CSCI 567: Machine Learning

Vatsal Sharan
Spring 2024

Lecture 4, February 2

Administrivia

- HW1 due next Wednesday midnight.

Recap

Ensuring generalization

Theorem. *Let* $\mathcal{F}$ *be a function class with size* $|\mathcal{F}|$. *Let* $y = f^*(\boldsymbol{x})$ *for some* $f^* \in \mathcal{F}$. *Suppose we get a training set* $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ *of size* n *with each datapoint drawn i.i.d. from the data distribution D. Let*

$$f_S^{ERM} = \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i), y_i).$$

For any constants $\epsilon, \delta \in (0, 1)$, *if* $n \geq \frac{\ln(|\mathcal{F}|/\delta)}{\epsilon}$, *then with probability* $(1 - \delta)$ *over* $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, $R(f_S^{ERM}) < \epsilon$.

A useful rule of thumb: to guarantee generalization, make sure that your training data set size n is at least linear in the number d of free parameters in the function that you're trying to learn.

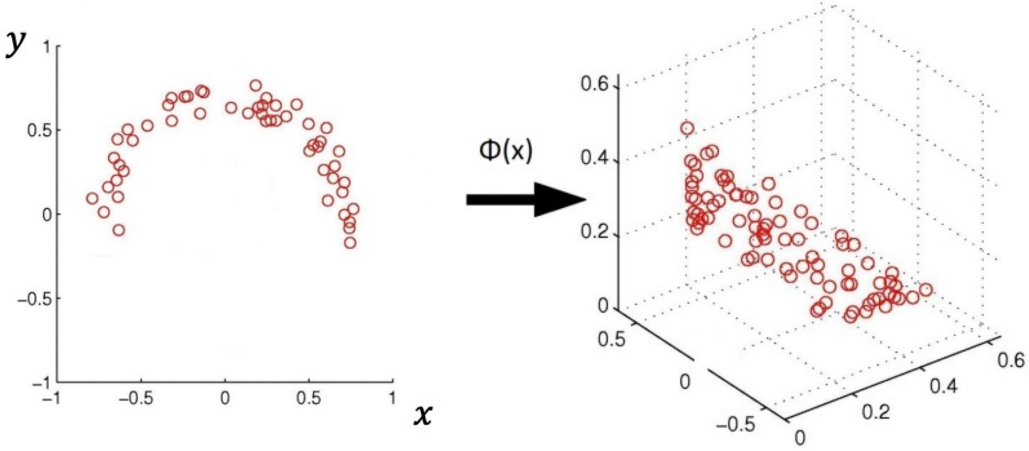
Beyond linear models: nonlinearly transformed features

1. Use a nonlinear mapping

$$\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).



Polynomial basis functions

Polynomial basis functions for $d = 1$

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \quad \Rightarrow \quad f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

Learning a linear model in the new space
$=$ learning an *M-degree polynomial model* in the original space

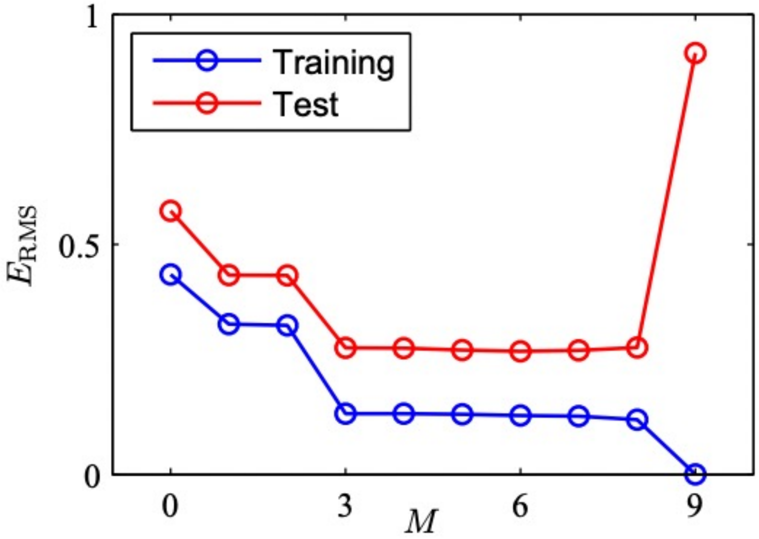
Underfitting and **overfitting**

$M \leq 2$ is *underfitting* the data

- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**



More complicated models ⇒ larger gap between training and test error

How to prevent overfitting?

See Colab notebook

Preventing overfitting: Regularization

Regularized linear regression: new objective

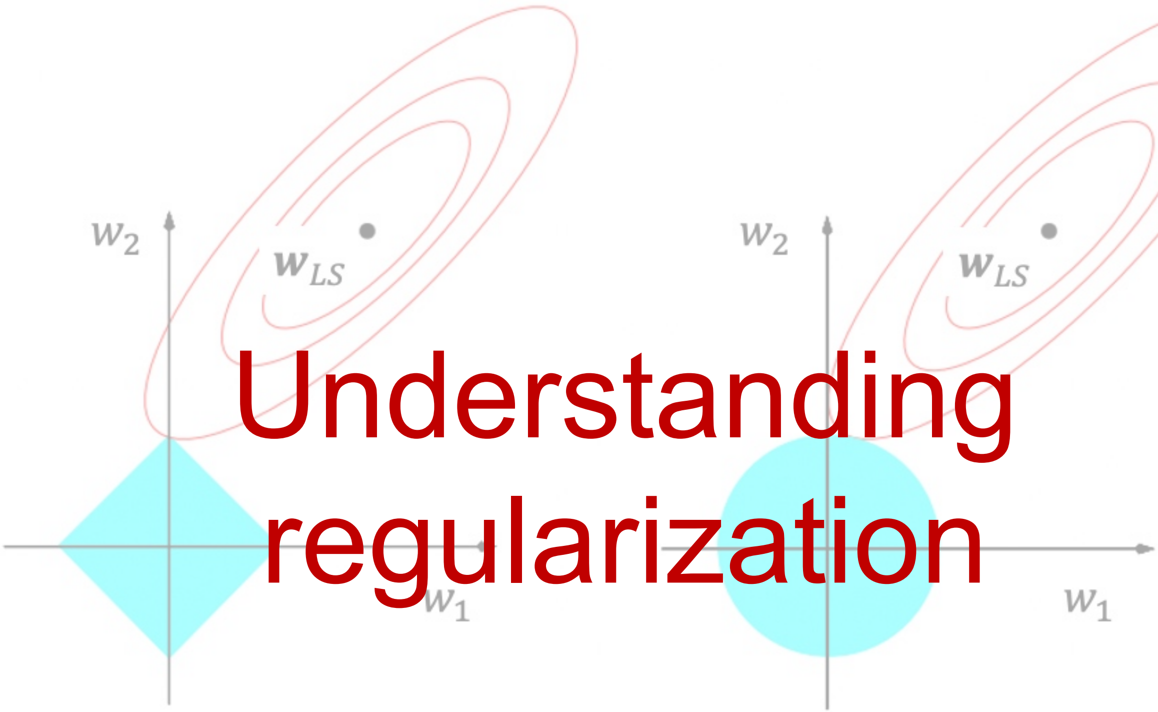
$$G(\boldsymbol{w}) = \text{RSS}(\boldsymbol{w}) + \lambda\psi(\boldsymbol{w})$$

