Kernel Machines

Ramon Fuentes[1,2], Artur Gower[3]

August 8, 2019

[1]Visiting Researcher, Dynamics Research Group
The University of Sheffield

[2]Research Scientist, Callsign Ltd

[3]Lecturer in Dynamics
The University of Sheffield

Recap

- Ordinary Linear Regression
- Expansions into polynomial and other bases
- Bias and variance in models
- Regularisation as a method of balancing model complexity

Recap

- We are generally looking to solve $\mathbf{y} = \mathbf{X}\mathbf{w}$
- OLS: $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$
- Ridge Regression: $\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

Feature spaces

- Linear regression model nonlinear problems through the use of an expansion:

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w}$$

- For instance, a quadratic expansion would be defined as,

$$\begin{aligned}\mathbf{\Phi} = [1, \mathbf{x}_1,\ \mathbf{x}_2,\ \mathbf{x}_3,\ ...,\ \mathbf{x}_d, \mathbf{x}_1^2, \mathbf{x}_1\mathbf{x}_2, \mathbf{x}_1\mathbf{x}_3, ..., \mathbf{x}_1\mathbf{x}_d,\\ \mathbf{x}_2\mathbf{x}_1, \mathbf{x}_2^2, \mathbf{x}_2 x_3, ..., \mathbf{x}_2\mathbf{x}_d, \mathbf{x}_3\mathbf{x}_1, \mathbf{x}_3\mathbf{x}_2, \mathbf{x}_3^2, ...,\\ \mathbf{x}_3\mathbf{x}_d, \ldots, \mathbf{x}_d\mathbf{x}_1, \mathbf{x}_d\mathbf{x}_2, \mathbf{x}_d\mathbf{x}_3, ..., \mathbf{x}_d^2]\end{aligned}$$

Expansions

- The solution with such an expansion can be simply formulated as,
$$\mathbf{w} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{y},$$

Expansions

- The solution with such an expansion can be simply formulated as,

$$\mathbf{w} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{y},$$

  where $\mathbf{\Phi}$ is a matrix with d^p columns and n rows:

$$\mathbf{\Phi} = \begin{bmatrix} \phi(\mathbf{x}_1) \\ \vdots \\ \phi(\mathbf{x}_n) \end{bmatrix}.$$

Expansions

- expansions, such as the polynomials can be very expressive - we can model complex problems with them
- but lets think for a moment about how many columns there are in a degree p polynomial...
- $\sim d^p$!
- imagine you had a data set with 10 variables (d) and required fitting a polynomial with $p = 5$, how many features is that ?

Expansions

- The term $\mathbf{\Phi}^T\mathbf{\Phi}$ yields a $d^p \times d^p$ matrix, which we need to invert
- Usually we need roughly as many training samples as we have dimensions (!)
- Defining the expansions **explicitly** is
    - computationally intractable
    - and leads to numerically unstable matrix inversions
- Where on earth are we going to collect d^p training samples? ... that's a lot of time in the lab!

but there is hope...

what if...

- If only there was a way to learn and make predictions using large number of features **without** actually having to compute them?
- It turns out, there is!
- Using kernels

Dual form ridge regression

- There are two forms of linear regression: primal and dual
- So far we've learned about the primal, so lets have a look at this other equivalent version

The dual form

Let's use the feature map $\mathbf{\Phi}$ with d dimensions so that:

The dual form

Let's use the feature map $\boldsymbol{\Phi}$ with d dimensions so that:

$$y = \boldsymbol{\Phi}\mathbf{w} \;\text{ where }\; \mathbf{w} = \mathbf{A}^{-1}\boldsymbol{\Phi}^T\mathbf{y}, \;\; \mathbf{A} = \boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}_{d^p}.$$

The dual form

Let's use the feature map $\mathbf{\Phi}$ with d dimensions so that:

$$y = \mathbf{\Phi w} \;\text{ where }\; \mathbf{w} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y}, \;\; \mathbf{A} = \mathbf{\Phi}^T\mathbf{\Phi} + \lambda \mathbf{I}_{d^p}.$$

For a large number of features d^p, and limited samples n, we can avoid inverting the $d^p \times d^p$ matrix $\mathbf{A}$ by using some linear algebra!

