CS5339 Lecture Notes #5: Kernel Methods

Jonathan Scarlett

March 31, 2021

Useful references:

- Slide set `lecture_bo0.pdf` from a one-day course I gave[1]
- MIT lecture notes,[2] lectures 6 and 7
- Chapters 6 and 7 of Bishop's "Pattern Recognition and Machine Learning" book
- Chapter 16 of "Understanding Machine Learning" book
- Kernel cookbook[3]
- (Advanced) Lecture videos by Julien Mairal and Jean-Philippe Vert[4]

1 Feature Space Mappings

- In previous lectures, we looked at *linear classifiers* for binary labels:

$$\text{Predict positive label} \iff \boldsymbol{\theta}^T \mathbf{x} + \theta_0 > 0$$

  and also *linear predictors* for real-valued variables:

$$\hat{y}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0.$$

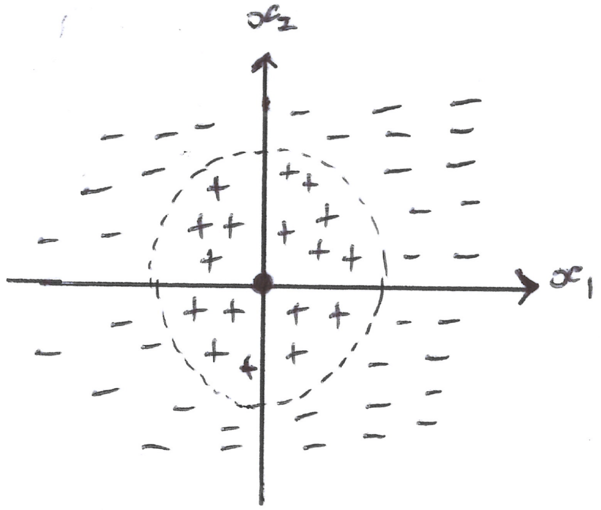
- What if $\mathbf{x}$ and y and related in a highly non-linear manner?
  - We can still use linear classification and regression methods!
- Motivating example 1: Fitting a quadratic. If we know that y is (well-approximated by) a quadratic function of a single input x, then we can use x to construct $\tilde{\mathbf{x}} = [x \;\; x^2]^T$, and then perform linear regression with input $\tilde{\mathbf{x}}$ and $\boldsymbol{\theta} \in \mathbb{R}^2$. Then $\hat{y} = \boldsymbol{\theta}^T \tilde{\mathbf{x}} + \theta_0$ evaluates to $\hat{y} = \theta_2 x^2 + \theta_1 x + \theta_0$ – an arbitrary quadratic function!

[1] `https://www.comp.nus.edu.sg/~scarlett/gp_slides`
[2] `http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-867-machine-learning-fall-2006/lecture-notes/`
[3] `http://www.cs.toronto.edu/~duvenaud/cookbook/`
[4] `https://www.youtube.com/channel/UCotztBOmGVl9pPGIN4YqcRw/videos`

- Motivating example 2: A circular classification region. Suppose that $d = 2$ and the binary labels are generated according to

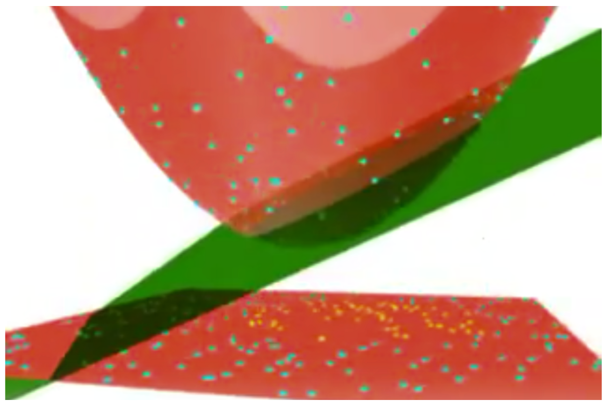
$$y_t = \begin{cases} +1 & x_1^2 + x_2^2 \le 1 \\ -1 & \text{otherwise} \end{cases}$$

yielding a circular region:



Any linear classifier with input $\mathbf{x} = [x_1 \;\; x_2]^T$ will perform poorly. But a linear classifier with input $\tilde{\mathbf{x}} = [x_1 \;\; x_2 \;\; x_1^2 + x_2^2]^T$ exists that classifies perfectly!