Goal: find $\boldsymbol{w}^* = \text{argmin}_w\, G(\boldsymbol{w})$

- $\psi : \mathbb{R}^d \to \mathbb{R}^+$ is the *regularizer*
  - measure how complex the model $\boldsymbol{w}$ is, penalize complex models
  - common choices: $\|\boldsymbol{w}\|_2^2$, $\|\boldsymbol{w}\|_1$, etc.

- $\lambda > 0$ is the *regularization coefficient*
  - $\lambda = 0$, no regularization
  - $\lambda \to +\infty$, $\boldsymbol{w} \to \text{argmin}_w\, \psi(\boldsymbol{w})$
  - i.e. control **trade-off** between training error and complexity

Understanding regularization



ℓ_2 **regularization**: penalizing large weights

ℓ_2 regularization, $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$:

$$G(\boldsymbol{w}) = \mathrm{RSS}(\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$$

$$\nabla G(\boldsymbol{w}) = 2(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}) + 2\lambda\boldsymbol{w} = 0$$
$$\Rightarrow \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)\boldsymbol{w} = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$
$$\Rightarrow \boldsymbol{w}^* = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda\boldsymbol{I}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

Linear regression with ℓ_2 regularization is also known as **ridge regression**.

With a Bayesian viewpoint, corresponds to a Gaussian prior for $\boldsymbol{w}$.

Encouraging sparsity: ℓ_0 regularization

Continuing from the frequentist view, having small norm is one possible structure to impose on the model. Another very common one is **sparsity**.

Sparsity of w: Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

E.g. $\boldsymbol{w} = [1, 0, -1, 0, 0.2, 0, 0\,]$ is 3-sparse

Encouraging sparsity: ℓ_0 regularization

Sparsity of w: Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.

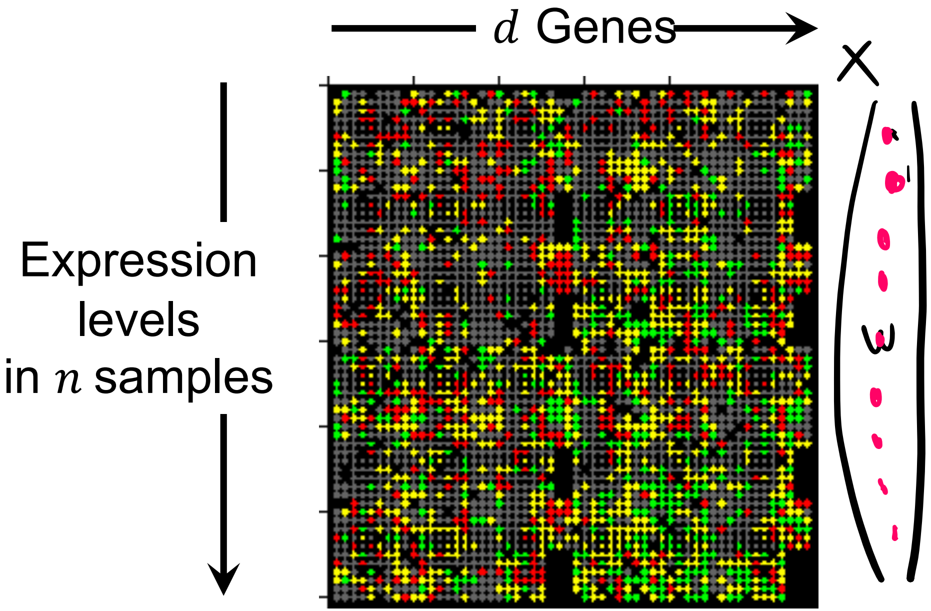
Encouraging sparsity: ℓ_0 regularization

Sparsity of w: Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.

$Xw = y$



d Genes

Expression levels in n samples

Suppose we want to fit a linear models from gene expression to an outcome (disease, phenotype etc.).

d is huge, but likely that only a few genes are related.

Encouraging sparsity: ℓ_0 regularization

Sparsity of w: Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.
- Sparse models may also be more **interpretable**. They could narrow down a small number of features which carry a lot of signal.

E.g. $\boldsymbol{w} = [1.5, 0, -1.1, 0, 0.25, 0, 0\,]$ is more interpretable than,
$\boldsymbol{w} = [1, 0.2, -1.3, 0.15, 0.2, 0.05, 0.12\,]$

For a sparse model, it could be easier to understand the model. It is also easier to verify whether the features which have a high weight have a relation with the outcome (they are not spurious artifacts of the data).

Encouraging sparsity: ℓ_0 regularization

Sparsity of w: Number of non-zero coefficients in $\boldsymbol{w}$. Same as $||\mathbf{w}||_0$

Advantage:

- Sparse models are a natural inductive bias in many settings. In many applications we have numerous possible features, only some of which may have any relationship with the label.
- Sparse models may also be more **interpretable**. They could narrow down a small number of features which carry a lot of signal.
- Data required to learn sparse model maybe significantly less than to learn dense model.

We'll see more on the third point next.

ℓ_0 regularization: The good, the bad and the ugly

Choose $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_0$.

$$G(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2 + \lambda \|\boldsymbol{w}\|_0.$$

Good : Need less data to learn

Suppose weights in w are $\{-1, 0, 1\}$.

