446 Section $\phi(7)$

TA: Yufei Zhang

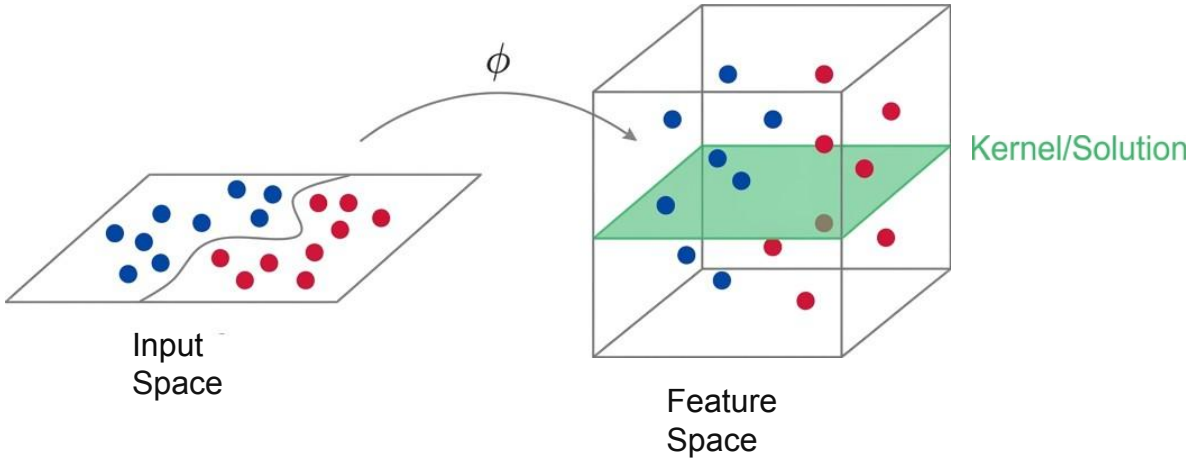
Plans for today!

1. This
2. Reminders
3. Kernels
4. Problem 2a, 2b
5. PyTorch Colab Notebook (No Demo)

Reminders

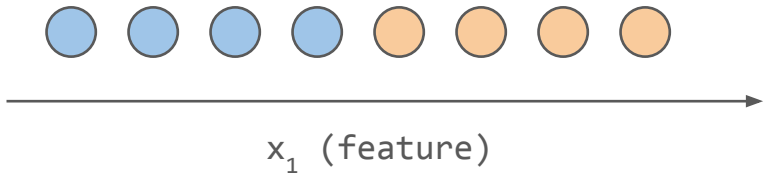
- HW3 due **Wed, Feb 25 @ 11:59pm**
- Midterm grades out!
  - Please only submit a regrade request if you have a *strong* case for extra points

Kernels



Motivation

Consider a 1D dataset with just feature x_1. The blue and orange dots denote two different classes.



Can you draw a horizontal line to separate these classes?

No

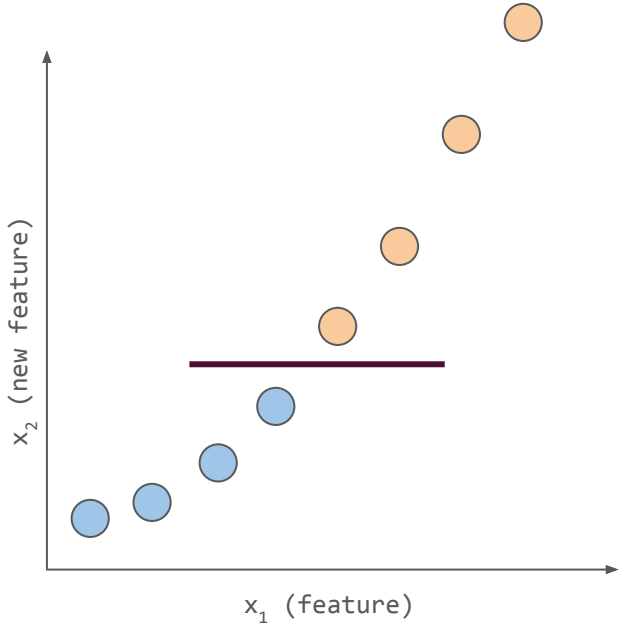
Kernels

We need to create a new feature x_2 from what we have (which is x_1)

$$\phi(x) = [\phi_1(x), \phi_2(x)]$$

$$\phi_1(x) = x$$

$$\phi_2(x) = x^2$$



Can you draw a horizontal line to separate these classes?