  - Visual illustration: `http://www.youtube.com/watch?v=3liCbRZPrZA`



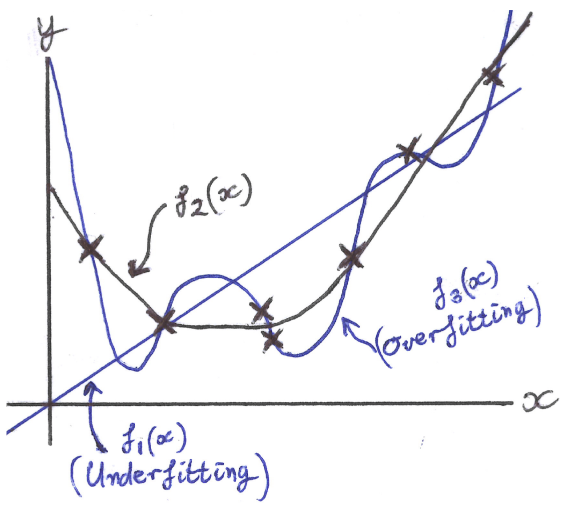
  - 2D Points at bottom (not linearly separable), mapping to 3D shown above (linearly separable)

- In general, we might hope to get a better classifier by mapping each $\mathbf{x}$ to a well-designed *feature space*:

$$\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_N(\mathbf{x})]^T$$

for some real-valued functions $\phi_1, \ldots, \phi_N$.

- **A word of caution.** We can always get to zero training error by adding more and more features, but is that always a good idea? Which prediction rule will work better in the following regression example?



  The most complex classifier achieves zero training error, but intuitively we should expect the quadratic classifier to generalize better to unseen points. We will explore the notions of *generalization error* and *overfitting* in a later lecture.

  - See the blog post `https://www.alexpghayes.com/blog/overfitting-a-guided-tour/#fn1` for a nice introduction

2 Kernel Methods

Overview.

- Many machine learning algorithms depend on the data $\mathbf{x}_1, \dots, \mathbf{x}_n$ only through the pairwise *inner products* $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$.

  - Examples to come later: Ridge regression, SVM
  - Example explored in the project: K-nearest neighbors (given a new point, find the K closest points in the training set, and classify according to a majority vote)

  Inner products capture the *geometry* of the data set, so one generally expects geometrically inspired algorithms (e.g., SVM) to depend only on inner products:

  - For algorithms that use distances, note that $\|\mathbf{x} - \mathbf{x}'\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{x}', \mathbf{x}' \rangle - 2\langle \mathbf{x}, \mathbf{x}' \rangle$, so distances can be expressed in terms of inner products.
  - For algorithms that use angles, note that $\mathrm{angle}(\mathbf{x}, \mathbf{x}') = \cos^{-1}\left(\frac{\langle \mathbf{x}, \mathbf{x}' \rangle}{\|\mathbf{x}\| \cdot \|\mathbf{x}'\|}\right)$, so since $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$, angles can also be expressed in terms of inner products.

- We know that moving to feature spaces can help, so we could map each $\mathbf{x}_i \to \boldsymbol{\phi}(\mathbf{x}_i)$ and apply the algorithm using $\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle$.

