New Dimensional Space

Key points:

- During training in the new high dimensional space of $\emptyset(x)$ we want to compute γ_i through kernels, without ever computing any $\emptyset(x_i)$ or even w.
- We previously established that $w = \sum_{j=1}^{n} \alpha_j \phi(x_j)$, and $\gamma_i = 2(w^T \phi(x_i) - y_i)$

$$\gamma_i = 2(\sum_{j=1}^{n} \alpha_j K_{ij} - y_i)$$

$$\alpha_i^{t+1} \leftarrow \alpha_i^t - 2s(\sum_{j=1}^{n} \alpha_j^t K_{ij} - y_i)$$

- We have n such updates to do.

General Kernels

Popular Kernel Functions:

- Linear: $K(x, z) = x^T z$ (It's faster to use a kernel matrix if data dimension is high)
- Polynomial: $K(x, z) = (1 + x^T z)^d$
- Radial Basis Function (RBF) (aka Gaussian Kernel): $K(x, z) = e^{\frac{-||x-z||^2}{\sigma^2}}$
  - The RBF kernel is the most commonly used Kernel.
- Exponential Kernel: $K(x, z) = e^{\frac{-||x-z||}{2\sigma^2}}$
- Laplacian Kernel: $K(x, z) = e^{\frac{-|x-z|}{\sigma}}$
- Sigmoid Kernel: $K(x, z) = \tanh(ax^T + c)$

Kernel Functions

What function can be used as a kernel $\boldsymbol{K}(\cdot,\cdot) \to \mathcal{R}$?

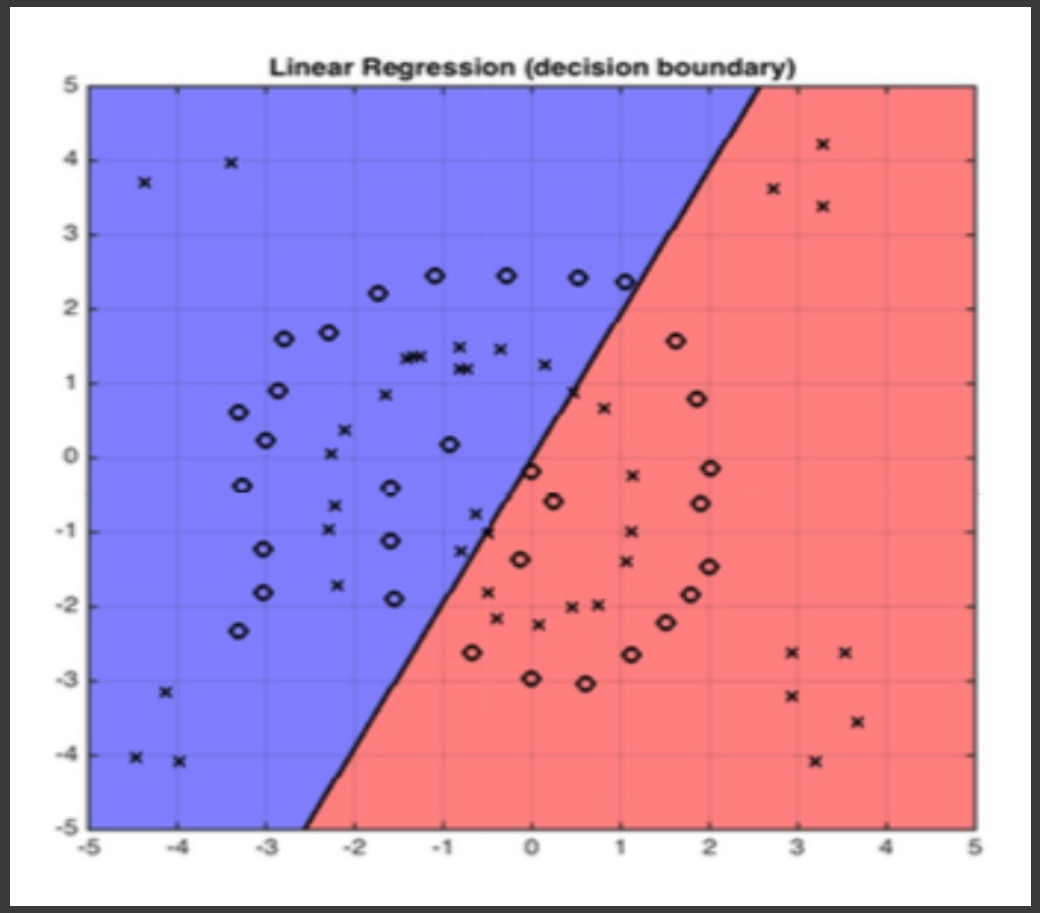
- The matrix $K(x_i, x_j)$ has to correspond to real inner-products after some transformation $x \to \phi(x)$.

