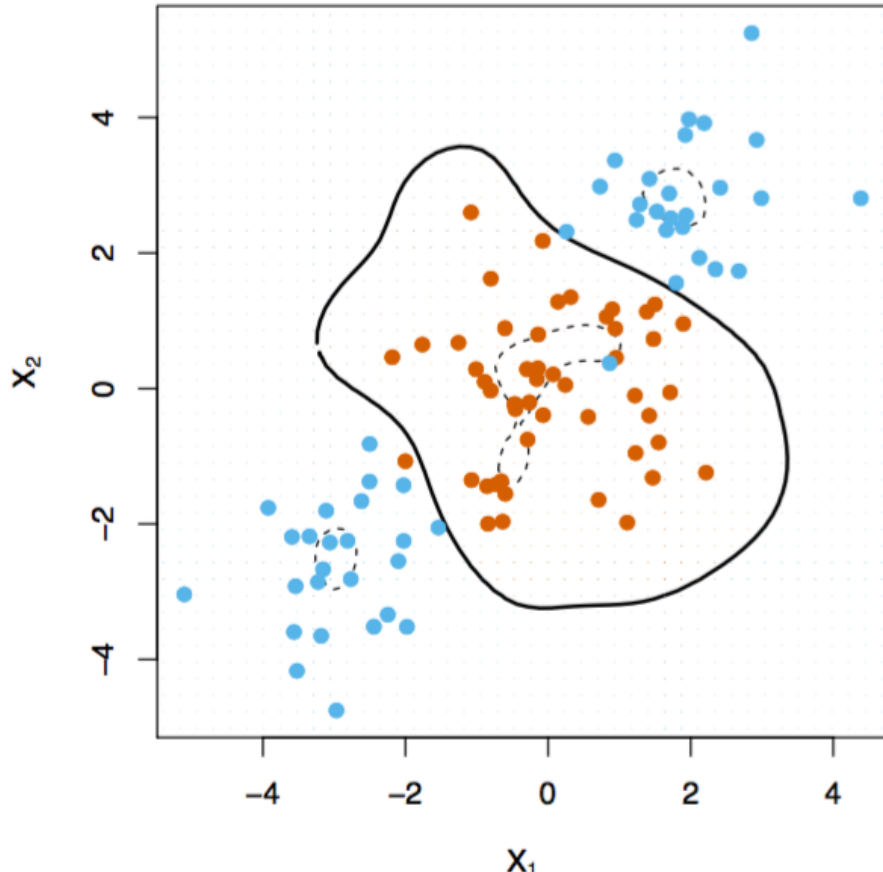
RBF kernel for SVMs

$$\hat{w} = \arg \min_{w,b} \frac{1}{n} \sum_{i=1}^n \max\{0, 1 - y_i(b + w^T x_i)\} + \lambda \|w\|_2^2$$

$$\hat{\alpha}, \hat{b} = \arg \min_{\alpha \in \mathbb{R}^n, b} \frac{1}{n} \sum_{i=1}^n \max\{0, 1 - y_i(b + \sum_{j=1}^n \alpha_j K(x_j, x_i))\} + \lambda \sum_{i=1, j=1}^n \alpha_i \alpha_j K(x_i, x_j)$$

Bandwidth σ is large enough

Bandwidth σ is small



Bootstrap

W

Confidence intervals

- Suppose you have training data $\{(x_i, y_i)\}_{i=1}^n$ drawn i.i.d. from some true distribution $P_{x,y}$

- We train a kernel ridge regressor, with some choice of a kernel

$$K : \mathbb{R}^{d \times d} \rightarrow \mathbb{R}$$

$$\text{minimize}_{\alpha} \|\mathbf{P}\alpha - \mathbf{y}\|_2^2 + \lambda \alpha^T \mathbf{P}\alpha$$