The dual form

Let's use the feature map $\mathbf{\Phi}$ with d dimensions so that:

$$y = \mathbf{\Phi w} \ \text{ where } \ \mathbf{w} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y}, \ \ \mathbf{A} = \mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I}_{d^p}.$$

For a large number of features d^p, and limited samples n, we can avoid inverting the $d^p \times d^p$ matrix $\mathbf{A}$ by using some linear algebra!

$$\mathbf{A\Phi}^T = \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n) \implies \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \mathbf{A}^{-1}\mathbf{\Phi}^T$$

The dual form

Let's use the feature map $\mathbf{\Phi}$ with d dimensions so that:

$$y = \mathbf{\Phi w} \text{ where } \mathbf{w} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y}, \;\; \mathbf{A} = \mathbf{\Phi}^T\mathbf{\Phi} + \lambda \mathbf{I}_{d^p}.$$

For a large number of features d^p, and limited samples n, we can avoid inverting the $d^p \times d^p$ matrix $\mathbf{A}$ by using some linear algebra!

$$\mathbf{A\Phi}^T = \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n) \implies \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \mathbf{A}^{-1}\mathbf{\Phi}^T$$

Which means that

$$\begin{aligned} \mathbf{w} &= \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y} && \text{(primal form)} \\ \mathbf{w} &= \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \mathbf{\Phi}^T\mathbf{K}^{-1}\mathbf{y}, && \text{(dual form)} \end{aligned}$$

where $\mathbf{K} = \mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n$.

The dual form

Let's use the feature map $\boldsymbol{\Phi}$ with d dimensions so that:

$$y = \boldsymbol{\Phi}\mathbf{w} \text{ where } \mathbf{w} = \mathbf{A}^{-1}\boldsymbol{\Phi}^T\mathbf{y}, \;\; \mathbf{A} = \boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}_{d^p}.$$

For a large number of features d^p, and limited samples n, we can avoid inverting the $d^p \times d^p$ matrix $\mathbf{A}$ by using some linear algebra!

$$\mathbf{A}\boldsymbol{\Phi}^T = \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I}_n) \implies \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \mathbf{A}^{-1}\boldsymbol{\Phi}^T$$

Which means that

$$\begin{aligned} \mathbf{w} &= \mathbf{A}^{-1}\boldsymbol{\Phi}^T\mathbf{y} && \text{(primal form)} \\ \mathbf{w} &= \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \boldsymbol{\Phi}^T\mathbf{K}^{-1}\mathbf{y}, && \text{(dual form)} \end{aligned}$$

where $\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\mathbf{I}_n$.

Amazing! $\mathbf{K}$ is a $n \times n$ matrix, and $\mathbf{A}$ is a $d^p \times d^p$ matrix.

The dual form

Let's use the feature map $\mathbf{\Phi}$ with d dimensions so that:

$$y = \mathbf{\Phi w} \text{ where } \mathbf{w} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y}, \ \ \mathbf{A} = \mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I}_{d^p}.$$

For a large number of features d^p, and limited samples n, we can avoid inverting the $d^p \times d^p$ matrix $\mathbf{A}$ by using some linear algebra!

$$\mathbf{A\Phi}^T = \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n) \implies \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \mathbf{A}^{-1}\mathbf{\Phi}^T$$

Which means that

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{\Phi}^T\mathbf{y} \qquad \text{(primal form)}$$
$$\mathbf{w} = \mathbf{\Phi}^T(\mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n)^{-1} = \mathbf{\Phi}^T\mathbf{K}^{-1}\mathbf{y}, \quad \text{(dual form)}$$

where $\mathbf{K} = \mathbf{\Phi\Phi}^T + \lambda\mathbf{I}_n$.

Amazing! $\mathbf{K}$ is a $n \times n$ matrix, and $\mathbf{A}$ is a $d^p \times d^p$ matrix.

We can choose to calculate $\mathbf{K}^{-1}$ or $\mathbf{A}^{-1}$.

A digression on notation...

