446 Section 07

TA: Varun Ananth

Plans for today!

1. This
2. Reminders
3. Kernelized Linear Regression
4. PyTorch Colab Notebook (No Demo)

Reminders

- HW2 due 2 days ago
- HW3 due next 11/19 @ 11:59 PM
- Midterm grades out!
  - Please only submit a regrade request if you have a *strong* case for extra points

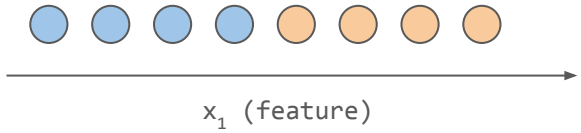
Nothing else to say… How’s life?

Kernels

Kernels ?????????????????? ??????????????

This concept stumped me when I took the class. It can be difficult to understand so spend some time digesting it

- But maybe we can simplify things… Let's start with *why*



Can you draw a horizontal line to separate these classes?

No

Kernels
??????????????????
??????????????

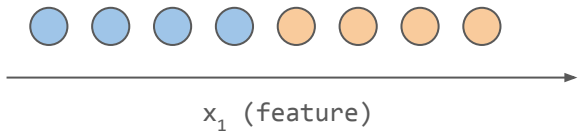
We need to create a new feature from what we have (which is x_1)

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

Feature map

$$\phi_1(x) = x$$

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



Can you draw a horizontal line to separate these classes?

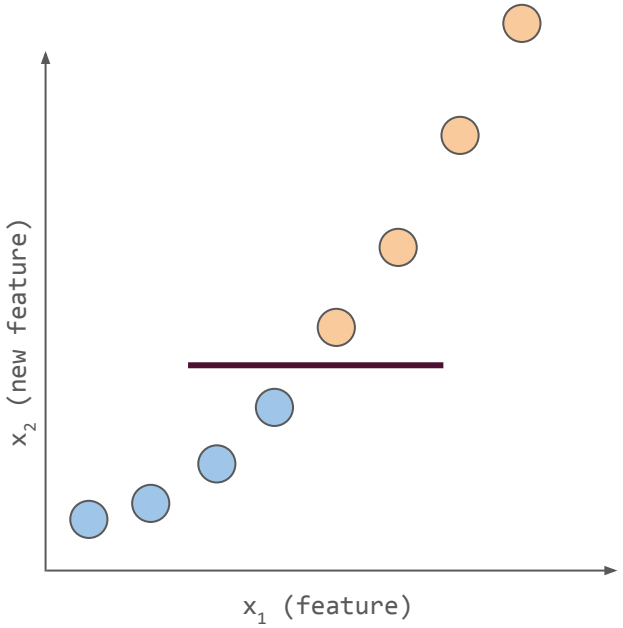
Kernels
??????????????????
??????????????

We need to create a new feature 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:

(You can write these down if you want to follow along!)

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**

It involves multiple pieces that *seem* disjoint but **will** fit together at the end, so feel free to ask for some backtracking

Step 1: Show $\hat{w} = X^\top a$ for some a

↳ same as showing $w \in \text{span}(X)$

Why is this important?

1. We need to show **we can find weights *w* that are in the span of the datapoints** because…
2. It implies that **we can write *w* as a linear combination** of the datapoints (x_i) and some scale factors (a_i)...
3. And this fact is **crucial** for some later algebra

In other words:

We want to show this is true *even if* d > n

$$\hat{w} = X^\top a = \sum_{i=1}^{n} x_i a_i$$

Step 1: Show $\hat{w} = X^T a$ for some a

↳ same as showing $w \in \text{span}(X)$

(1)

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

(2)

$$= X^T (X X^T + \lambda I) y$$

$$\hat{w} = X^T a$$

$$\begin{cases} (X^T X + \lambda I) X^T = X^T (X X^T + \lambda I) \\ X^T = (X^T X + \lambda I)^{-1} X^T (X X^T + \lambda I) \\ X^T (X X^T + \lambda I)^{-1} = (X^T X + \lambda I)^{-1} X^T \end{cases}$$

(2) (1)

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

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

② $= X^\top (X X^\top + \lambda I) y$

$\hat{w} = X^\top a$

Step 1: Show $\hat{w} = X^\top a$ for some a

↳ Same as showing $w \in \text{span}(X)$

$\therefore$ w can be expressed as a linear combination of $(x_1, x_2 \dots x_n)$

$$w = \sum_{i=1}^{n} x_i a_i = X^\top a$$

(Note that this is always true of *w* at any point along the optimization trajectory not just at the end… but that proof is beyond the scope here)

Let's do feature expansion!