- The resulting predictor is

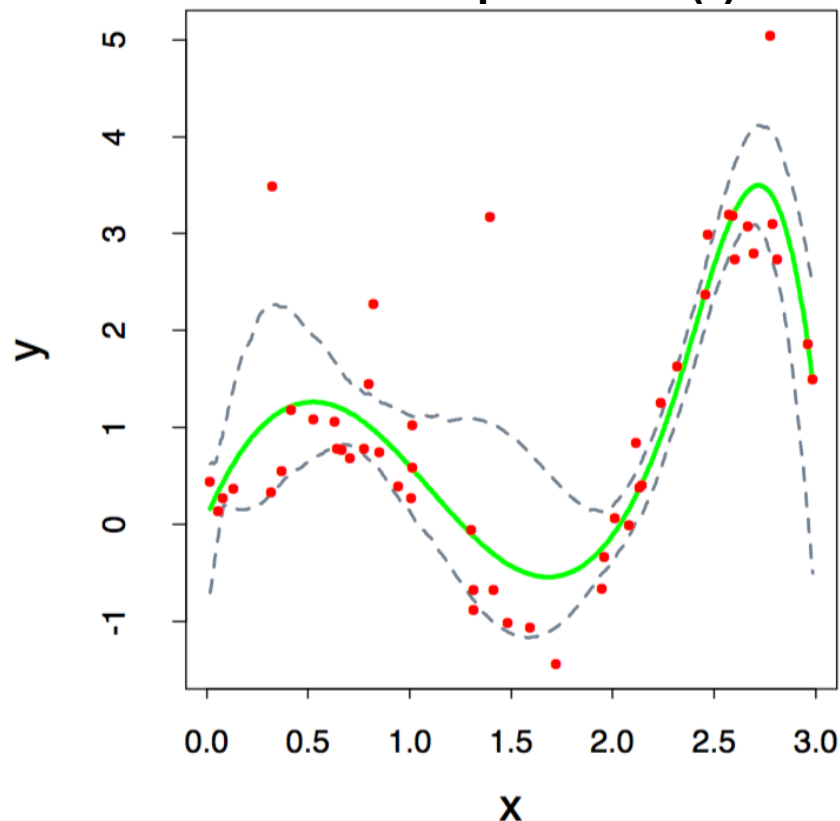
$$f(x) = \sum_{i=1}^n K(x_i, x) \hat{\alpha}_i,$$

where

$$\hat{\alpha} = (\mathbf{P} + \lambda \mathbf{I})^{-1} \mathbf{y} \in \mathbb{R}^n$$

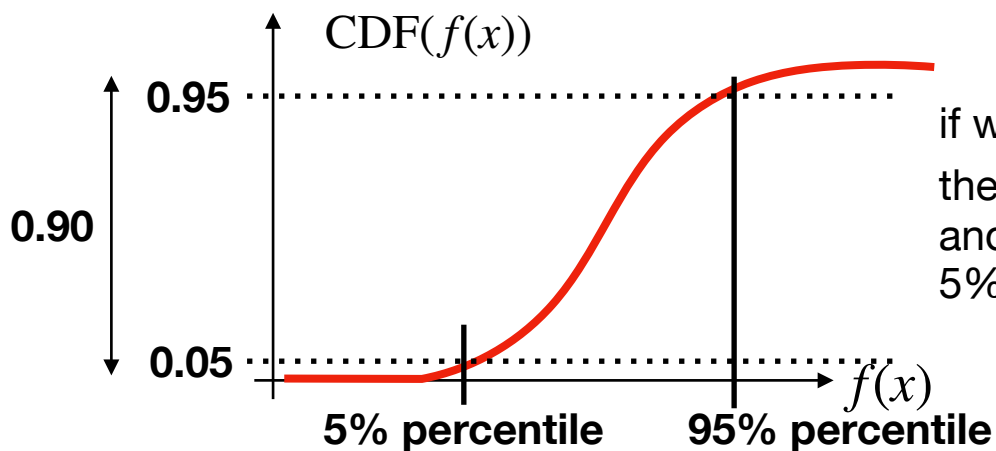
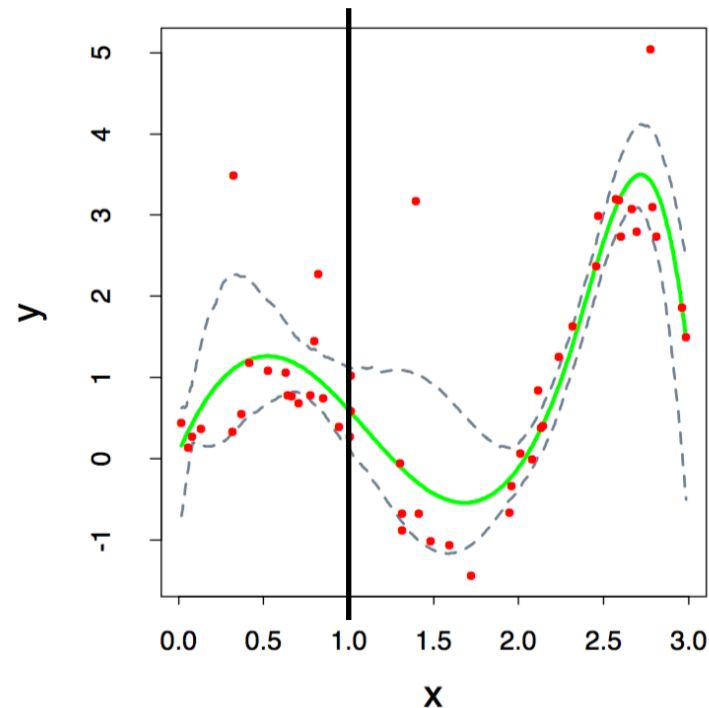
- We wish to build a confidence interval for our predictor $f(x)$, using 5% and 95% percentiles