Yes!

Problems with efficiency ← THIS IS WHY!

If we did things step by step…

1. Take our *n* data points, each one a *d* dimensional vector
2. To EACH datapoint, apply our kernel map ($\phi(x)$), further blowing up the space/time complexity
3. Perform linear regression on the exploded set of features
4. Most likely fail…

Kernel trick!

Ridge Regression Reminders:

Variables:

$$X \in \mathbb{R}^{n \times d};\ y \in \mathbb{R}^{n};\ w \in \mathbb{R}^{d}$$

Optimization objective:

$$\hat{w} = \underset{w}{\operatorname{argmin}} ||y - Xw||_2^2 + \lambda ||w||_2^2$$

Closed-form solution:

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

Our goal is to get from **ridge regression** to **kernelized linear regression**

Step 1: The Representer Theorem

The optimal weight vector $\hat{w}$ lies in the span of the data points.

$$\hat{w} = X^T \boldsymbol{\alpha} = \sum_{i=1}^{n} x_i \alpha_i$$

Where $\alpha \in \mathbb{R}^n$ is a vector of coefficients.

This fact implies that **we can write *w* as a linear combination** of the data points (x_i) and some scale factors (a_i) → useful for out proof later!

Let's do feature expansion!

From now on we are working with the *expanded* feature set:

$$X \to \phi(X)$$

NEW Optimization objective:

$$\hat{w}_\phi = \underset{w}{\text{argmin}} ||y - \phi(X)w||_2^2 + \lambda||w||_2^2$$

What if our feature set blows up so that d >> n? Or even blows up to infinite dimensions?

Step 2: Substitution

$$X \rightarrow \phi(X)$$

$$\hat{w}_\phi = \underset{w}{\operatorname{argmin}} ||y - \phi(X)[w]||_2^2 + \lambda ||[w]||_2^2$$

$$\text{If } w = \phi(X)^\top a$$

$$\hat{\alpha} = \underset{\alpha}{\operatorname{argmin}} \; \|y - \phi(X)[\phi(X)^T \alpha]\|_2^2 + \lambda \|[\phi(X)^T \alpha]\|_2^2$$

This is why we needed step 1. It makes this substitution possible **no matter the dimensionality** of the “blown up” X!

Step 2: Define the Kernel Matrix

The **Kernel Function**:

$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

(Measures similarity between two points)

The **Kernel Matrix**:

$\mathbf{K} \in \mathbb{R}^{n \times n}$

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$

How can we construct the kernel **matrix** using the kernel *function?*

Step 2: Define the Kernel Matrix

The **Kernel Function:**

$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$

(Measures similarity between two points)

The **Kernel Matrix:**

$\mathbf{K} \in \mathbb{R}^{n \times n}$

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$

$$\mathbf{K} \begin{bmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_n) \end{bmatrix}$$

The Trick: We can compute **K** without ever calculating ϕ.

$$K(x_i, x_j) = \phi(x_i)\,\phi(x_j)$$

$$K_{ij} \in \mathbb{R}^{n \times n}, \quad K_{ij} = K(x_i, x_j)$$

$$K : n \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_2)^T\phi(x_1) & \cdots & \phi(x_n)^T\phi(x_1) \\ \phi(x_1)^T\phi(x_2) & \ddots & & \\ \vdots & & \ddots & \\ \phi(x_1)^T\phi(x_n) & & & \phi(x_n)^T\phi(x_n) \end{bmatrix} \quad (n \text{ columns})$$

The main takeaway is the formulation of the kernel matrix below.