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

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

NEW Optimization objective:

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

But our computational demons surface again. What if our feature set blows up so that d >> n? Or even blows up to infinite dimensions?

You actually already implemented this with polynomial regression

Step 2: Rewrite Ridge Regression with kernels

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

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

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

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

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

Step 2b: Define the kernel matrix

(I know this seems like a tangent. Trust me it will all come full circle)

Important distinction!

- Kernel function: $K(x_i, x_j) = \phi(x_i)\phi(x_j)$

vs.

- Kernel matrix: $K_{ij} \in \mathbb{R}^{n \times n}$, $K_{ij} = K(x_i, x_j)$

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

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

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

$$K : \begin{matrix} \\ n \end{matrix} \overset{\overset{n}{\downarrow}}{\begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_2)^T\phi(x_1) & \cdots & \phi(x_n)^T\phi(x_1) \\ \phi(x_1)^T\phi(x_2) & \ddots & & \\ \vdots & & \ddots & \\ \phi(x_1)^T\phi(x_n) & & & \phi(x_n)^T\phi(x_n) \end{bmatrix}}$$

Take a second to convince yourself this is true. **The main takeaway is the formulation of the kernel matrix below.**

$$\begin{matrix} \\ n \end{matrix} \overset{\overset{m}{\downarrow}}{\begin{bmatrix} — & \phi(x_1) & — \\ & \vdots & \\ — & \phi(x_n) & — \end{bmatrix}} \quad \begin{matrix} \\ m \end{matrix} \overset{\overset{n}{\downarrow}}{\begin{bmatrix} | & & | \\ \phi(x_1)^T & \cdots & \phi(x_n)^T \\ | & & | \end{bmatrix}}$$

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

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

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

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

$$= \underset{a}{\text{argmin}} ||y - \mathbf{K}a||_2^2 + \lambda a^\top \mathbf{K} a$$

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

Go to your section handout…

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

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:

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

Setting gradient of L(w) with respect to α equal to 0:

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

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

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.

Mercer's conditions:

- *K* must be symmetric
- *K* must be positive definite

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

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

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:**

$$\begin{aligned}
\hat{\mathbf{Y}} &= \mathbf{X}_{\text{test}} \hat{w} \\
&= \mathbf{X}_{\text{test}} \mathbf{X}_{\text{Train}}^T \hat{\alpha} \\
&= \mathbf{X}_{\text{test}} \mathbf{X}_{\text{train}}^T \left(\mathbf{K}_{\text{train}} + \lambda I\right)^{-1} \mathbf{y}_{\text{train}}
\end{aligned}$$

General Solution for Kernel Ridge

$$\hat{\mathbf{Y}} = \mathbf{K}_{\text{train, test}} \hat{\alpha}$$

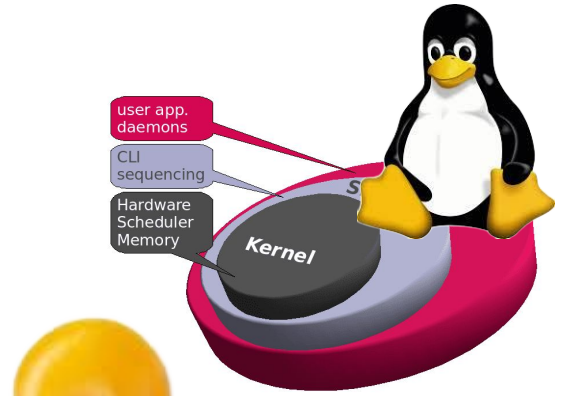
Where $\mathbf{K}_{train,test} = \mathbf{X}_{test}\mathbf{X}_{train}^T$