Example of 5% and 95% percentile curves for predictor $f(x)$



Confidence intervals

- Let's focus on a single $x \in \mathbb{R}^d$
- Note that our predictor $f(x)$ is a random variable, whose randomness comes from the training data $S_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$
- If we know the statistics (in particular the CDF of the random variable $f(x)$) of the predictor, then the **confidence interval** with **confidence level 90%** is defined as



if we know the distribution of our predictor $f(x)$, the green line is the expectation $\mathbb{E}[f(x)]$ and the black dashed lines are the 5% and 95% percentiles in the figure above

- As we do not have the cumulative distribution function (CDF), we need to approximate them

Confidence intervals

- Hypothetically, if we can sample as many times as we want, then we can train $B \in \mathbb{Z}^+$ i.i.d. predictors, each trained on n fresh samples to get **empirical estimate of the CDF of $\hat{y} = f(x)$**

- For $b=1, \dots, B$

- Draw n fresh samples $\{(x_i^{(b)}, y_i^{(b)})\}_{i=1}^n$

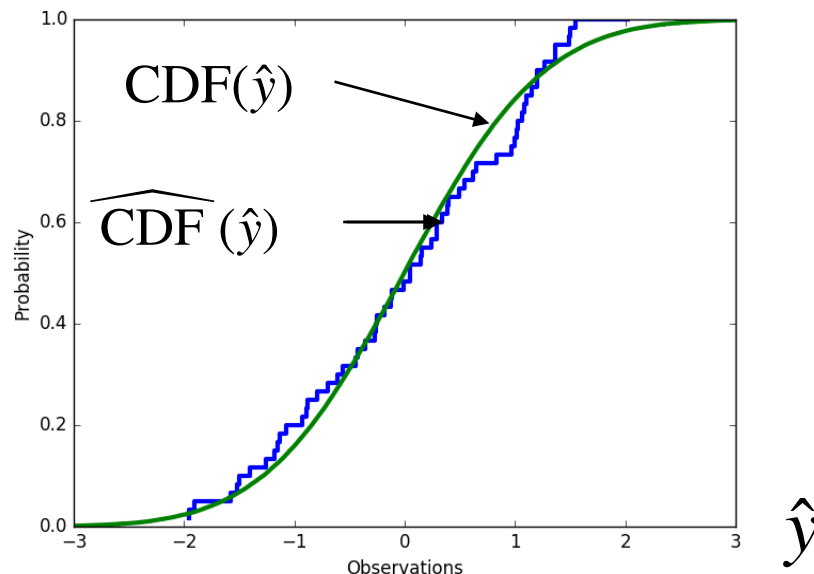
- Train a regularized kernel regression $\alpha^{*(b)}$

- Predict $\hat{y}^{(b)} = \sum_{i=1}^n K(x_i^{(b)}, x) \alpha_i^{*(b)}$

- Let the empirical CDF of those B predictors $\{\hat{y}^{(b)}\}_{b=1}^B$ be $\widehat{\text{CDF}}(\hat{y})$, defined as

$$\widehat{\text{CDF}}(\hat{y}) = \frac{1}{B} \sum_{b=1}^B \mathbf{I}\{\hat{y}^{(b)} \leq \hat{y}\}$$