$$\Downarrow$$

$$n \begin{bmatrix} — & \phi(x_1) & — \\ & \vdots & \\ — & \phi(x_n) & — \end{bmatrix} \quad (m \text{ columns}) \qquad m \begin{bmatrix} | & & | \\ \phi(x_1)^T & \cdots & \phi(x_n)^T \\ | & & | \end{bmatrix} \quad (n \text{ columns})$$

$$K = \phi(X)\,\phi(X)^T \;:\; X \in \mathbb{R}^{n \times m}$$

Substitute into Linear Regression

$$\text{If } \mathbf{K} = \phi(X)\phi(X)^\top$$

$$\begin{aligned}
\hat{a} &= \underset{a}{\operatorname{argmin}} ||y - \phi(X)\phi(X)^\top a||_2^2 + \lambda ||\phi(X)^\top a||_2^2 \\
&= \underset{a}{\operatorname{argmin}} ||y - \phi(X)\phi(X)^\top a||_2^2 + \lambda a^\top \phi(X)\phi(X)^\top a \\
&= \underset{a}{\operatorname{argmin}} ||y - \mathbf{K}a||_2^2 + \lambda a^\top \mathbf{K} a
\end{aligned}$$

The dimension d has completely vanished.
The problem now depends only on n.

Now you are gonna take this all the way to the end!

Step 3: Derive

2a. Kernelized Linear Regression

Recall that the definition of a kernel is the following:
Definition 1. A function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a *kernel* for a map ϕ if $K(x, x') = \phi(x) \cdot \phi(x') = \langle \phi(x), \phi(x') \rangle$ for all x, x'.

Consider regularized linear regression (without a bias, for simplicity). Our objective to find the optimal parameters $\hat{w} = \arg\min_w L(w)$ for a dataset $(x_i, y_i)_{i=1}^n$ that minimize the following loss function:

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

Note that from class, we know there is an optimal $\hat{w}$ that lies in the span of the datapoints. Concretely, there exist $\alpha_1, ..., \alpha_n \in \mathbb{R}$ such that $\hat{w} = \sum_i^n \alpha_i x_i$. Also recall from lecture that the expression of our loss function $L(w)$ in terms of the kernel is:

$$L(w) = ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Solve for the optimal α

2a. Kernelized Linear Regression

$$L(w) = ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda\alpha^T\mathbf{K}\alpha$$

1. **Gradient:** Take derivative of $L(\alpha)$ w.r.t α:
$$\nabla_\alpha L = -2K(y - K\alpha) + 2\lambda K\alpha$$
2. **First-Order Optimality:** Set to 0 and factor out K:
$$K(K\alpha - y + \lambda\alpha) = 0$$
$$K((K + \lambda I)\alpha - y) = 0$$
3. **Invert:** Assuming K is positive semi-definite (PSD):
$$(K + \lambda I)\alpha = y$$
4. **Solution:**
$$\boxed{\hat{\alpha} = (K + \lambda I)^{-1} y}$$

$$\nabla_\alpha L(w) = 0$$

$$\begin{aligned}
-2\mathbf{K}(\mathbf{y} - \mathbf{K}\alpha) + 2\lambda\mathbf{K}\alpha &= 0 \\
-\mathbf{K}(\mathbf{y} - \mathbf{K}\alpha) + \lambda\mathbf{K}\alpha &= 0 \\
\mathbf{K}(\mathbf{K}\alpha - \mathbf{y} + \lambda\alpha) &= 0 \\
\mathbf{K}((\mathbf{K} + \lambda I)\alpha - \mathbf{y}) &= 0 \\
\mathbf{K}(\mathbf{K} + \lambda I)\alpha &= \mathbf{K}\mathbf{y} \\
\hat{\alpha} &= (\mathbf{K} + \lambda I)^{-1}\mathbf{y}
\end{aligned}$$

Quick high-level recap

A kernel $K(a, b)$ takes in d dimensional vectors a, b and gives us a scalar.

When you construct a kernel such that $K(a, b) = \phi(a)^\top \phi(b)$

We can ultimately use K to efficiently apply ϕ to our features.

Can have infinite dimensions

Step 4: Predict

Normal case: $\hat{y} = \hat{w} \cdot z \quad : \quad z \in \mathbb{R}^d$

Kernel case: $\hat{y} = \hat{w}_\phi \cdot \phi(z) \qquad \hat{w}_\phi = \phi(x)^T a$

$$= \phi(x)^T a \, \phi(z)$$

$$= \sum_{i=1}^{n} a_i \phi(x_i)^T \phi(z)$$

$$= \sum_{i=1}^{n} a_i K(x_i, z)$$

Let's vectorize this! →

You need some extra steps to predict a new datapoint, **but the predictive power compared to computational cost is well worth it!**