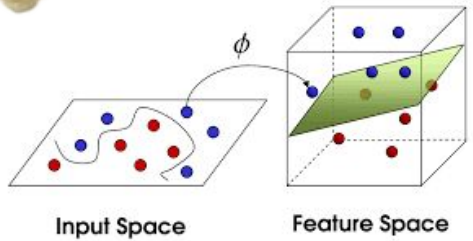
Takeaways

- Kernels are at their core a *computational efficiency trick*
  - Employed when feature mapping is computationally too much
- Kernels must satisfy Mercer's condition
  - Symmetric, positive-definite
- We perform kernelized linear regression which has extra steps

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$Ax = 0$$





What you think of when someone says "Kernel" says a lot about you…

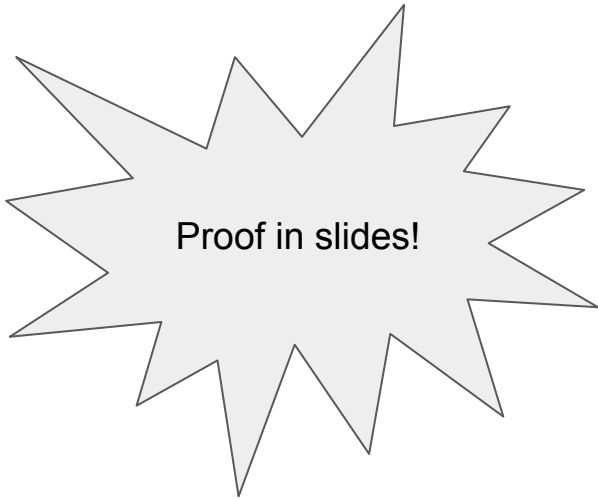
Do this in your own time if you want a tutorial on **how to make a neural net** in PyTorch!

PyTorch Colab

Questions/Chat Time!

Old Slides

How do we use *K*?



Kernelized Linear Regression rewrites how we calculate $\hat{w}$:

$$\hat{w} = \sum_{i=1}^{n} \alpha_i x_i$$

How do we find $\alpha \in \mathbb{R}^n$?

Define the kernel *matrix* $\mathbf{K} \in \mathbb{R}^{n \times n}$ where $\mathbf{K}_{i,j} = K(x_i, x_j) = \phi(x_i)^\top \phi(x_j)$.

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

Now if we want to make a new prediction $\hat{y}$ on a datapoint z, we have to do something extra:

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

RBF Kernels

$$K(x, y) = e^{-\gamma \|x - y\|^2}$$

RBF kernels measure the similarity between the input vectors by calculating their distance in the input feature space.

If x and y are close together, K(x, y) approaches 1

If x and y are further apart, K(x, y) approaches 0

RBF Kernel is infinite-dimensional

Consider a Taylor series expansion

$$e^z = \sum_{n=0}^{\infty} \frac{z^n}{n!} = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \dots$$

$$\boxed{z = -\gamma \|x - y\|^2}$$

Taylor series expansion of RBF Kernel

$$K(x,y) = e^{-\gamma\|x-y\|^2}$$

$$= 1 + (-\gamma\|x-y\|^2) + \frac{(-\gamma\|x-y\|^2)^2}{2!} + \frac{(-\gamma\|x-y\|^2)^3}{3!} + \dots$$

$$= 1 - \gamma\|x-y\|^2 + \frac{\gamma^2\|x-y\|^4}{2!} - \frac{\gamma^3\|x-y\|^6}{3!} + \dots$$