- Compute the confidence interval using $\widehat{\text{CDF}}(\hat{y})$

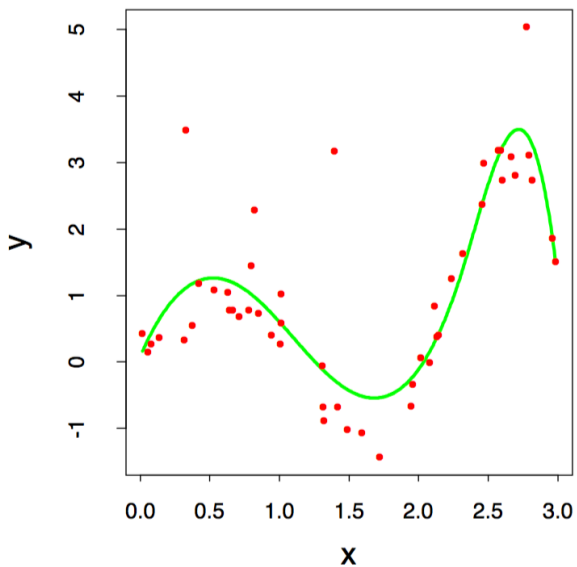


Bootstrap

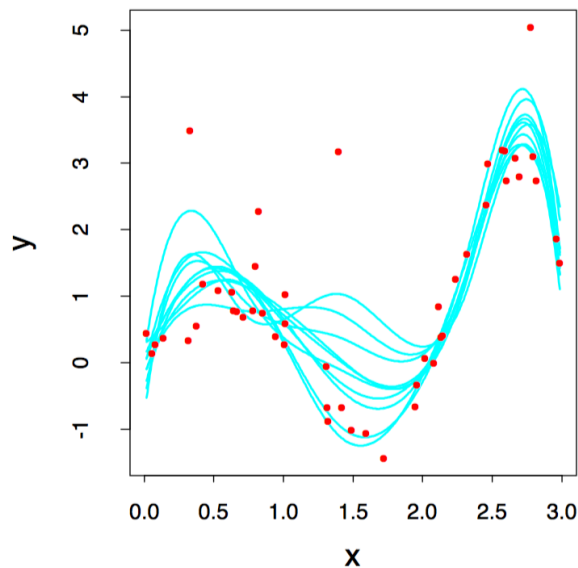
- As we cannot sample repeatedly (in typical cases), we use **bootstrap samples** instead
- Bootstrap is a general tool for assessing statistical accuracy
- We learn it in the context of confidence interval for trained models
- A **bootstrap dataset** is created from the training dataset by taking n (the same size as the training data) examples uniformly at random **with replacement** from the training data $\{(x_i, y_i)\}_{i=1}^n$
- For $b=1, \dots, B$
 - Create a bootstrap dataset $\mathcal{S}_{\text{bootstrap}}^{(b)}$
 - Train a regularized kernel regression $\alpha^{*(b)}$
 - Predict $\hat{y}^{(b)} = \sum_{i=1}^n K(x_i^{(b)}, x) \alpha_i^{*(b)}$
- Compute the empirical CDF from the bootstrap datasets, and compute the confidence interval

bootstrap

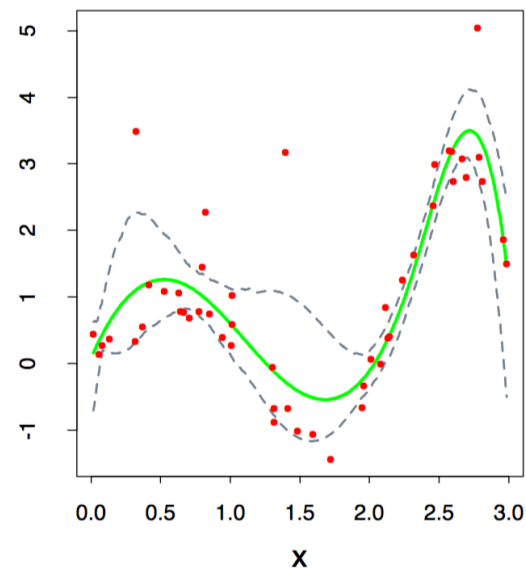
training a single predictor



multiple bootstrapped predictors



90% confidence interval



Figures from Hastie et al