How many such s-sparse vectors are there in d dimensions?

Answer: $\binom{d}{s} \cdot 2^s$ possibilities

ℓ_0 regularization: The good, the bad and the ugly

How much data to learn?

$\binom{d}{s} \le d^s$

About $\log(|F|)$ many samples to learn

→ $\log\left(\binom{d}{s} 2^s\right) \le \log(d^s \cdot 2^s)$

$= \log(d^s) + \log(2^s)$

$= s \log d + s \log(2)$

How many free parameters?

→ choose which s of the d coordinates ≈ s parameters

→ choose value ($\{\pm 1\}$) for each coordinate = s parameters

ℓ_0 regularization: The good, the bad and the ugly

In contrast, without s- sparsity need about $\simeq d$ samples to learn (in d dimensions).

$\therefore$ If $s \ll d$, need much less data to generalize !!

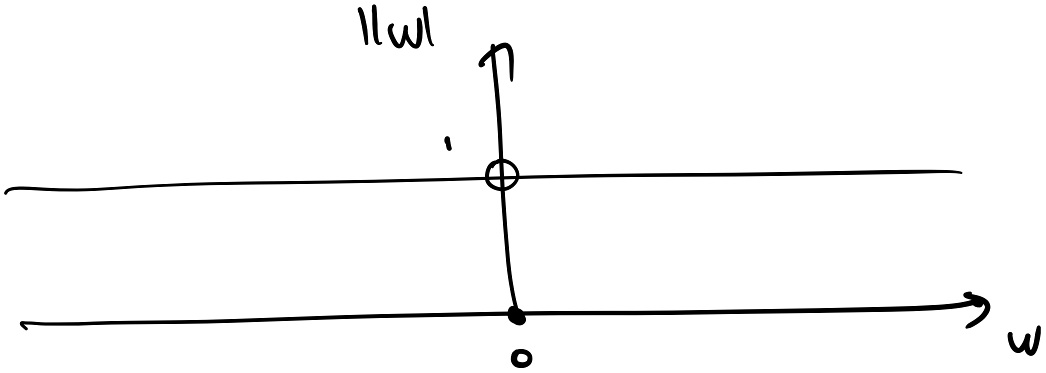
Bad : $\|w\|_0$ is non-convex ($\|w\|_p$, $p<1$ is non-convex).

minimizing $G(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda \|w\|_0$

is NP-Hard :(

ℓ_0 regularization: The good, the bad and the ugly

Ugly: $\|w\|_0$ is highly-discontinuous



GD has no hope !! :(

ℓ_1 regularization as a proxy for ℓ_0 regularization

Choose $\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$. $= \sum_{i=1}^{d} |w_i|$

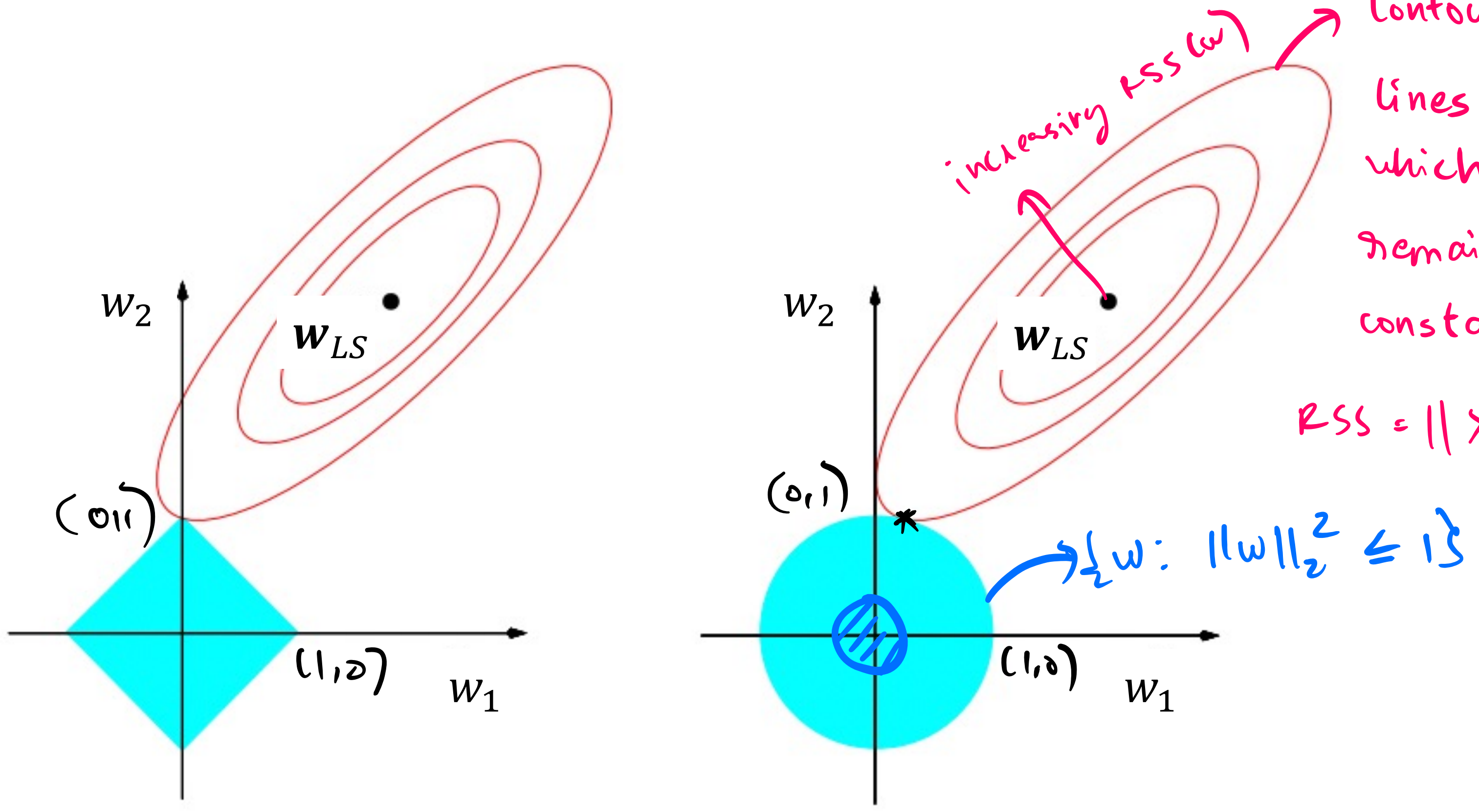
$$G(\boldsymbol{w}) = \sum_{i=1}^{n} (\boldsymbol{w}^T \boldsymbol{x}_i - y_i)^2 + \lambda \|\boldsymbol{w}\|_1.$$

There is theory which says that under some appropriate conditions, doing ℓ_1 regularization has the same effect as if we did ℓ_0 regularization, i.e. we get sparsity, and have the same data requirement as if we did ℓ_0 regularization!