At the this point, it is important to introduce notation to distinguish between training and prediction samples:

Training samples: $\mathbf{x}, \mathbf{y}$

Prediction points: $\mathbf{x}^*, \mathbf{y}^*$

Dual Ridge Regression, predictive equations

In dual form we have that,

$$\mathbf{w} = \mathbf{\Phi}^T \mathbf{K}^{-1} \mathbf{y}$$

so the predictive model is,

$$\mathbf{y}^* = \mathbf{\Phi}^* \mathbf{\Phi}^T \mathbf{K}^{-1} \mathbf{y}$$

where,

$$\mathbf{K} = (\mathbf{\Phi}\mathbf{\Phi}^T + \lambda I_n)$$

Dual form ridge regression

- Why have we gone through all this trouble?

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$
- However, note:

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$
- However, note:
    - In primal, we invert a $d^p \times d^p$ matrix

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$
- However, note:
  - In primal, we invert a $d^p \times d^p$ matrix
  - In dual, we invert an $n \times n$ matrix

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$
- However, note:
  - In primal, we invert a $d^p \times d^p$ matrix
  - In dual, we invert an $n \times n$ matrix
  - So which is better?

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$
- However, note:
  - In primal, we invert a $d^p \times d^p$ matrix
  - In dual, we invert an $n \times n$ matrix
  - So which is better?
  - Dual is better when $d^p \gg n$, and this is the case in large feature expansions

Dual form ridge regression

- Why have we gone through all this trouble?
- Both primal and dual achieve the same end: predict $\mathbf{y}^*$ given $\mathbf{X}$ and $\mathbf{x}^*$
- However, note:
  - In primal, we invert a $d^p \times d^p$ matrix
  - In dual, we invert an $n \times n$ matrix
  - So which is better?
  - Dual is better when $d^p \gg n$, and this is the case in large feature expansions
- If you had a 100000-dimensional space from a 5^{th}-order polynomial, but only 10 samples, you could solve for it by only using those 10 samples, and inverting a 10×10 matrix !!!

Dual form regression

- So, we have gone from:
  - having to compute a d^p-dimensional feature space and solving a severely under-determined $d^p \times d^p$ system

Dual form regression

- So, we have gone from:
    - having to compute a d^p-dimensional feature space and solving a severely under-determined $d^p \times d^p$ system
    - solving such system by only inverting a 10×10 matrix, provided we can compute $\mathbf{\Phi}\mathbf{\Phi}^T$

Dual form regression

- So, we have gone from:
    - having to compute a d^p-dimensional feature space and solving a severely under-determined $d^p \times d^p$ system
    - solving such system by only inverting a 10×10 matrix, provided we can compute $\mathbf{\Phi}\mathbf{\Phi}^T$
- that is great!
- but what if we didn't even have to compute $\mathbf{\Phi}\mathbf{\Phi}^T$?

The kernel trick

- Have you noticed how both the dual and primal (our first approach) depend on inner products? For example, $\mathbf{\Phi}\mathbf{\Phi}^T$, $\mathbf{\Phi}^T\mathbf{\Phi}$ and $\mathbf{\Phi}\mathbf{\Phi}^{T*}$) ?

The kernel trick

- Have you noticed how both the dual and primal (our first approach) depend on inner products? For example, $\mathbf{\Phi}\mathbf{\Phi}^T$, $\mathbf{\Phi}^T\mathbf{\Phi}$ and $\mathbf{\Phi}\mathbf{\Phi}^{T*}$) ?
- It turns out there are easier ways to evaluate these inner products in our feature space.

The kernel trick

- Have you noticed how both the dual and primal (our first approach) depend on inner products? For example, $\mathbf{\Phi}\mathbf{\Phi}^T$, $\mathbf{\Phi}^T\mathbf{\Phi}$ and $\mathbf{\Phi}\mathbf{\Phi}^{T*}$) ?
- It turns out there are easier ways to evaluate these inner products in our feature space.
- To evaluate these inner products we use a kernel function $\kappa(\mathbf{x}, \mathbf{x}')$

The kernel trick

- Have you noticed how both the dual and primal (our first approach) depend on inner products? For example, $\mathbf{\Phi}\mathbf{\Phi}^T$, $\mathbf{\Phi}^T\mathbf{\Phi}$ and $\mathbf{\Phi}\mathbf{\Phi}^{T*}$) ?
- It turns out there are easier ways to evaluate these inner products in our feature space.
- To evaluate these inner products we use a kernel function $\kappa(\mathbf{x}, \mathbf{x}')$
- A linear kernel function gives us:

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}' = \sum_j x_j x_j'$$

The kernel trick

Using the dual to predict y^* from $\mathbf{x}^*$ and $\mathbf{X}$ we need to calculate

The kernel trick

Using the dual to predict y^* from $\mathbf{x}^*$ and $\mathbf{X}$ we need to calculate

$$y^* = \phi(\mathbf{x}^*)\mathbf{w} = \phi(\mathbf{x}^*)\mathbf{\Phi}^T\mathbf{K}^{-1}\mathbf{y} = \\ \begin{bmatrix}\phi(\mathbf{x}^*)\cdot\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}^*)\cdot\phi(\mathbf{x}_n)\end{bmatrix}\mathbf{K}^{-1}\mathbf{y},$$

The kernel trick

Using the dual to predict y^* from $\mathbf{x}^*$ and $\mathbf{X}$ we need to calculate

$$y^* = \phi(\mathbf{x}^*)\mathbf{w} = \phi(\mathbf{x}^*)\mathbf{\Phi}^T\mathbf{K}^{-1}\mathbf{y} = \begin{bmatrix}\phi(\mathbf{x}^*)\cdot\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}^*)\cdot\phi(\mathbf{x}_n)\end{bmatrix}\mathbf{K}^{-1}\mathbf{y},$$

where $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I}_n$ whose components are

$$\mathbf{K} = \begin{bmatrix}\phi(x_1)\cdot\phi(x_1) & \phi(x_1)\cdot\phi(x_2) & \dots & \phi(x_1)\cdot\phi(x_n) \\ \phi(x_2)\cdot\phi(x_1) & \phi(x_2)\cdot\phi(x_2) & \dots & \phi(x_2)\cdot\phi(x_n) \\ \vdots & \vdots & & \\ \phi(x_n)\cdot\phi(x_1) & \phi(x_2)\cdot\phi(x_n) & \dots & \phi(x_n)\cdot\phi(x_n)\end{bmatrix} + \lambda\mathbf{I}_n$$

The kernel trick

Using the dual to predict y^* from $\mathbf{x}^*$ and $\mathbf{X}$ we need to calculate

$$y^* = \phi(\mathbf{x}^*)\mathbf{w} = \phi(\mathbf{x}^*)\mathbf{\Phi}^T\mathbf{K}^{-1}\mathbf{y} = \\ \begin{bmatrix}\phi(\mathbf{x}^*)\cdot\phi(\mathbf{x}_1) & \dots & \phi(\mathbf{x}^*)\cdot\phi(\mathbf{x}_n)\end{bmatrix}\mathbf{K}^{-1}\mathbf{y},$$

where $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T + \lambda\mathbf{I}_n$ whose components are

$$\mathbf{K} = \begin{bmatrix} \phi(x_1)\cdot\phi(x_1) & \phi(x_1)\cdot\phi(x_2) & \dots & \phi(x_1)\cdot\phi(x_n) \\ \phi(x_2)\cdot\phi(x_1) & \phi(x_2)\cdot\phi(x_2) & \dots & \phi(x_2)\cdot\phi(x_n) \\ \vdots & \vdots & & \\ \phi(x_n)\cdot\phi(x_1) & \phi(x_2)\cdot\phi(x_n) & \dots & \phi(x_n)\cdot\phi(x_n) \end{bmatrix} + \lambda\mathbf{I}_n$$

So we need only calculate $\kappa(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})\cdot\phi(\mathbf{x}')$ many times!

The kernel trick

- Now let's use a polynomial kernel

$$\begin{aligned} \kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x} \cdot \mathbf{x}' + 1)^p \\ &= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}') \end{aligned}$$

The kernel trick

- Now let's use a polynomial kernel

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x} \cdot \mathbf{x}' + 1)^{p} \\ &= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')\end{aligned}$$

  then, $\kappa(\mathbf{x}, \mathbf{x}')$ would contain every monomial in $\mathbf{x}$ of degree $0, ..., p$.