2b. Kernelized Linear Regression

$$L(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda ||w||_2^2 \qquad L(w) = ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Let us assume that we were using a linear kernel where $\mathbf{K}_{ij} = x_i^T x_j$. Suppose we have $\mathbf{X}_{\text{test}}$ that we want to make prediction for after training on $\mathbf{X}_{\text{train}}$. Express the estimate $\hat{\mathbf{Y}}$ in terms of $\mathbf{K}_{train} = \mathbf{X}_{train}\mathbf{X}_{train}^T$, $\mathbf{y}_{train}$, $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$. What would the general prediction formula look like if we are not using a linear kernel? Express the solution in terms of $\mathbf{K}_{\text{train, test}}$ **Solution:**

2b. Kernelized Linear Regression

$$L(w) = \sum_{i=1}^{n} (w^T x_i - y_i)^2 + \lambda ||w||_2^2 \qquad L(w) = ||\mathbf{y} - \mathbf{K}\alpha||_2^2 + \lambda \alpha^T \mathbf{K} \alpha$$

Let us assume that we were using a linear kernel where $\mathbf{K}_{ij} = x_i^T x_j$. Suppose we have $\mathbf{X}_{\text{test}}$ that we want to make prediction for after training on $\mathbf{X}_{\text{train}}$. Express the estimate $\hat{\mathbf{Y}}$ in terms of $\mathbf{K}_{train} = \mathbf{X}_{train}\mathbf{X}_{train}^T$, $\mathbf{y}_{train}$, $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$. What would the general prediction formula look like if we are not using a linear kernel? Express the solution in terms of $\mathbf{K}_{\text{train, test}}$ **Solution:**

Express the estimate $\hat{Y}$:

$$\hat{Y} = X_{test}\hat{w}$$
$$= X_{test} X_{train}^T \hat{\alpha}$$
$$= X_{test} X_{train}^T (K_{train} + \lambda I)^{-1} y_{train}$$

General formula:

$$= X_{test} X_{train}^T \hat{\alpha}$$
$$= K_{test,train}\hat{\alpha}$$
$$(m \times n)$$

Takeaways

1. **Efficiency**: Kernels allow us to operate in infinite dimensions without the computational cost (n vs d)
2. **Substitution**: We replace explicit features phi(x) with pairwise dot products K
3. **Result**: A regression model solved using only the data count n

Mercer’s Conditions

Not just any function can be a Kernel. To ensure the optimization problem is convex and solvable, the Kernel Matrix **K** must satisfy:

- **Symmetric:**

$$K_{ij} = K_{ji}$$

(The similarity between A and B is the same as B and A).

- **Positive Semi-Definite (PSD):**

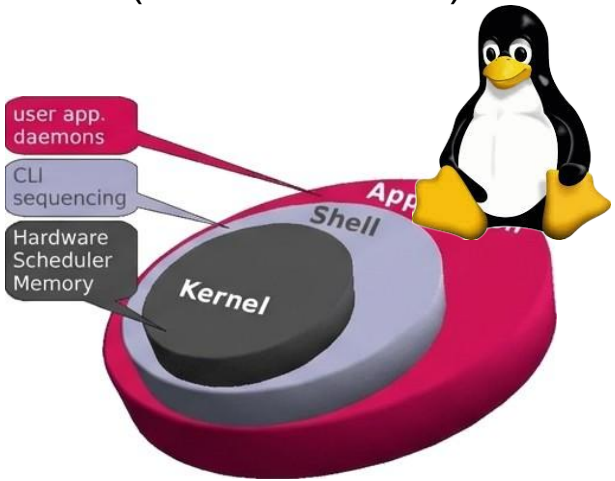
$$\mathbf{v}^T K \mathbf{v} \geq 0 \ \ \forall \, \mathbf{v}$$

(Ensures the loss surface is a bowl, not a saddle).

A Tale of Three Kernels

Context matters.

OS Kernel
(Linux/Windows)



Corn Kernel
(Zea mays)

Image Kernel
(Convolutions/CNNs)

ML Kernel Method: A similarity function $K(x, z) = \langle \phi(x), \phi(z) \rangle$.

Do this in your own time if you want a tutorial on **how to make a neural net** in PyTorch!

PyTorch Colab