


Kernel Methods

Instructor: Hui Guan

Slides adapted from: https://github.com/jermwatt/machine_learning_refined

Outline

- Fixed-Shape Universal Approximators 
- The Kernel Trick
- Kernels as Similarity Measures
- Optimization of Kernelized Models
- Cross-Validating Kernelized Learners

12.2 Fixed-shape Universal Approximators

$$f(x) = x^d \quad \text{model} = x + x^2 + \dots + x^d$$

- What generally characterizes fixed-shaped universal approximators are:
 - **their lack of internal parameters**, and
 - **straightforward organization** of units in terms of, e.g., degree
- These characteristics make fixed-shape approximators, like the polynomials extremely popular in many areas adjacent to machine learning such as applied mathematics, physics, and engineering.

Large input dimensions and the scaling challenge

With N dimensional input a polynomial unit takes the form

$$f_m(\mathbf{x}) = \underline{x_1^{m_1} x_2^{m_2} \cdots x_N^{m_N}}$$

To construct a polynomial model of degree \$D\$ we collect all such terms where

$$m_1 + m_2 + \cdots + m_N \leq D$$

$$\boxed{C_{N+D}^{D+N-1}}$$

$$\underline{m_1 \cdots + m_N} = D$$

$$m_1 \cdots + m_N + k_1 \cdots + k_D$$

Q: what is the possible number of combinations of $m_1 \dots m_N$?

- It is easy to verify that the precise number M of (non-bias) features/feature weights of a degree D polynomial is.

$$\binom{N+D}{D} - 1 = \frac{(N+D)!}{N!D!} - 1$$

- Even if the input dimension is of reasonably small size, for instance $N=100$ or $N=500$, then just the associated degree $D=5$ polynomial feature map of these input dimensions has dimension $M=96,560,645$ and $M=268,318,178,226$ respectively!
- This serious scaling issue motivates the so-called ***kernel trick***.

12.3 The Kernel Trick

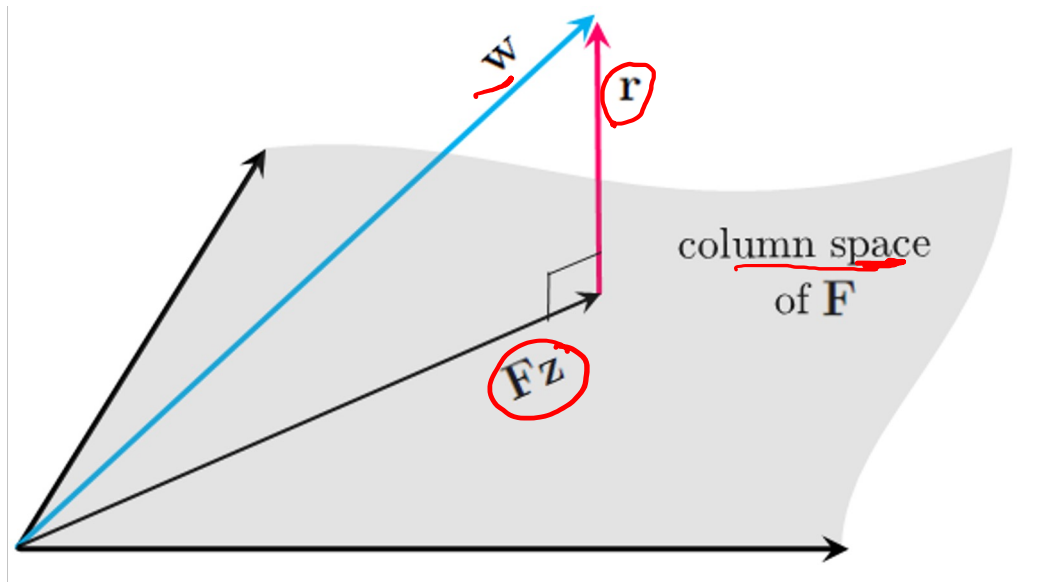
- This crucial issue, of not being able to effectively store and compute with high dimensional fixed-shape feature transformations, motivates notion of **kernelization**.
 - allows us to avoid this scaling problem,
 - provides a way of generating new fixed-shape features defined solely through such a kernelized representation.