Why does ℓ_1 regularization encourage sparse solutions?

Optimization problem: $\text{argmin}_w \, \text{RSS}(\boldsymbol{w})$, subject to $\psi(\boldsymbol{w}) \leq \beta$

$\beta = 1$



Contour lines

lines along which RSS(w) remains constant

$RSS = \|Xw - y\|_2^2$

$\Psi(w) = \|w\|_1$

$\Psi(w) = \|w\|_2^2$

Adapted from ESL

Diving deeper: ℓ_1 and ℓ_2 regularization for the "isotropic" case

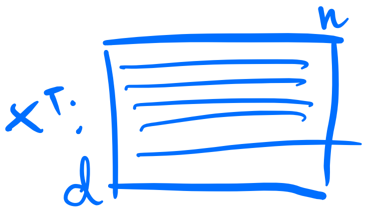
Isotropic assumption : $X^T X = I$

informally,

① all features have mean 0

② all features have variance 1

③ features are uncorrelated.



① $\psi(w) = \|w\|_2^2$

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

$$w^* = (X^T X + \lambda I)^{-1} X^T y$$

Now, $X^T X = I \Rightarrow w^* = ((1+\lambda) I)^{-1} X^T y = \left(\frac{1}{1+\lambda}\right) X^T y$

$$w_j^* = \left(\frac{1}{1+\lambda}\right) X_{(j)}^T y$$

jth row of X^T

jth co-ordinate of $w^* = \left(\frac{1}{1+\lambda}\right)$ · correlation of jth feature with label

Diving deeper: ℓ_1 and ℓ_2 regularization for the "isotropic" case

Without ℓ_2 regularization, with the isotropic assumption ($\boldsymbol{X}^\top \boldsymbol{X} = I$) we had

$$w_j^* = \boldsymbol{X}_{(j)}^\top \boldsymbol{y} = \beta_j$$

where we define $\beta_j = \boldsymbol{X}_{(j)}^\top \boldsymbol{y}$ to be the correlation of j-th feature with label.

With ℓ_2 regularization and the isotropic assumption we get,

$$w_j^* = \left(\frac{1}{1+\lambda}\right) \beta_j.$$

Therefore, ℓ_2 regularization ”shrinks” the estimated parameters.

Note: When features have unequal variance, ℓ_2 regularization applies similar shrinkage to all of them. So, scaling features properly can be important.

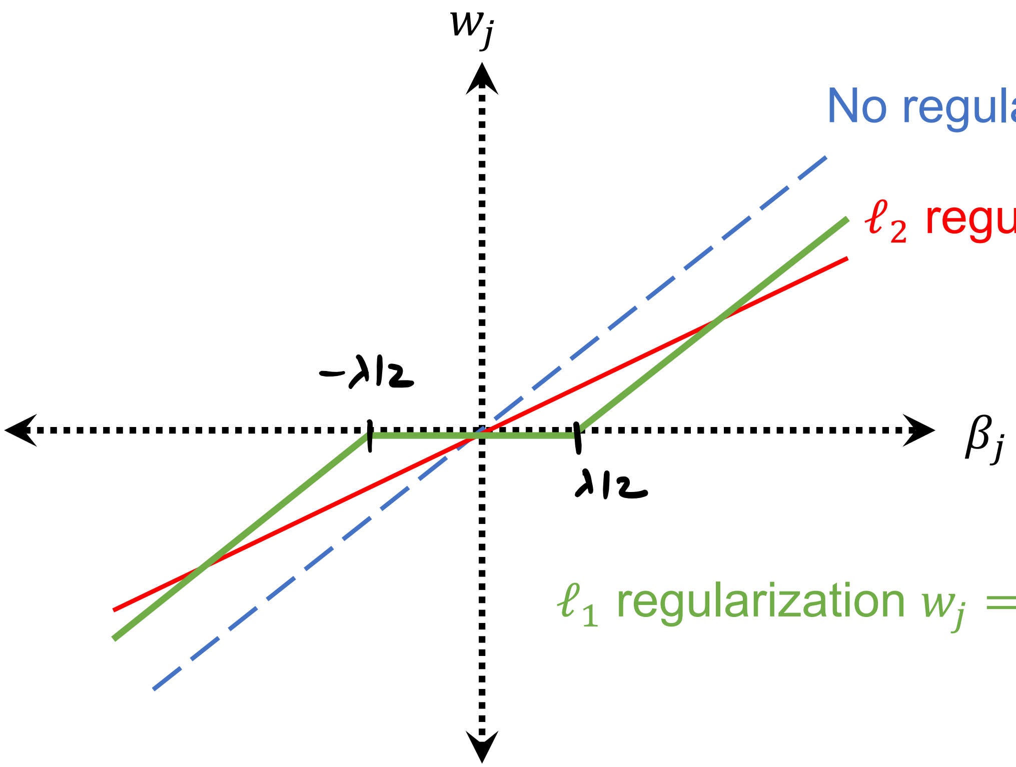
Diving deeper: ℓ_1 and ℓ_2 regularization for the "isotropic" case

What about ℓ_1 regularization ($\psi(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$) ?
Let $\beta_j = \boldsymbol{X}_{(j)}^T \boldsymbol{y}$ as before
It is possible to show that for the ℓ_1 regularized case:

$$w_j = \begin{cases} \beta_j - \lambda/2, \beta_j > \lambda/2 \\ 0, |\beta_j| \leq \lambda/2 \\ \beta_j + \lambda/2, \beta_j < -\lambda/2 \end{cases}$$

Diving deeper: ℓ_1 and ℓ_2 regularization for the "isotropic" case

Summary: Isotropic case ($\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{I}$).
Let $\beta_j = \boldsymbol{X}_{(j)}^T\boldsymbol{y}$



No regularization $w_j = \beta_j$

ℓ_2 regularization $w_j = \beta_j/(1+\lambda)$

ℓ_1 regularization $w_j = \begin{cases} \beta_j - \lambda/2, \beta_j > \lambda/2 \\ 0, |\beta_j| \le \lambda/2 \\ \beta_j + \lambda/2, \beta_j < -\lambda/2 \end{cases}$

Implicit regularization

So far, we explicitly added a $\psi(\boldsymbol{w})$ term to our objective function to regularize.