- A *kernel function* $k(\mathbf{x}_i, \mathbf{x}_j)$ can be thought of as an inner product in a *possibly implicit* feature space
  - **Key idea.** There are clever choices of the mapping $\boldsymbol{\phi}(\cdot)$ ensuring that we can efficiently compute $\langle \boldsymbol{\phi}(\mathbf{x}_i), \boldsymbol{\phi}(\mathbf{x}_j) \rangle$ *without ever explicitly mapping to the feature space*
  - In some cases, the feature space is infinite-dimensional, so we could not explicitly map to it even if we wanted to.
- **Kernel trick.** The terminology "kernel trick" simply refers to taking any algorithm the depends on the data only through inner products, and replacing each inner product $\langle \mathbf{x}, \mathbf{x}' \rangle$ by a kernel value $k(\mathbf{x}, \mathbf{x}')$.
- Intuition 1. The kernel function is a *measure of similarity* between $\mathbf{x}_i$ and $\mathbf{x}_j$.
- Intuition 2. The kernel trick applies to problems that depend on the *geometry* of the data (think of SVM for example!). In particular, note that $\|\mathbf{x} - \mathbf{x}'\|^2 = \langle \mathbf{x} - \mathbf{x}', \mathbf{x} - \mathbf{x}' \rangle = \langle \mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}, \mathbf{x}' \rangle + \langle \mathbf{x}', \mathbf{x}' \rangle$, so algorithms depending on distances can be rewritten in terms of inner products.

Example 1: Polynomial kernels.

- Start with the 1D setting, $d = 1$. Naively, one might map $x \xrightarrow{\boldsymbol{\phi}} (1, x, x^2)$ for quadratic features, $x \xrightarrow{\boldsymbol{\phi}} (1, x, x^2, x^3)$ for cubic features, and so on.
  - Similarly for higher dimensions, e.g., $(x_1, x_2) \xrightarrow{\boldsymbol{\phi}} (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Notice the presence of the cross-term x_1x_2.
- Now, considering the cubic example, I will claim that $x \xrightarrow{\boldsymbol{\phi}} (1, \sqrt{3}x, \sqrt{3}x^2, x^3)$ is a "better" choice than the one above mapping to $(1, x, x^2, x^3)$.
  - First note that the set of all possible linear classifiers (meaning linear in $\boldsymbol{\phi}(\mathbf{x})$) is exactly the same regardless of the choice of mapping. This is because the middle two coefficients are just scaled up or down by $\sqrt{3}$.
  - But notice the inner product simplifies nicely under the second choice:

$$\begin{aligned}\langle \boldsymbol{\phi}(x), \boldsymbol{\phi}(x') \rangle &= 1 + 3xx' + 3x^2(x')^2 + x^3(x')^3 \\ &= (1 + xx')^3.\end{aligned}$$

- Generalizing this idea to polynomials of degree p and functions in d dimensions, we arrive at the *polynomial kernel* $k(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^p$. The computational savings of avoiding the construction of $\boldsymbol{\phi}(\mathbf{x})$ can be **much** more significant than the above example.
  - Instead of computing a huge number of features (specifically, it can be shown to be $\binom{p+d}{d}$), the computation time is *linear in d* (do d multiplications, sum them, and apply $(1 + \text{result})^p$)

Example 2: String kernels

- Continuing the interpretation of a kernel being a measure of similarity, how can we "measure similarity" between the following?
  - $\mathbf{x}_1$ = "This sentence is the first string in my data set"
  - $\mathbf{x}_2$ = "The second string in my data set is this sentence"

  - $\mathbf{x}_3$ = "Tihs sentance is the third stirng in my dataset"

- One possible approach is to let $k(\mathbf{x},\mathbf{x}')$ be the number of words appearing in both strings. This corresponds to a feature space with dimension equal to the total number of possible words, and with $\phi_j(\mathbf{x}) = \mathbf{1}\{\mathbf{x} \text{ contains the } j\text{-th word}\}$. This approach has limitations, such as not handling spelling errors well.