Feature 1

Feature 2

Feature 3

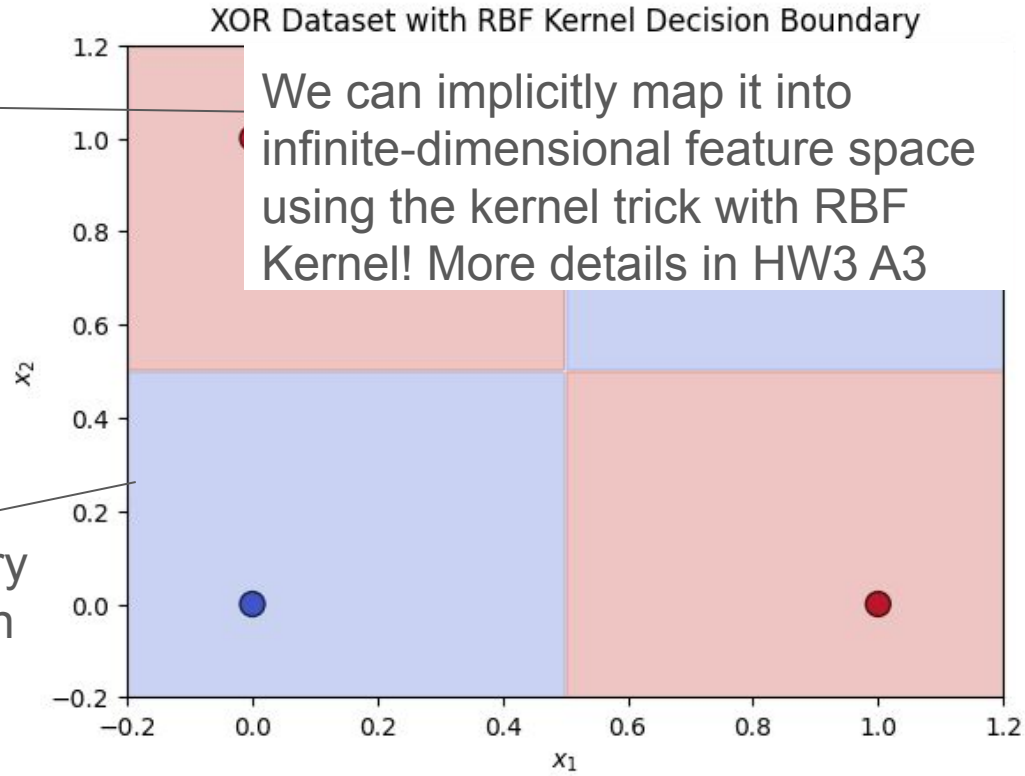
Feature 4

There's infinite number of features!

Why do we care about infinite dimensions?

By mapping the XOR dataset into infinite-dimensional feature space, we can use linear regression to draw a linear decision boundary, correctly separating the XOR dataset in this infinite-dimensional feature space

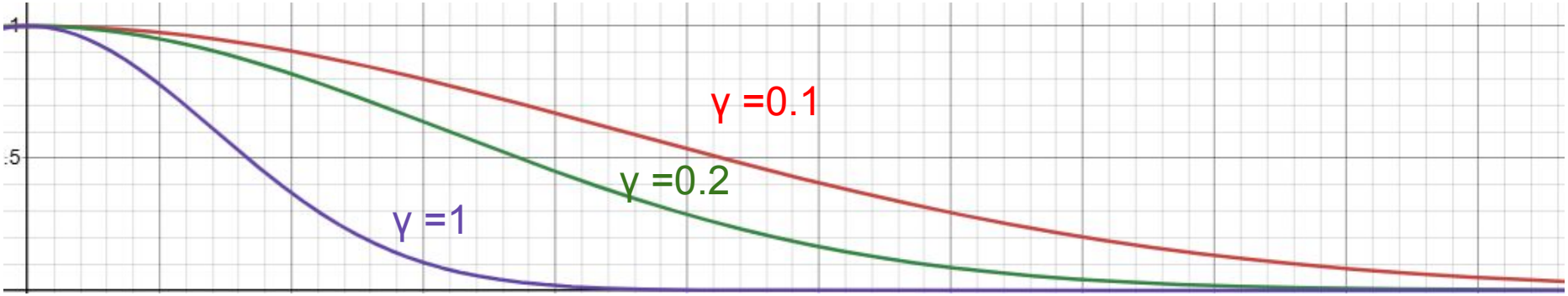
This is what the decision boundary looks like when we project it down to the original feature space