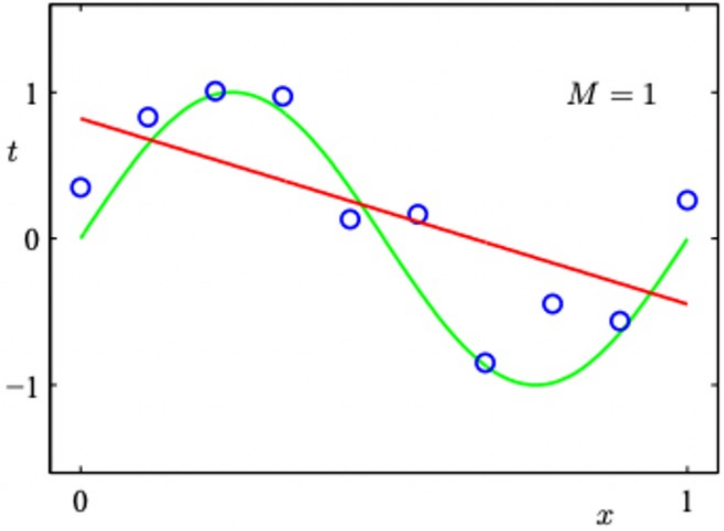
In many cases, the optimization algorithm we use can themselves act as regularizers, favoring some solutions over others.

Currently a very active area of research, you’ll see more in the homework.

Bias-variance tradeoff

The phenomenon of underfitting and overfitting is often referred to as the ***bias-variance tradeoff*** in the literature.

A model whose complexity is too *small* for the task will *underfit*. This is a model with a large bias because the model's accuracy will not improve even if we add a lot of training data.

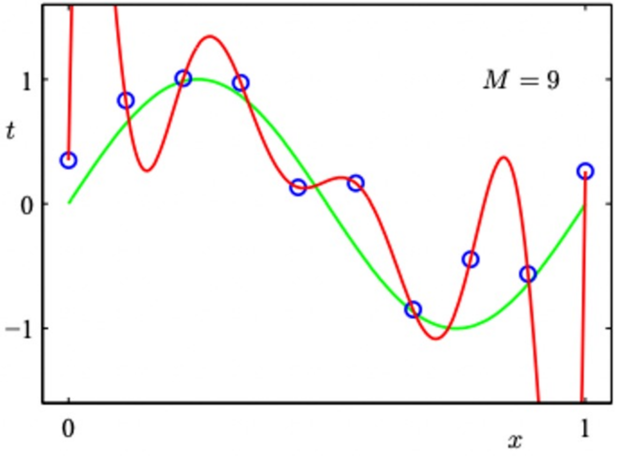


sin(x) fitting example we saw in Lec 3

Bias-variance tradeoff

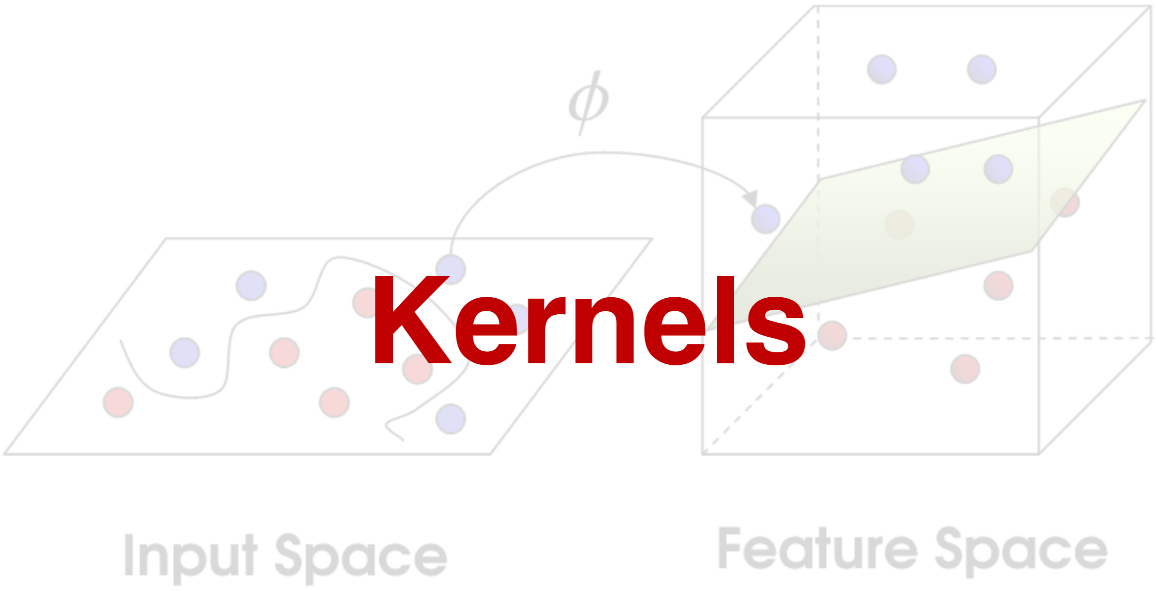
The phenomenon of underfitting and overfitting is often referred to as the ***bias-variance tradeoff*** in the literature.

A model whose complexity is too *large* for the amount of available training data will *overfit*. This is a model with high variance, because the model's predictions will vary a lot with the randomness in the training data (it can even fit any noise in the training data).



sin(x) fitting example we saw in Lec 3

Kernels



Motivation

Recall the nonlinear function map for linear regression:

1. **Use a nonlinear mapping**

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

to transform the data to a more complicated feature space

2. **Then apply linear regression** (hope: linear model is a better fit for the new feature space).