A useful fact from the fundamental theorem of linear algebra

Any vector ω in an M-dimensional space can be decomposed over the column space of a given matrix \mathbf{F} , as

$$\underline{\omega} = \underline{\mathbf{F}}\underline{\mathbf{z}} + \underline{\mathbf{r}}. \quad \mathbf{F} = \begin{bmatrix} f_1(x_1) & f_1(x_p) \\ f_2(x_1) & \vdots \\ f_n(x_1) & f_n(x_p) \end{bmatrix}$$

The vector \mathbf{Fz} belongs in the subspace determined by the columns of \mathbf{F} while \mathbf{r} is orthogonal to this subspace.



Kernelizing machine learning cost functions

Example: Kernelizing regression via the Least Squares cost

Suppose we want to perform a generic nonlinear regression using B units and the corresponding model evaluated at the p^{th} input \mathbf{x}_p

$$\text{model}(\mathbf{x}_p, \mathbf{w}) = w_0 + f_1(\mathbf{x}_p) w_1 + f_2(\mathbf{x}_p) w_2 + \cdots + \underline{f_B(\mathbf{x}_p)} w_B.$$

$$B \sim C_D^{N+D}$$

$$B \sim N! \quad \times$$

Using the compact notation

$$b = w_0 \quad \underline{\omega} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \quad \underline{\mathbf{f}}_p = \begin{bmatrix} f_1(\mathbf{x}_p) \\ f_2(\mathbf{x}_p) \\ \vdots \\ f_B(\mathbf{x}_p) \end{bmatrix}$$

we can write our model more compactly - exposing the features apart from our bias - as

$$\text{model}(\mathbf{x}_p, b, \omega) = b + \underline{\mathbf{f}}_p^T \omega$$

In this notation our Least Squares cost for regression takes the form

$$g(b, \boldsymbol{\omega}) = \sum_{p=1}^P (b + \mathbf{f}_p^T \boldsymbol{\omega} - y_p)^2.$$

Denote by \mathbf{F} the $B \times P$ matrix \mathbf{F} formed by stacking the vectors \mathbf{f}_p column-wise. Now, employing the fundamental theorem of linear algebra discussed above we may write \mathbf{w} here as

$$\underline{\omega} = \mathbf{Fz} + \mathbf{r}$$

where \mathbf{r} satisfies $\mathbf{F}^T \mathbf{r} = \mathbf{0}_{P \times 1}$

$$B \sim N!$$

F

$$W \in \mathbb{R}^B$$

Plugging this representation of \mathbf{w} back into the cost function then gives

$$(b + \mathbf{f}_p^T \mathbf{w} - y_p)^2$$

$$\sum_{p=1}^P (b + \mathbf{f}_p^T (\mathbf{F}\mathbf{z} + \mathbf{r}) - y_p)^2 = \sum_{p=1}^P (b + \mathbf{f}_p^T \mathbf{F}\mathbf{z} - y_p)^2$$

$$\sum_{p=1}^P (b + \mathbf{h}^T \mathbf{z} - y_p)^2$$

$$\mathbf{f}_p^T \mathbf{r} = 0$$

$$\mathbf{h} \in \mathbb{R}^P \quad \mathbf{h}^T = \mathbf{f}_p^T \mathbf{F} \in \mathbb{R}^P$$

$$\mathbf{f}_p \in \mathbb{R}^B$$

$$\mathbf{F} \in \mathbb{R}^{B \times P}$$

$$x_p \quad b + \mathbf{f}_p^T \mathbf{w}$$

$$\in \mathbb{R}^B$$

$$B \sim N!$$

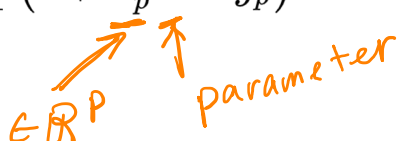
$$b + \mathbf{h}^T \mathbf{z}$$

$$\in \mathbb{R}^P$$

$$P$$

Finally, denoting the symmetric $P \times P$ kernel matrix $\mathbf{H} = \mathbf{F}^T \mathbf{F}$ (and where $\mathbf{h}_p = \mathbf{F}^T \mathbf{f}_p$ is the p^{th} column of this matrix), referred to as a *kernel matrix* or just the *kernel*, our original cost function becomes equivalently

$$g(b, \mathbf{z}) = \sum_{p=1}^P (b + \mathbf{h}_p^T \mathbf{z} - y_p)^2$$


 $\in \mathbb{R}^P$ parameter

Our corresponding model evaluated at the p^{th} input now takes the form

$$\text{model}(\underline{\mathbf{x}}_p, b, \underline{\mathbf{z}}) = b + \underline{\mathbf{h}}_p^T \mathbf{z}.$$

Note that we have changed the arguments of the cost function from $g(b, \mathbf{w})$ to $g(b, \mathbf{z})$ due to our substitution of \mathbf{w} .