The kernel trick

- Now let's use a polynomial kernel

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x} \cdot \mathbf{x}' + 1)^p \\ &= \phi(\mathbf{x}) \cdot \phi(\mathbf{x}')\end{aligned}$$

  then, $\kappa(\mathbf{x}, \mathbf{x}')$ would contain every monomial in $\mathbf{x}$ of degree $0, ..., p$.
  Which is easier to calculate: $(\mathbf{x} \cdot \mathbf{x}' + 1)^p$ or both $\phi(\mathbf{x})$ and $\phi(\mathbf{x}')$?

The kernel trick: example

- We have now defined the regression problem in terms of a kernel function

The kernel trick: example

- We have now defined the regression problem in terms of a kernel function
- To compute the kernel matrix, with the polynomial kernel, we don't need to evaluate $\mathbf{\Phi}$ at all!

The kernel trick: example

- We have now defined the regression problem in terms of a kernel function
- To compute the kernel matrix, with the polynomial kernel, we don't need to evaluate $\mathbf{\Phi}$ at all!
- Instead, we evaluate the function $(\mathbf{x} \cdot \mathbf{x}' + 1)^p$ for every pair of training samples $\mathbf{x}$ and $\mathbf{x}'$

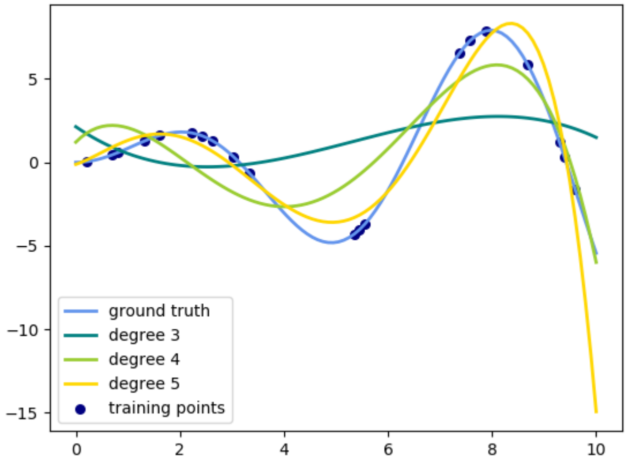
The kernel trick: example

- We have now defined the regression problem in terms of a kernel function
- To compute the kernel matrix, with the polynomial kernel, we don't need to evaluate $\mathbf{\Phi}$ at all!
- Instead, we evaluate the function $(\mathbf{x} \cdot \mathbf{x}' + 1)^p$ for every pair of training samples $\mathbf{x}$ and $\mathbf{x}'$
- We can now do polynomial regression with an exponentially long, high-order polynomial in less time than it would take even to compute $\mathbf{\Phi}$

The kernel trick: example

- We have now defined the regression problem in terms of a kernel function
- To compute the kernel matrix, with the polynomial kernel, we don't need to evaluate $\mathbf{\Phi}$ at all!
- Instead, we evaluate the function $(\mathbf{x} \cdot \mathbf{x}' + 1)^p$ for every pair of training samples $\mathbf{x}$ and $\mathbf{x}'$
- We can now do polynomial regression with an exponentially long, high-order polynomial in less time than it would take even to compute $\mathbf{\Phi}$
- This is MIND BLOWING!

Kernel Ridge regression example



Kernel trick

Lets recap on what we've done so far

- We started by defining linear regression in terms of long feature expansions $\mathbf{\Phi}$

Kernel trick

Lets recap on what we've done so far

- We started by defining linear regression in terms of long feature expansions $\mathbf{\Phi}$
- We converted the linear regression problem into dual form

Kernel trick

Lets recap on what we've done so far

- We started by defining linear regression in terms of long feature expansions $\mathbf{\Phi}$
- We converted the linear regression problem into dual form
- This gave us a solution in terms of the explicit inner product $\mathbf{\Phi}\mathbf{\Phi}^T$

Kernel trick

Lets recap on what we've done so far

- We started by defining linear regression in terms of long feature expansions $\mathbf{\Phi}$
- We converted the linear regression problem into dual form
- This gave us a solution in terms of the explicit inner product $\mathbf{\Phi}\mathbf{\Phi}^T$
- We've then replaced the explicit evaluation of the inner product an implicit evaluation in the **feature space** defined by the kernel function $\kappa(\mathbf{x}, \mathbf{x}')$

Kernel trick

Lets recap on what we've done so far

- We started by defining linear regression in terms of long feature expansions $\mathbf{\Phi}$
- We converted the linear regression problem into dual form
- This gave us a solution in terms of the explicit inner product $\mathbf{\Phi}\mathbf{\Phi}^T$
- We've then replaced the explicit evaluation of the inner product an implicit evaluation in the **feature space** defined by the kernel function $\kappa(\mathbf{x}, \mathbf{x}')$
- Which means we can do nonlinear regression in any feature space defined by κ, without having to actually compute it!