Kernel methods give a way to choose and efficiently work with the nonlinear map $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^M$ (for linear regression, and much more broadly).

Regularized least squares

Let’s continue with regularized least squares with non-linear basis:

$$\begin{aligned}\boldsymbol{w}^* &= \underset{\boldsymbol{w}}{\operatorname{argmin}}\, F(\boldsymbol{w}) \\ &= \underset{\boldsymbol{w}}{\operatorname{argmin}} \left(\|\boldsymbol{\Phi w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2\right) \\ &= \left(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}\end{aligned}$$

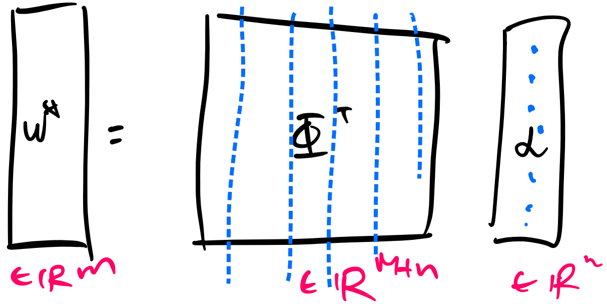
$$\underset{\in \mathbb{R}^{n+M}}{\boldsymbol{\Phi}} = \begin{pmatrix} \phi(\boldsymbol{x}_1)^{\mathrm{T}} \\ \phi(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \phi(\boldsymbol{x}_n)^{\mathrm{T}} \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

This operates in space $\mathbb{R}^M$ and M could be huge (and even infinite).

Regularized least squares solution: Another look

By setting the gradient of $F(\boldsymbol{w}) = \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$ to be $\mathbf{0}$:

$$\underbrace{\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{w}^* - \boldsymbol{y})} + \lambda\boldsymbol{w}^* = \mathbf{0}$$



we know

$$\boldsymbol{w}^* = \frac{1}{\lambda}\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}^*) = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$$

$$\alpha = \frac{(y - \Phi w^*)}{\lambda}$$

Thus the least square solution is **a linear combination of features of the datapoints**!

This calculation does not show what $\boldsymbol{\alpha}$ should be, but ignore that for now.

Why is this helpful?

Assuming we know $\boldsymbol{\alpha}$, the prediction of $\boldsymbol{w}^*$ on a new example $\boldsymbol{x}$ is

$$\boldsymbol{w}^{*\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$$

$$w^* = \sum_{i=1}^{n} \phi(x_i) \cdot \alpha_i$$

$$\Rightarrow w^* \phi(x) = \sum_{i=1}^{n} \alpha_i (\phi(x_i))^T \phi(x)$$

Therefore, *only inner products in the new feature space matter!*

Kernel methods are exactly about computing inner products *without explicitly computing ϕ*.

But we need to figure out what $\boldsymbol{\alpha}$ is first!

Solving for α, Step 1: Kernel matrix

Plugging in $\boldsymbol{w} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ into $F(\boldsymbol{w})$ gives

$$
\begin{aligned}
H(\boldsymbol{\alpha}) &= F(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}) \\
&= \|\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}\|_2^2 \\
&= \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha} \qquad (\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} \in \mathbb{R}^{n\times n})
\end{aligned}
$$

$$
\begin{aligned}
&(\Phi^T\alpha)^T(\Phi^T\alpha) \\
&= \alpha^T \Phi \Phi^T \alpha \\
&= \alpha^T K \alpha
\end{aligned}
$$

$\boldsymbol{K}$ is called **Gram matrix** or **kernel matrix** where the (i, j)-th entry is

$$
\boldsymbol{K}_{(i,j)} = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)
$$

Kernel matrix: Example

$$\phi(x_1) = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \quad \phi(x_2) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad \phi(x_3) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Gram/Kernel matrix

$$\begin{aligned} \boldsymbol{K} &= \begin{pmatrix} \phi(x_1)^{\mathrm{T}}\phi(x_1) & \phi(x_1)^{\mathrm{T}}\phi(x_2) & \phi(x_1)^{\mathrm{T}}\phi(x_3) \\ \phi(x_2)^{\mathrm{T}}\phi(x_1) & \phi(x_2)^{\mathrm{T}}\phi(x_2) & \phi(x_2)^{\mathrm{T}}\phi(x_3) \\ \phi(x_3)^{\mathrm{T}}\phi(x_1) & \phi(x_3)^{\mathrm{T}}\phi(x_2) & \phi(x_3)^{\mathrm{T}}\phi(x_3) \end{pmatrix} \\ &= \begin{pmatrix} 4 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 4 \end{pmatrix} \end{aligned}$$

Kernel matrix vs Covariance matrix

	dimensions	**entry** (i,j)	**property**
$\overset{\mathbb{R}^{n\times M}}{\Phi}\Phi^{T}$	$n \times n$	$\phi(x_i)^T \phi(x_j)$	both are symmetric & positive semi-definite (psd)
$\Phi^{T}\Phi$	$M \times M$	$\sum_{k=1}^{n} \underbrace{\phi(x_k)_i}_{\text{ith co-ordinate of datapoint } \phi(x_k)} \phi(x_k)_j$	

Any matrix $A = UU^T$ (for some matrix U) is psd.

$$x^T A x = x^T U U^T x = (U^T x)^T (U^T x) = \|U^T x\|_2^2 \geq 0 .$$

Solving for α, Step 2: Minimize the dual

Minimize (the so-called *dual formulation*)

$$H(\boldsymbol{\alpha}) = \|\boldsymbol{K}\boldsymbol{\alpha} - \boldsymbol{y}\|_2^2 + \lambda\boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\alpha}$$

Setting the derivative to $\mathbf{0}$ we have

$$\mathbf{0} = (\boldsymbol{K}^2 + \lambda\boldsymbol{K})\boldsymbol{\alpha} - \boldsymbol{K}\boldsymbol{y} = \boldsymbol{K}\left(\underbrace{(\boldsymbol{K} + \lambda\boldsymbol{I})\boldsymbol{\alpha} - \boldsymbol{y}}_{=0}\right)$$

Thus $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$ **is a minimizer** and we obtain

$$\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{y}$$

Exercise: *are there other minimizers? and are there other $\boldsymbol{w}^*$'s?*

Comparing two solutions

Minimizing $F(\boldsymbol{w})$ **gives** $\boldsymbol{w}^* = (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y}$

Minimizing $H(\boldsymbol{\alpha})$ **gives** $\boldsymbol{w}^* = \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$

Note $\boldsymbol{I}$ has different dimensions in these two formulas.