- A more "robust" approach is the *String Subsequence Kernel* (SSK), which looks at substrings that are not necessarily contiguous (e.g., "sentnce" is a substring of both "sentence" and "sentance"), but decreases the weight when this substring is spread a cross a longer length (e.g., "sentnce" is also a substring of "sent this once", but this is spread over a longer length so carries less weight).
  - Can be computed somewhat-efficiently using dynamic programming techniques.
  - If exact computation is still too demanding, approximation computation can be used instead.
  - The idea of using substrings instead of exact matches is also very important in biology applications.

Other examples.

- Below we will introduce the commonly-used RBF kernel:

$$k_{\mathrm{RBF}}(\mathbf{x},\mathbf{x}') = \exp\bigg(-\frac{1}{2}\|\mathbf{x}-\mathbf{x}'\|^2\bigg),$$

  which indicates that similarity exponentially decays to zero as the squared distance increases.

- The kernel cookbook (link on p1) gives several other examples and insights.

- For data represented in a not-so-standard format (e.g., text, graphs), more unconventional/creative choices of kernels are often adopted.

Formal definition.

- **Definition.** A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is said to be a positive semidefinite (PSD) kernel if (i) it is symmetric, i.e., $k(\mathbf{x},\mathbf{x}') = k(\mathbf{x}',\mathbf{x})$; (ii) For any integer $m > 0$ and any set of inputs $\mathbf{x}_1,\dots,\mathbf{x}_m$ in $\mathbb{R}^d$, the following matrix is positive semi-definite:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1,\mathbf{x}_1) & \dots & k(\mathbf{x}_1,\mathbf{x}_m) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_m,\mathbf{x}_1) & \dots & k(\mathbf{x}_m,\mathbf{x}_m) \end{bmatrix} \succeq \mathbf{0}.$$

  This matrix, with (i,j)-th entry equal to $k(\mathbf{x}_i,\mathbf{x}_j)$, is called the *kernel matrix* (you might also see it referred to as the *Gram matrix*).

- **Theorem.** A function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a PSD kernel if and only if it equals an inner product $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\rangle$ for some (possibly infinite dimensional) mapping $\boldsymbol{\phi}(\mathbf{x})$.[5]

[5] In fact, this statement is a bit imprecise, since in general it may need to be a generalized notion of inner product beyond standard vector spaces (namely, to *Hilbert spaces*). However, we will avoid such technicalities and focus on the standard inner product applied to real-valued vectors.

- The "if" part is easy to show (at least when $\boldsymbol{\phi}$ is finite-dimensional): The inner product is certainly symmetric, and the Gram matrix can be written as $\mathbf{K} = \mathbf{\Phi}^T\mathbf{\Phi}$, where $\mathbf{\Phi} \in \mathbb{R}^{\dim(\boldsymbol{\phi})\times m}$ contains the m feature vectors $\{\boldsymbol{\phi}(\mathbf{x}_t)\}_{t=1}^m$ as columns. The matrix $\mathbf{K} = \mathbf{\Phi}^T\mathbf{\Phi}$ is trivially positive semidefinite, since for any $\mathbf{z}$ we have $\mathbf{z}^T\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{z} = \|\mathbf{\Phi}\mathbf{z}\|^2 \ge 0$.
- The "only if" part is more challenging, but can be understood fairly easily in the case that $\mathbf{x}$ only takes on finitely many values, using the idea of eigenvalue decomposition.[6] Such an approach can be extended to more general scenarios via *Mercer's theorem*, and a fully general treatment is possible via the notion of a *Reproducing Kernel Hilbert Spaces* (RKHS).

3 Constructing More Complicated Kernels from Simpler Ones

- **Claim.** If k_1 and k_2 are kernels, then so are the following:

  1. $k(\mathbf{x},\mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}')$ for some function f
  2. $k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}')$
  3. $k(\mathbf{x},\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}')k_2(\mathbf{x},\mathbf{x}')$

  To set up notation, let $\boldsymbol{\phi}^{(1)}(\mathbf{x})$ and $\boldsymbol{\phi}^{(2)}(\mathbf{x})$ be the feature vectors corresponding to k_1 and k_2.