Using the same sort of argument, we may kernelize many machine learning problems, summarized for convenience in the table below.

Table 12.1 Popular supervised learning cost functions and their kernelized versions.

Cost function	Original version	Kernelized version
Least Squares	$\frac{1}{P} \sum_{p=1}^P (b + \mathbf{f}_p^T \boldsymbol{\omega} - y_p)^2$	$\frac{1}{P} \sum_{p=1}^P (b + \mathbf{h}_p^T \mathbf{z} - y_p)^2$
Two-class Softmax	$\frac{1}{P} \sum_{p=1}^P \log(1 + e^{-y_p(b + \mathbf{f}_p^T \boldsymbol{\omega})})$	$\frac{1}{P} \sum_{p=1}^P \log(1 + e^{-y_p(b + \mathbf{h}_p^T \mathbf{z})})$
Squared-margin SVM	$\frac{1}{P} \sum_{p=1}^P \max^2(0, 1 - y_p(b + \mathbf{f}_p^T \boldsymbol{\omega}))$	$\frac{1}{P} \sum_{p=1}^P \max^2(0, 1 - y_p(b + \mathbf{h}_p^T \mathbf{z}))$
Multi-class Softmax	$\frac{1}{P} \sum_{p=1}^P \log \left(1 + \sum_{\substack{j=0 \\ j \neq y_p}}^{C-1} e^{(b_j - b_{y_p}) + \mathbf{f}_p^T (\boldsymbol{\omega}_j - \boldsymbol{\omega}_{y_p})} \right)$	$\frac{1}{P} \sum_{p=1}^P \log \left(1 + \sum_{\substack{j=0 \\ j \neq y_p}}^{C-1} e^{(b_j - b_{y_p}) + \mathbf{h}_p^T (\mathbf{z}_j - \mathbf{z}_{y_p})} \right)$
ℓ_2 regularizer ^a	$\lambda \ \boldsymbol{\omega}\ _2^2$	$\lambda \mathbf{z}^T \mathbf{H} \mathbf{z}$

^a The ℓ_2 regularizer can be added to any cost function $g(b, \boldsymbol{\omega})$ in the middle column and the resulting kernelized form of the sum $g(b, \boldsymbol{\omega}) + \lambda \|\boldsymbol{\omega}\|_2^2$ will be the sum of the kernelized cost and the kernelized regularizer, i.e., $g(b, \mathbf{z}) + \lambda \mathbf{z}^T \mathbf{H} \mathbf{z}$.

Popular kernels

$$\mathbf{H} = [\vec{h}_1 \dots \vec{h}_p]$$

The real value of kernelizing any machine learning cost is that the kernel matrix $\mathbf{H} = \mathbf{F}^T \mathbf{F}$ may be constructed *without* first building the matrix \mathbf{F} , as we will see through the examples below.

Q: what is the tensor shape of H?

$$\underline{\underline{p \times p}}$$

$$\hat{y} = \mathbf{h}^T \cdot \mathbf{z} + b$$

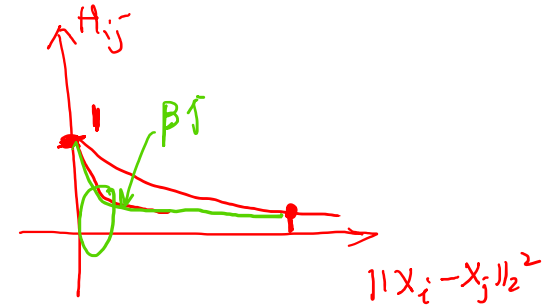
A **polynomial kernel matrix** can be defined entry-wise for general degree D and N dimensional input, as