Natural question: *are the two solutions the same or different?*

They have to be the same because $F(\boldsymbol{w})$ has a unique minimizer!

And they are:

$$
\begin{aligned}
&(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{y} \\
&= (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y} \\
&= (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{\Phi}^{\mathrm{T}})(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y} \\
&= (\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1}(\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi} + \lambda \boldsymbol{I})\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y} \\
&= \boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}
\end{aligned}
$$

The kernel trick

If the solutions are the same, then what is the difference?

takes $O(M^3)$ time

First, computing $(\overbrace{\mathbf{\Phi}\mathbf{\Phi}^{\mathrm{T}}}^{n+n \text{ dim. } \therefore \text{ takes } O(n3)} + \lambda \boldsymbol{I})^{-1}$ can be more efficient than computing $(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \lambda \boldsymbol{I})^{-1}$ when $n \leq M$.

$M+M$ dimensions

More importantly, computing $\boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1}\boldsymbol{y}$ also *only requires computing inner products in the new feature space!*

Now we can conclude that the exact form of $\phi(\cdot)$ is not essential; *all we need to do is know the inner products $\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}')$.*

For some ϕ it is indeed possible to compute $\phi(\boldsymbol{x})^{\mathrm{T}}\phi(\boldsymbol{x}')$ without computing/knowing ϕ. This is the *kernel trick*.

The kernel trick: Example 1

Consider the following polynomial basis $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^3$:

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \qquad x' = \begin{pmatrix} x_1' \\ x_2' \end{pmatrix}$$

What is the inner product between $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\boldsymbol{x}')$?

$$\begin{aligned} \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}') &= {x_1}^2{x_1'}^2 + 2x_1x_2x_1'x_2' + {x_2}^2{x_2'}^2 \\ &= (x_1x_1' + x_2x_2')^2 = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}')^2 \end{aligned}$$

Therefore, *the inner product in the new space is simply a function of the inner product in the original space*.

The kernel trick: Example 2

$\boldsymbol{\phi} : \mathbb{R}^d \to \mathbb{R}^{2d}$ is parameterized by θ:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \qquad \boldsymbol{\phi}_\theta(\boldsymbol{x}) = \begin{pmatrix} \cos(\theta x_1) \\ \sin(\theta x_1) \\ \vdots \\ \cos(\theta x_d) \\ \sin(\theta x_d) \end{pmatrix}$$

What is the inner product between $\boldsymbol{\phi}_\theta(\boldsymbol{x})$ and $\boldsymbol{\phi}_\theta(\boldsymbol{x}')$?

$$\boldsymbol{\phi}_\theta(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}_\theta(\boldsymbol{x}') = \sum_{m=1}^{d} \cos(\theta x_m)\cos(\theta x'_m) + \sin(\theta x_m)\sin(\theta x'_m)$$

$$= \sum_{m=1}^{d} \cos(\theta(x_m - x'_m)) \qquad \text{(trigonometric identity)}$$

Once again, *the inner product in the new space is a simple function of the features in the original space.*

The kernel trick: Example 3

Based on ϕ_θ, define $\phi_L : \mathbb{R}^d \to \mathbb{R}^{2d(L+1)}$ for some integer L:

$$\phi_L(\boldsymbol{x}) = \begin{pmatrix} \phi_0(\boldsymbol{x}) \\ \phi_{\frac{2\pi}{L}}(\boldsymbol{x}) \\ \phi_{2\frac{2\pi}{L}}(\boldsymbol{x}) \\ \vdots \\ \phi_{L\frac{2\pi}{L}}(\boldsymbol{x}) \end{pmatrix}$$

What is the inner product between $\phi_L(\boldsymbol{x})$ and $\phi_L(\boldsymbol{x}')$?

$$\begin{aligned} \phi_L(\boldsymbol{x})^{\mathrm{T}}\phi_L(\boldsymbol{x}') &= \sum_{\ell=0}^{L} \phi_{\frac{2\pi\ell}{L}}(\boldsymbol{x})^{\mathrm{T}}\phi_{\frac{2\pi\ell}{L}}(\boldsymbol{x}') \\ &= \sum_{\ell=0}^{L}\sum_{m=1}^{d} \cos\left(\frac{2\pi\ell}{L}(x_m - x'_m)\right) \end{aligned}$$

The kernel trick: Example 4

When $L \to \infty$, even if we cannot compute $\phi(x)$ (since it's a vector of *infinite dimension*), we can still compute inner product:

swap $\int$ & Σ

$$\boldsymbol{\phi}_\infty(\boldsymbol{x})^{\mathrm{T}}\boldsymbol{\phi}_\infty(\boldsymbol{x}') = \int_0^{2\pi} \sum_{m=1}^{d} \cos(\theta(x_m - x'_m))\, d\theta$$

$$\left(\frac{2\pi \ell}{L} = \theta \right)$$

$$= \sum_{m=1}^{d} \frac{\sin(2\pi(x_m - x'_m))}{(x_m - x'_m)}$$

$$\int_0^{2\pi} \cos(x\theta)\, d\theta = \frac{\sin x\theta}{x}\Big|_0^{2\pi} = \frac{\sin 2\pi x}{x}$$

Again, a simple function of the original features.

Note that when using this mapping in linear regression, we are *learning a weight $\boldsymbol{w}^*$ with infinite dimension!*

Kernel functions

Definition: a function $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is called a *kernel function* if there exists a function $\boldsymbol{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^M$ so that for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}')$$

Examples we have seen

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\mathrm{T}} \boldsymbol{x}')^2$$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{m=1}^{d} \frac{\sin(2\pi(x_m - x'_m))}{(x_m - x'_m)}$$

Using kernel functions

Choosing a nonlinear basis ϕ becomes equivalent to choosing a kernel function.

As long as computing the kernel function is more efficient, we should apply the kernel trick.