Kernel trick

Lets recap on what we've done so far

- We started by defining linear regression in terms of long feature expansions $\mathbf{\Phi}$
- We converted the linear regression problem into dual form
- This gave us a solution in terms of the explicit inner product $\mathbf{\Phi}\mathbf{\Phi}^T$
- We've then replaced the explicit evaluation of the inner product an implicit evaluation in the **feature space** defined by the kernel function $\kappa(\mathbf{x}, \mathbf{x}')$
- Which means we can do nonlinear regression in any feature space defined by κ, without having to actually compute it!
- This is known as the **kernel trick**

The Gaussian Kernel

- The polynomial kernel allows us to do fast computation in spaces of exponentially increasing dimensions.

The Gaussian Kernel

- The polynomial kernel allows us to do fast computation in spaces of exponentially increasing dimensions.
- Here's something even more awesome...
- We can go all the way and compute features of **infinitely large** dimensional spaces...

The Gaussian Kernel

- The polynomial kernel allows us to do fast computation in spaces of exponentially increasing dimensions.
- Here's something even more awesome...
- We can go all the way and compute features of **infinitely large** dimensional spaces...
- Enter the Gaussian kernel function,

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp\Big(-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2\sigma^2}\Big)$$

The Gaussian kernel

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp\Big(-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2\sigma^2}\Big)$$

The Gaussian kernel

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp\Big(-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2\sigma^2}\Big)$$

This innocent-looking expression actually comes from this feature vector (for one dimension only),

The Gaussian kernel

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp\Big(-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2\sigma^2}\Big)$$

This innocent-looking expression actually comes from this feature vector (for one dimension only),

$$\phi(x_1) = \exp\Big(-\frac{x_1^2}{2\sigma^2}\Big)\Big[1, \frac{x_1}{\sigma\sqrt{1!}}, \frac{x_1^2}{\sigma^2\sqrt{2!}}, \frac{x_1^3}{\sigma^3\sqrt{3!}}, ...,\Big]^T$$

The Gaussian kernel

$$\kappa(\mathbf{x}_1, \mathbf{x}_2) = \exp\Big(-\frac{|\mathbf{x}_1 - \mathbf{x}_2|^2}{2\sigma^2}\Big)$$

This innocent-looking expression actually comes from this feature vector (for one dimension only),

$$\phi(x_1) = \exp\Big(-\frac{x_1^2}{2\sigma^2}\Big)\Big[1, \frac{x_1}{\sigma\sqrt{1!}}, \frac{x_1^2}{\sigma^2\sqrt{2!}}, \frac{x_1^3}{\sigma^3\sqrt{3!}}, ...,\Big]^T$$

which is an infinite vector but still $\phi(x_1) \cdot \phi(x_2)$ converges to $\kappa(\mathbf{x}_1, \mathbf{x}_2)$

The Gaussian kernel

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

- This is really powerful, as it gives us a numerically tractable way of using an infinite-dimensional feature space.

The Gaussian kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\Big(-\frac{|\mathbf{x} - \mathbf{x}'|^2}{2\sigma^2}\Big)$$

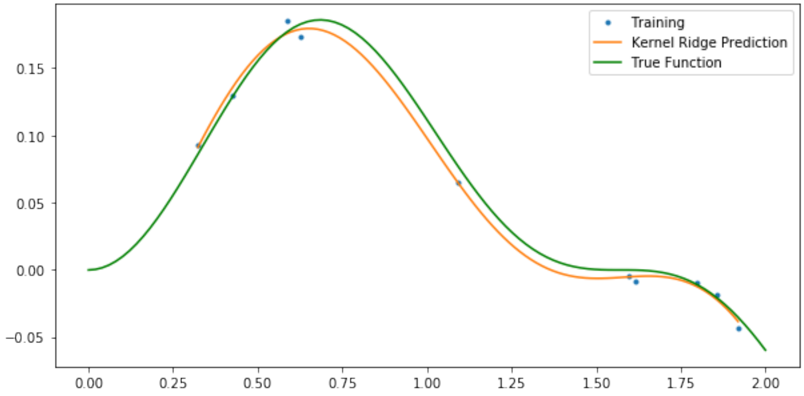
- This is really powerful, as it gives us a numerically tractable way of using an infinite-dimensional feature space.
- At this point, it helps to think of kernels simply as measures of similarity and closeness between pairs of samples.