- Proof of #1.
  - Let $\boldsymbol{\phi}(\mathbf{x}) = f(\mathbf{x})\boldsymbol{\phi}^{(1)}(\mathbf{x})$. Then

$$\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\rangle = \boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}'),$$

    which is the desired $k(\mathbf{x},\mathbf{x}')$ in the first claim.

- Proof of #2.
  - Let $\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \boldsymbol{\phi}^{(1)}(\mathbf{x}) \\ \boldsymbol{\phi}^{(2)}(\mathbf{x}) \end{bmatrix}$, and write

$$\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\rangle = \boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}') = k_1(\mathbf{x},\mathbf{x}') + k_2(\mathbf{x},\mathbf{x}').$$

- Proof of #3.
  - Let $\boldsymbol{\phi}(\mathbf{x})$ contain entries $\boldsymbol{\phi}_{ij}(\mathbf{x}) = \boldsymbol{\phi}_i^{(1)}(\mathbf{x})\boldsymbol{\phi}_j^{(2)}(\mathbf{x})$ for each i,j, where $\boldsymbol{\phi}_i^{(1)}$ is the i-th feature in $\boldsymbol{\phi}^{(1)}$ and similarly for $\boldsymbol{\phi}_j^{(2)}$.

[6]Supposing that $\mathbf{x}$ can only take values in a finite set $\{\mathbf{x}_1,\dots,\mathbf{x}_m\}$, the entire function is described by an $m \times m$ matrix $\mathbf{K}_{\text{full}}$ with (i,j)-th entry $k(\mathbf{x}_i,\mathbf{x}_j)$. By assumption $\mathbf{K}_{\text{full}}$ is a PSD matrix, and then it is known from linear algebra that it admits an eigenvalue decomposition of the form $\mathbf{K}_{\text{full}} = \sum_{j=1}^m \lambda_j \mathbf{v}_j\mathbf{v}_j^T$. This fact can easily be translated to a length-m feature map with i-th entry given by $\phi(\mathbf{x}_j) = \sqrt{\lambda_i}(\mathbf{v}_i)_j$, where $(\mathbf{v}_i)_j$ is the j-th entry of $\mathbf{v}_i$.

– We have

$$\begin{aligned}
\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle &= \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}) \\
&= \sum_{i,j} \boldsymbol{\phi}_{ij}(\mathbf{x}) \boldsymbol{\phi}_{ij}(\mathbf{x}') \\
&= \sum_{i,j} \boldsymbol{\phi}_i^{(1)}(\mathbf{x}) \boldsymbol{\phi}_j^{(2)}(\mathbf{x}) \boldsymbol{\phi}_i^{(1)}(\mathbf{x}') \boldsymbol{\phi}_j^{(2)}(\mathbf{x}') \\
&= \sum_{i} \boldsymbol{\phi}_i^{(1)}(\mathbf{x}) \boldsymbol{\phi}_i^{(1)}(\mathbf{x}') \sum_{j} \boldsymbol{\phi}_j^{(2)}(\mathbf{x}) \boldsymbol{\phi}_j^{(2)}(\mathbf{x}') \\
&= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}').
\end{aligned}$$

Example 3: Radial basis function (RBF) kernel

- Letting $k_1(\mathbf{x}, \mathbf{x}') = k_2(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$ in property 3 above, we see that $(\mathbf{x}^T\mathbf{x}')^2$ is a PSD kernel. Repeating several times, the same is true of $(\mathbf{x}^T\mathbf{x}')^p$ for any integer p. (Side note: A similar argument gives an alternative proof that the polynomial kernel $(1 + \mathbf{x}^T\mathbf{x}')^p$ is a PSD kernel.)

- By the identity $e^t = \sum_{j=0}^{\infty} \frac{t^j}{j!}$, and applying property 2 an "infinite number of times" (a formal proof of the validity of this is omitted here), we deduce that $e^{\mathbf{x}^T\mathbf{x}}$ is also a PSD kernel.