Gram/kernel matrix becomes:

$$\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_1, \boldsymbol{x}_n) \\ k(\boldsymbol{x}_2, \boldsymbol{x}_1) & k(\boldsymbol{x}_2, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_2, \boldsymbol{x}_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_1) & k(\boldsymbol{x}_n, \boldsymbol{x}_2) & \cdots & k(\boldsymbol{x}_n, \boldsymbol{x}_n) \end{pmatrix}$$

In fact, k is a kernel if and only if $\boldsymbol{K}$ is positive semidefinite for *any n and any $\boldsymbol{x}_1$, $\boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$* (**Mercer theorem**).

- useful for proving that a function is not a kernel

Examples which are not kernels

Function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2^2$$

is *not a kernel*, why?

If it is a kernel, the kernel matrix for two data points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$:

this entry is $\|x_1 - x_1\|_2^2 = 0$

$$\boldsymbol{K} = \begin{pmatrix} 0 & \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 \\ \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2^2 & 0 \end{pmatrix}$$

must be positive semidefinite, *but is it?*

suppose $\|x_1 - x_2\| = 1 \Rightarrow K = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, which is not psd.

why? take $(1 \quad -1)$. $(1 \quad -1)\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \end{pmatrix} = (-1 \quad 1)\begin{pmatrix} 1 \\ -1 \end{pmatrix} = -2$

Properties of kernels

For any function $f : \mathbb{R}^d \to \mathbb{R}$, $k(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})f(\boldsymbol{x}')$ is a kernel. → What is ϕ?

$\phi : \mathbb{R}^d \to \mathbb{R}$,

$\phi(x) = f(x)$

If $k_1(\cdot, \cdot)$ and $k_2(\cdot, \cdot)$ are kernels, then the following are also kernels:

- **conical combination**: $\alpha k_1(\cdot, \cdot) + \beta k_2(\cdot, \cdot)$ if $\alpha, \beta \geq 0$ → What is ϕ?
- **product**: $k_1(\cdot, \cdot)k_2(\cdot, \cdot)$
- **exponential**: $e^{k(\cdot, \cdot)}$
- $\cdots$

ϕ_1 : map for k_1

ϕ_2 : map for k_2

ϕ' : map for $\alpha k_1 + \beta k_2$

Verify using the definition of kernel!

Exercise : what is ϕ'?

Popular kernels

Polynomial kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}' + c)^M$$

for $c \geq 0$ and M is a positive integer.

What is the corresponding ϕ?

$c = 0$, $M = 2$, we saw earlier $\phi(x) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$

Popular kernels

Gaussian kernel or Radial basis function (RBF) kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0.$$

What is the corresponding ϕ?

$$(\|x - x'\|_2^2 = \|x\|_2^2 + \|x'\|_2^2 - 2x^T x'$$

$$k(x, x') = \underbrace{\exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right) \exp\left(-\frac{\|x'\|_2^2}{2\sigma^2}\right)}_{} \exp\left(\frac{x^T x'}{\sigma^2}\right)$$

$$k(x, x') = f(x) f(x')$$

For $f(x) = \exp\left(-\frac{\|x\|_2^2}{2\sigma^2}\right)$

Popular kernels

Gaussian kernel or Radial basis function (RBF) kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right) \quad \text{for some } \sigma > 0.$$

What is the corresponding ϕ?

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

$$\exp\left(\frac{x^T x'}{\sigma^2}\right) = 1 + \frac{x^T x'}{\sigma^2} + \underbrace{\frac{1}{2!}\frac{(x^T x')^2}{(\sigma^2)^2}} + \frac{1}{3!}\frac{(x^T x')^3}{(\sigma^2)^3} + \dots$$

each of these is polynomial kernel

Popular kernels

Appropriate kernels have also been developed for tasks like Natural Language Processing where inputs are discrete.

For two strings s_1 and s_2 and some parameter t,

$$k_t(s_1, s_2) = \text{Number of sub-strings of length } t \text{ which appear in both } s_1 \text{ and } s_2.$$

For e.g. if $t = 1$,

$$k_t(\text{'machine','learning'}) = 4.$$

What is the corresponding ϕ? Exercise !

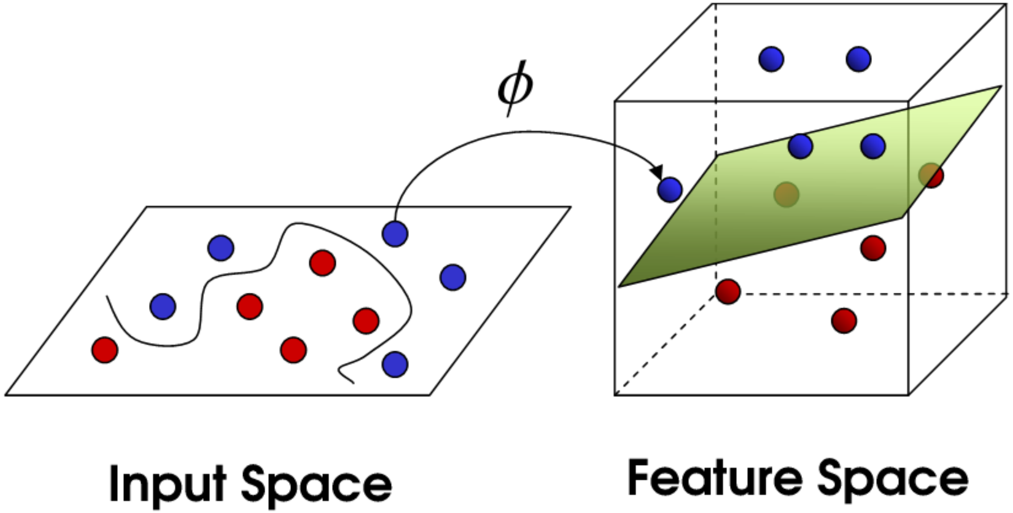
Prediction with kernels

As long as $\boldsymbol{w}^* = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$, prediction on a new example $\boldsymbol{x}$ becomes

$$\boldsymbol{w}^{*\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^n \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}).$$

This is known as a **non-parametric method**. Informally speaking, this means that there is no fixed set of parameters that the model is trying to learn (remember $\boldsymbol{w}^*$ could be infinite). Nearest-neighbors is another non-parametric method we have seen.

Classification with kernels



Similar ideas extend to the classification case, and we can predict using $\mathrm{sign}(\boldsymbol{w}^T\boldsymbol{\phi})$. Data may become linearly separable in the feature space!