The Gaussian kernel

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\Big(- \frac{|\mathbf{x} - \mathbf{x}'|^2}{2\sigma^2} \Big)$$

- This is really powerful, as it gives us a numerically tractable way of using an infinite-dimensional feature space.
- At this point, it helps to think of kernels simply as measures of similarity and closeness between pairs of samples.
- (actually a large chunk of kernel methods were developed to deal with spatial statistical modelling of forest density...)

Kernel Ridge Regression example - Gaussian kernel



Some intuition on kernels

We can predict complex functions on large dimension using,

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

Some intuition on kernels

We can predict complex functions on large dimension using,

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

But why does it work?

Some intuition on kernels

We can predict complex functions on large dimension using,

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

But why does it work? Take one training point $[x_1, y_1] = [6, 10]$, then $\mathbf{K} = [\kappa(x_1, x_1)] = [1]$ and $\mathbf{y} = [y_1]$ (training data),

Some intuition on kernels

We can predict complex functions on large dimension using,

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

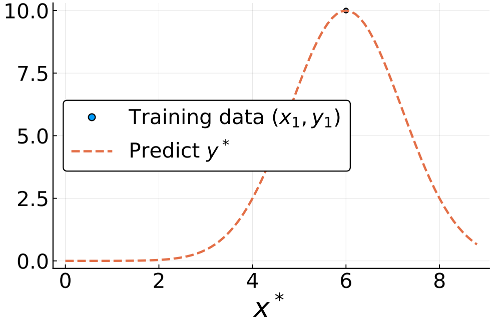
But why does it work? Take one training point $[x_1, y_1] = [6, 10]$, then $\mathbf{K} = [\kappa(x_1, x_1)] = [1]$ and $\mathbf{y} = [y_1]$ (training data), $\mathbf{K}^* = [\kappa(x^*, x_1)]$ and $y^* \approx \kappa(x^*, x_1)y_1 = y_1 \mathrm{e}^{-\frac{(x^*-x_1)^2}{2\sigma^2}}$.

Some intuition on kernels

We can predict complex functions on large dimension using,

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

But why does it work? Take one training point $[x_1, y_1] = [6, 10]$, then $\mathbf{K} = [\kappa(x_1, x_1)] = [1]$ and $\mathbf{y} = [y_1]$ (training data), $\mathbf{K}^* = [\kappa(x^*, x_1)]$ and $y^* \approx \kappa(x^*, x_1)y_1 = y_1 \mathrm{e}^{-\frac{(x^* - x_1)^2}{2\sigma^2}}$.



Some intuition on kernels

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

What about adding more training data?

Some intuition on kernels

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

What about adding more training data? With three data points: $\mathbf{y} = [y_1, y_2, y_3]^T$, $(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} = [a_1, a_2, a_3]^T$ (only training data),

Some intuition on kernels

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

What about adding more training data? With three data points:
$\mathbf{y} = [y_1, y_2, y_3]^T$, $(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} = [a_1, a_2, a_3]^T$ (only training data),
$\mathbf{K}^* = [\kappa(x^*, x_1), \kappa(x^*, x_2), \kappa(x^*, x_3)]$,

Some intuition on kernels

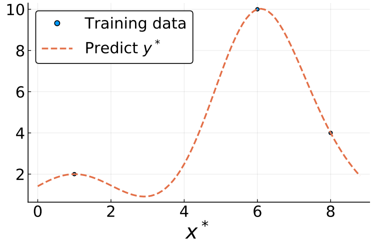
$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