- Finally, define

$$\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= \exp\bigg(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2 \bigg) && (1) \\
&= \exp\bigg(-\frac{1}{2}\|\mathbf{x}\|^2 \bigg) \cdot \exp\big(\mathbf{x}^T\mathbf{x}' \big) \cdot \exp\bigg(-\frac{1}{2}\|\mathbf{x}'\|^2 \bigg), && (2)
\end{aligned}$$

  and use property 1 to deduce that $k(\mathbf{x}, \mathbf{x}')$ is a PSD kernel.

- This kernel goes by several names: Radial basis function (RBF) kernel, Gaussian kernel, squared exponential kernel. It is usually defined with a *length-scale parameter* ℓ:

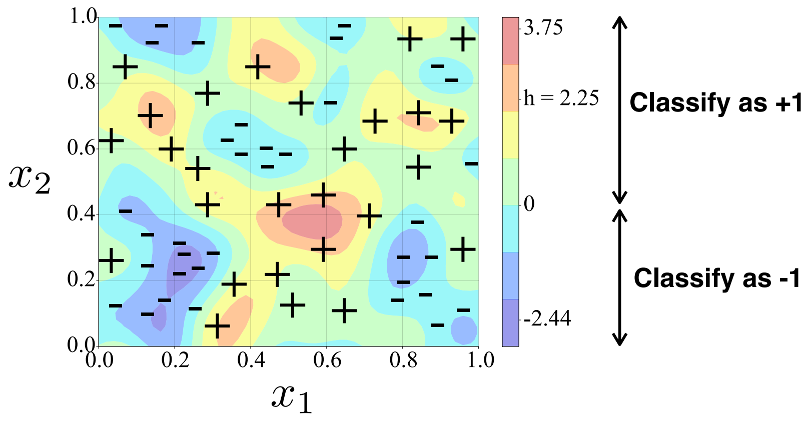
$$k(\mathbf{x}, \mathbf{x}') = \exp\bigg(-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2 \bigg).$$

  Such a parameter represents the rough scale over which the function varies (example below).

  – More generally, can have different lengths in each dimension, $\boldsymbol{\ell} = (\ell_1, \dots, \ell_d)$.

- The associated feature space is infinite-dimensional, as hinted by the fact that we used the infinite expansion $e^t = \sum_{j=0}^{\infty} \frac{t^j}{j!}$ in its derivation.

- An illustration of the sorts of classification regions that can be produced by the kernel SVM (to be introduced in the next lecture) with an RBF kernel
  – Here the prediction rule is of the form $\hat{y} = \text{sign}(g(\mathbf{x}))$ as usual, and the colors in this figure represent the values of $g(\mathbf{x})$.



4 Linear Regression Revisited

- Here we look at linear regression with kernels. In the next lecture we will look at SVM with kernels. The same can be done for logistic regression, but we will skip that.
- We previously considered the regularized least squares estimator (ridge regression); in the case of no offset ($\theta_0 = 0$), it is written as follows (with $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}$):

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2 + \lambda\|\boldsymbol{\theta}\|^2,$$

  and has the closed-form solution

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}.$$

- Now let's apply some useful matrix manipulations. To help with readability, let $\mathbf{I}_d$ and $\mathbf{I}_n$ denote identity matrices with the size made explicit. First observe

$$(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_d)\mathbf{X}^T = \mathbf{X}^T\mathbf{X}\mathbf{X}^T + \lambda\mathbf{X}^T = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_n).$$

  Multiplying by $(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_d)^{-1}$ on the left and $(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_n)^{-1}$ on the right gives

$$\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_n)^{-1} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_d)^{-1}\mathbf{X}^T,$$

  meaning we obtain the following equivalent form for $\hat{\boldsymbol{\theta}}$:

$$\hat{\boldsymbol{\theta}} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y}.$$