$$\mathbf{H}_{ij} = \underbrace{(1 + \mathbf{x}_i^T \mathbf{x}_j)^D - 1}$$



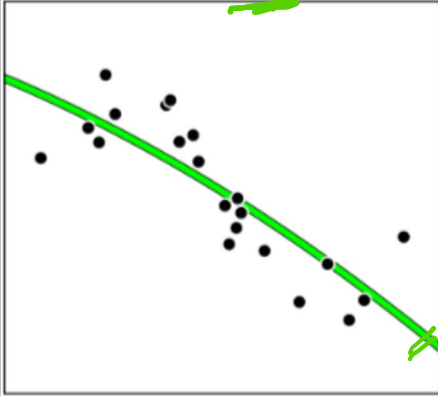
Another popular choice of kernel is the **radial basis function (RBF) kernel** which is typically defined explicitly as a kernel matrix over the input data as

$$\mathbf{H}_{ij} = e^{-\beta \| \mathbf{x}_i - \mathbf{x}_j \|_2^2}$$

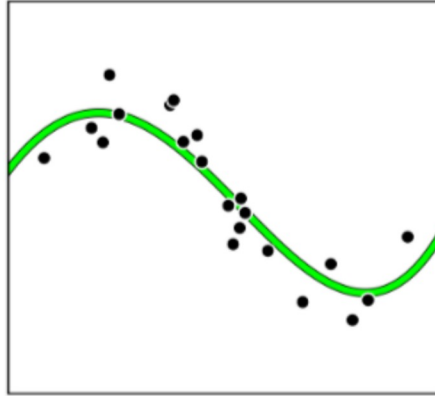


- The **larger** β is set the **higher the capacity** of the RBF kernel becomes.
- This is illustrated below via three examples: regression (top row), two-class (middle row), and multi-class (bottom row) classification.
- In each case we use using the RBF kernel with three distinct settings of β leading to underfitting, reasonable predictive behavior, and overfitting.

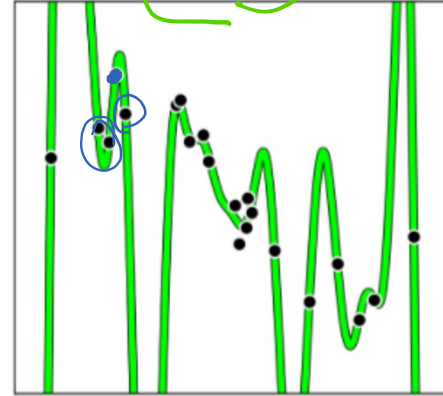
$\beta = 0.0001$



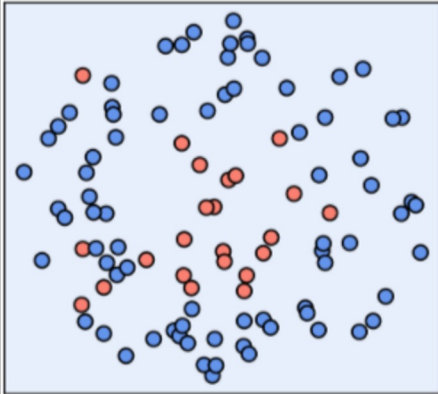
$\beta = 0.01$



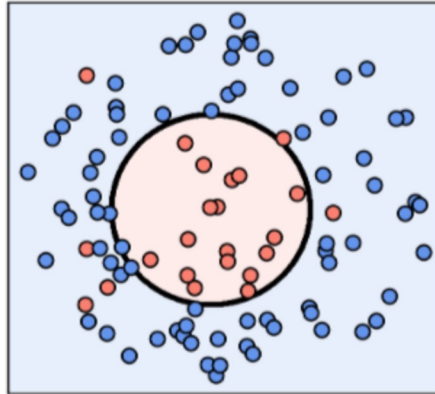
$\beta = 10$



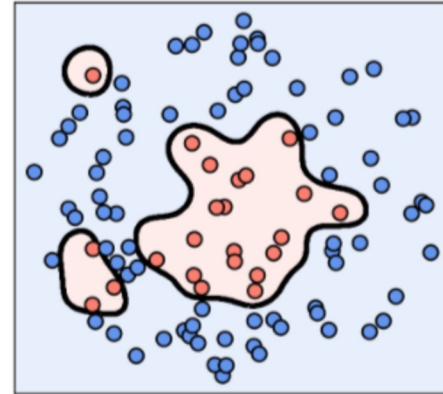
$\beta = 1e-08$



$\beta = 0.0001$



$\beta = 10$



Making predictions with kernelized models

- Assuming parameters b and ω have been fully tuned, the following model can be used for prediction purposes.