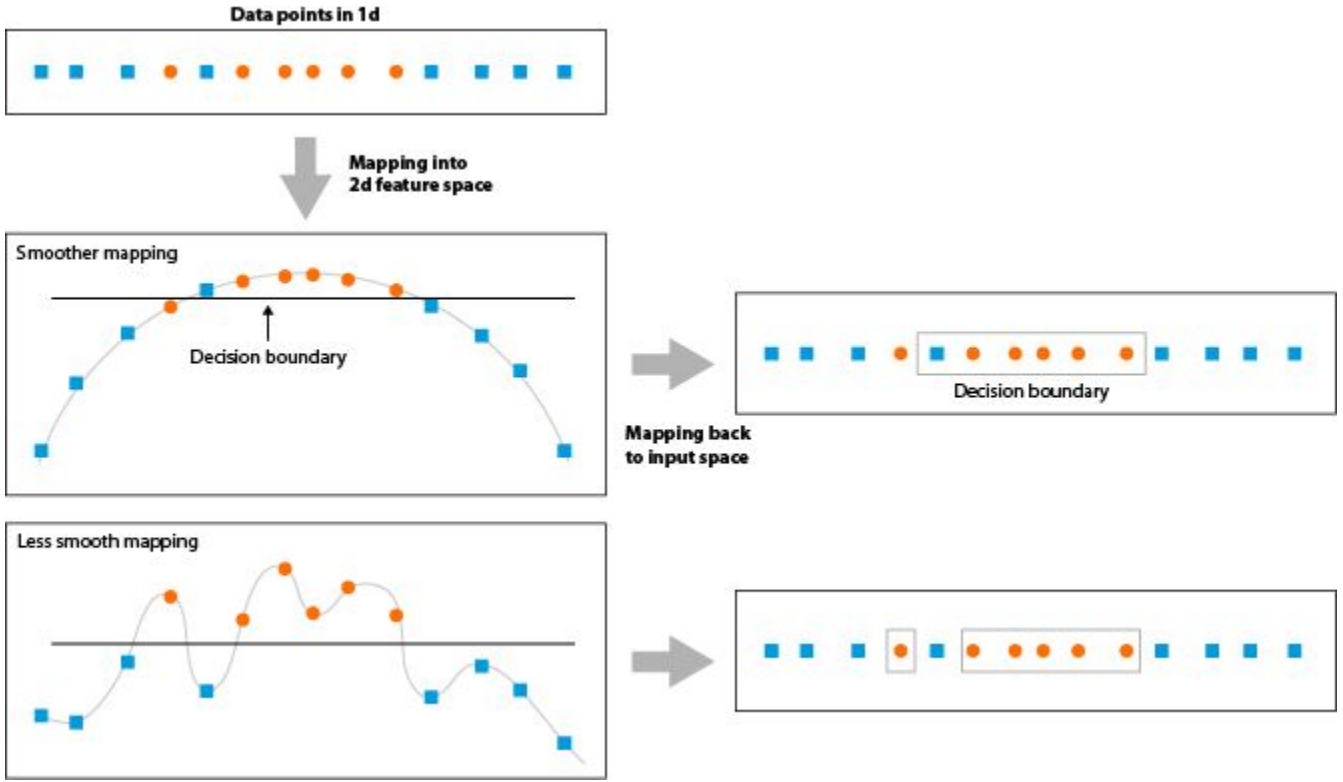


$$K(x, y) = e^{-\gamma \|x-y\|^2}$$

Controls the rate of decay for the similarity score

Smaller gamma results in a smoother mapping





A less smooth mapping can cause overfitting!

You will get more practice with using kernel regression in HW3

- $k_{\text{poly}}(x, z) = (1 + x^\top z)^d$ where $d \in \mathbb{N}$ is a hyperparameter,
- $k_{\text{rbf}}(x, z) = \exp(-\gamma \|x - z\|_2^2)$ where $\gamma > 0$ is a hyperparameter[1].

You will be doing polynomial kernel regression, and RBF kernel regression, just like what we showed you today, but on the same dataset.
You will also experiment with searching for hyperparameters such as gamma.