What about adding more training data? With three data points:
$\mathbf{y} = [y_1, y_2, y_3]^T$, $(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} = [a_1, a_2, a_3]^T$ (only training data),
$\mathbf{K}^* = [\kappa(x^*, x_1), \kappa(x^*, x_2), \kappa(x^*, x_3)]$, then
$y^* = \mathbf{K}^*[a_1, a_2, a_3]^T = a_1\kappa(x^*, x_1) + a_2\kappa(x^*, x_2) + a_3\kappa(x^*, x_3)$:

Some intuition on kernels

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

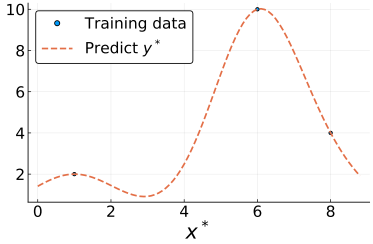
What about adding more training data? With three data points:
$\mathbf{y} = [y_1, y_2, y_3]^T$, $(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} = [a_1, a_2, a_3]^T$ (only training data),
$\mathbf{K}^* = [\kappa(x^*, x_1), \kappa(x^*, x_2), \kappa(x^*, x_3)]$, then
$y^* = \mathbf{K}^*[a_1, a_2, a_3]^T = a_1\kappa(x^*, x_1) + a_2\kappa(x^*, x_2) + a_3\kappa(x^*, x_3)$:

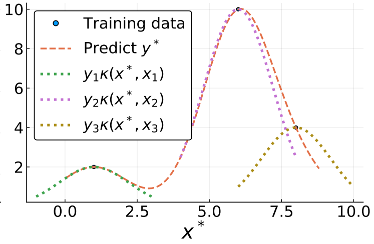


Some intuition on kernels

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right).$$

What about adding more training data? With three data points:
$\mathbf{y} = [y_1, y_2, y_3]^T$, $(\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{y} = [a_1, a_2, a_3]^T$ (only training data),
$\mathbf{K}^* = [\kappa(x^*, x_1), \kappa(x^*, x_2), \kappa(x^*, x_3)]$, then
$y^* = \mathbf{K}^*[a_1, a_2, a_3]^T = a_1\kappa(x^*, x_1) + a_2\kappa(x^*, x_2) + a_3\kappa(x^*, x_3)$:

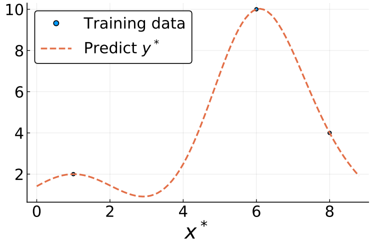


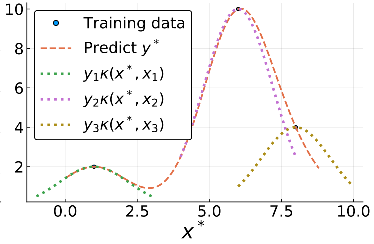


Some intuition on kernels

$$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y}, \quad \text{and} \quad \kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right).$$

What about adding more training data? With three data points:
$\mathbf{y} = [y_1, y_2, y_3]^T$, $(\mathbf{K} + \lambda \mathbf{I})^{-1}\mathbf{y} = [a_1, a_2, a_3]^T$ (only training data),
$\mathbf{K}^* = [\kappa(x^*, x_1), \kappa(x^*, x_2), \kappa(x^*, x_3)]$, then
$y^* = \mathbf{K}^*[a_1, a_2, a_3]^T = a_1\kappa(x^*, x_1) + a_2\kappa(x^*, x_2) + a_3\kappa(x^*, x_3)$:





If add $\sum_j y_j\kappa(x^*, x_j)$ we don't get y^*. The $a_j \neq y_j$ to compensate for the overlapping kernel functions $\kappa(x^*, x_j)$.

What have we learned today?

- learned about the dual form of linear regression
- introduced the kernel trick
- Shown that you can learn and predict fairly complex functions on large dimensions using,

  $$\mathbf{y}^* = \mathbf{K}^*(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

  where

  $$\mathbf{K} = \kappa(\mathbf{x}, \mathbf{x}')$$

  (all pairs of training points), and

  $$\mathbf{K}^* = \kappa(\mathbf{x}^*, \mathbf{x})$$

  (pairs of training and prediction points)
- This is called **Kernel Ridge Regression**