$$\text{model}(\mathbf{x}, b, \mathbf{z}) = b + \mathbf{h}^T \mathbf{z}$$

$\mathbf{h} \in \mathbb{R}^p$

- Here the kernelization **h** of the generic input **x** involves evaluation against *every point in the training set*. For instance, in the case of polynomials **h** is given as

$$\textcircled{\mathbf{h}} = \begin{bmatrix} (1 + \mathbf{x}_1^T \mathbf{x})^D + 1 \\ (1 + \mathbf{x}_2^T \mathbf{x})^D + 1 \\ \vdots \\ (1 + \mathbf{x}_P^T \mathbf{x})^D + 1 \end{bmatrix} \cdot \sum$$

- This necessity - to employ **every training point in making predictions** - is virtually unique to kernelized learners.

12.4 Kernels as Similarity Measures

- Studying the RBF kernel $\mathbf{H}_{ij} = \underbrace{e^{-\beta\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}}$ we can interpret it as a ***similarity measure*** between the two inputs \mathbf{x}_i and \mathbf{x}_j
- The more similar \mathbf{x}_i and \mathbf{x}_j are in the input space the larger \mathbf{H}_{ij} becomes, attaining the value 1 when $\mathbf{x}_i = \mathbf{x}_j$
- The further apart the two inputs the smaller \mathbf{H}_{ij} becomes, attaining 0 when they are infinitely far apart.

12.5 Optimization of Kernelized Models

- Because the final kernelized model remains linear in its parameters (the kernels themselves having no internal parameters tuned during optimization), corresponding kernelized cost functions themselves are quite 'nice' in terms of their general shape.
- For example, **any convex cost function for regression and classification *remains convex when kernelized***.
- This allows virtually any optimization method to be used to tune a kernelized supervised learner - from zero to first order and even powerful second order approaches like Newton's method.

- Notice, because kernel matrices **H** are sized $P \times P$ - where P is the size of the training set - **they inherently scale very poorly in the size of training data.**
- For example, with $P=10,000$ the corresponding kernel matrix will be of size, $10,000 \times 10,000$ with 10^8 values to store, far more than a modern computer can store all at once.

- Most standard ways of dealing with this crippling scaling issue revolve around avoiding the creation of the entire kernel matrix \mathbf{H} , especially during training.
- For example, one can use first order methods such as **stochastic gradient descent** to avoid construction of the entire kernel \mathbf{H} during training.
- Sometimes the explicit structure of certain problems allows can allow for the avoidance of explicit kernel construction as well.

[Prev](#)
[Up](#)
[Next](#)
scikit-learn 1.3.2
[Other versions](#)

Please **cite us** if you use the software.

1.4. Support Vector Machines

1.4.1. Classification

1.4.2. Regression

1.4.3. Density estimation, novelty detection

1.4.4. Complexity

1.4.5. Tips on Practical Use

1.4.6. Kernel functions

1.4.7. Mathematical formulation

1.4.8. Implementation details

1.4.6. Kernel functions

The *kernel function* can be any of the following:

- linear: $\langle x, x' \rangle$.
- polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`.
- ★ rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0.
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`.

Different kernels are specified by the `kernel` parameter:

```
>>> linear_svc = svm.SVC(kernel='linear')
>>> linear_svc.kernel
'linear'
>>> rbf_svc = svm.SVC(kernel='rbf')
>>> rbf_svc.kernel
'rbf'
```

<https://scikit-learn.org/stable/modules/svm.html#kernel-functions>

Summary

$$N=1 \quad f_1(x) = x \quad f_2(x) = x^2$$

$$\text{model: } w_0 + \underbrace{w_1 x + w_2 x^2}_{B = \{2\}}$$

2-degree
polys

- Fixed-Shape Universal Approximators
- The Kernel Trick
- Kernels as Similarity Measures
- Optimization of Kernelized Models
- Cross-Validating Kernelized Learners

x_p : p -th data point

$f_p^T \in \mathbb{R}^B$: p -th data point's transformed features

P : # samples

kernel:

$$y = b + f^T w$$

$\in \mathbb{R}^B$

$$y = b + h^T z$$

$\in \mathbb{R}^P$

$$f_1(x) = x_1 \quad f_2(x) = x_2 \quad \dots$$

$$N=2$$

$$\text{model} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 +$$

$$B = \{ w_4 x_1^2 + w_5 x_2^2 \}$$

2-degree

$$\uparrow \cong$$

$$B = \binom{N+D}{0} \sim N!$$