- Therefore, given a new input $\mathbf{x}' \in \mathbb{R}^d$, the prediction $\hat{y}(\mathbf{x}') = \hat{\boldsymbol{\theta}}^T\mathbf{x}' = (\mathbf{x}')^T\hat{\boldsymbol{\theta}}$ can be written as

$$\hat{y}(\mathbf{x}') = (\mathbf{x}')^T\mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I})^{-1}\mathbf{y}.$$

- Crucial observation. The prediction depends on the data only through inner products, since

$$(\mathbf{x}')^T\mathbf{X}^T = \begin{bmatrix} \langle \mathbf{x}', \mathbf{x}_1 \rangle \\ \vdots \\ \langle \mathbf{x}', \mathbf{x}_n \rangle \end{bmatrix}^T, \qquad \mathbf{X}\mathbf{X}^T = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \dots & \langle \mathbf{x}_1, \mathbf{x}_n \rangle \\ \vdots & \ddots & \vdots \\ \langle \mathbf{x}_n, \mathbf{x}_1 \rangle & \dots & \langle \mathbf{x}_n, \mathbf{x}_n \rangle \end{bmatrix}.$$

- Therefore, we can apply the **kernel trick** and consider the more general prediction function

$$\hat{y}(\mathbf{x}') = \mathbf{k}(\mathbf{x}')(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}, \tag{3}$$

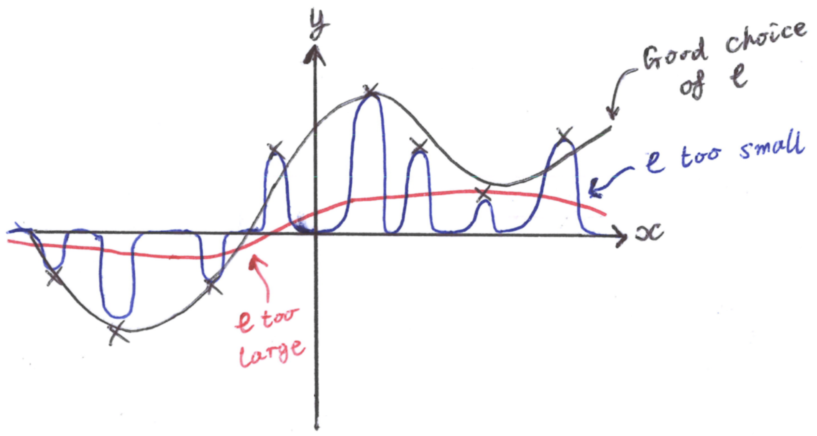
where

$$\mathbf{k}(\mathbf{x}') = \begin{bmatrix} k(\mathbf{x}', \mathbf{x}_1) \\ \vdots \\ k(\mathbf{x}', \mathbf{x}_n) \end{bmatrix}^T, \qquad \mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}.$$

This is known as *kernel ridge regression.*

  - **Rough intuition behind equation** (3)**.** The estimate $\hat{y}$ is a weighted sum of the previously-observed outputs $y_1, \dots, y_n$. The more similar $\mathbf{x}'$ is to the corresponding $\mathbf{x}_t$ (i.e., the higher $k(\mathbf{x}, \mathbf{x}')$), the more weight is given to that $\mathbf{y}$. (But the similarities among training points themselves also play a role through the presence of $\mathbf{K}$)
  - Can be thought of as performing regularized least squares on the model $y = \boldsymbol{\theta}^T\boldsymbol{\phi}(\mathbf{x}) + z$, where $\boldsymbol{\phi}(\mathbf{x})$ is the feature vector corresponding to the kernel k, and z is Gaussian noise.

- An example using the RBF kernel (with length-scale ℓ):



  - Observe that even among a given class of kernels, the choice of their parameter(s) may be very important (e.g., length-scale ℓ in this example, degree p in polynomial example, etc.).
  - Often the parameters are chosen using maximum likelihood.
  - We will also cover *model selection* later, a special case of which is kernel selection.