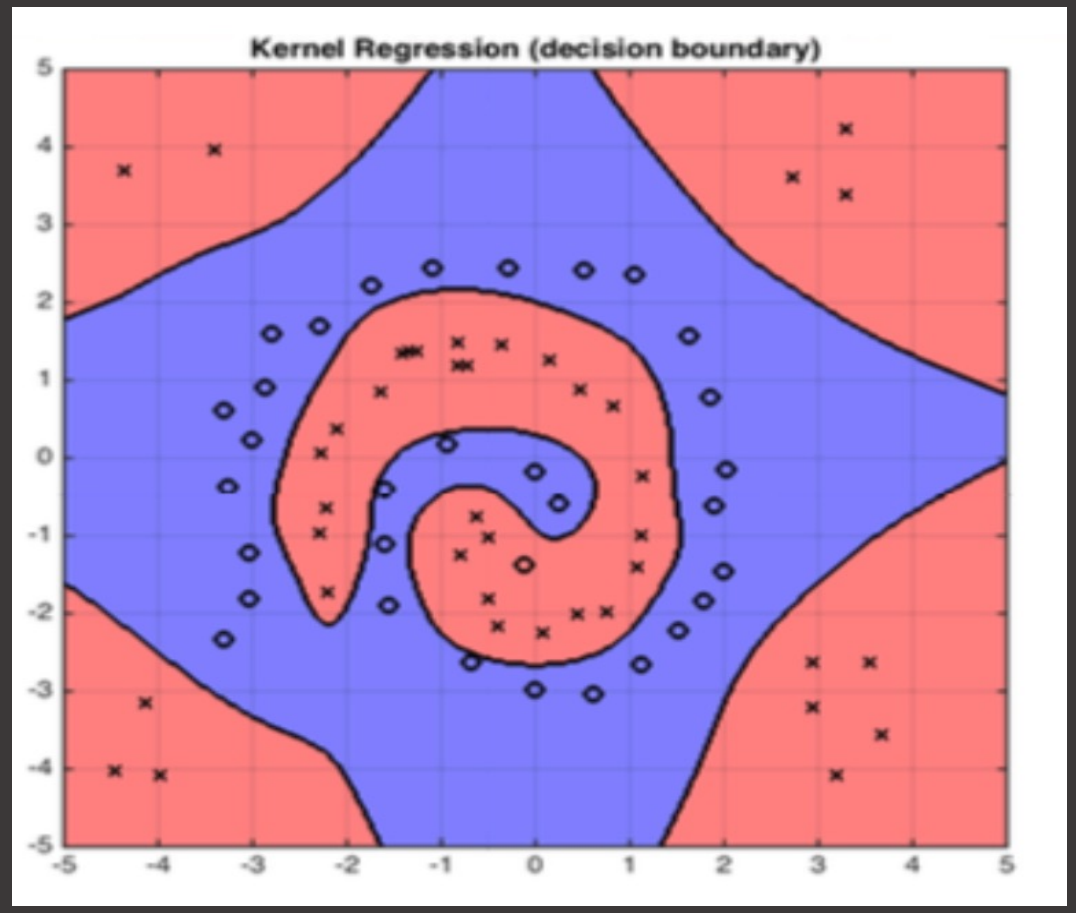
Definition: A matrix $A \in \mathbb{R}^{n\times n}$ is positive semi-definite iff $\forall \mathbf{q} \in \mathbb{R}^n$, $\mathbf{q}^\top A\mathbf{q} \geq 0$.

- The element $K_{ij} = \phi(x_i)^T\phi(x_j)$, so $K = \phi^T\phi$, where $\phi = [\varphi(x_1), \dots, \varphi(x_n)]$ so K is p.s.d. because $q^T K q = (\phi^T q)^2 \geq 0$.

- If any matrix A is p.s.d., it can be decomposed as $A = \phi^T\phi$ for some ϕ.

Kernel Functions





Slides and image credits: Kilian Weinberger

Well-defined Kernels

- The most common kernels: Linear, RBF, Polynomial.
- Kernels built by recursively combining one or more of the following rules are called well-defined kernels:

1. $k(x,z) = x^T z$
2. $k(x,z) = ck_1(x,z)$
3. $k(x,z) = k_1(x,z) + k_2(x,z)$
4. $k(x,z) = g(k(x,z))$
5. $k(x,z) = k_1(x,z)\, k_2(x,z)$
6. $k(x,z) = f(x)k_1(x,z)f(z)$
7. $k(x,z) = e^{k_1(x,z)}$
8. $k(x,z) = x^T A z$

- Linear Kernel
- $c \geq 0$
- k_1, k_2 are well-defined kernels
- g is a polynomial function with positive coefficients
- k_1, k_2 are well-defined kernels
- f is any function
- k_1 is well-defined kernel
- $A \succcurlyeq 0$ is positive semi-definite

Well-defined Kernels

Theorem. *The RBF kernel* $k(x,z) = e^{\frac{-(x-z)^2}{\sigma^2}}$ *is a well-defined kernel matrix.*

- `Prove it!`

- $k_1(x,z) = x^T z$ well defined by rule 1
- $k_2(x,z) = \frac{2}{\sigma^2} k_1(x,z) = \frac{2}{\sigma^2} x^T z$ well defined by rule 2
- $k_3(x,z) = e^{k_2(x,z)} = e^{\frac{2x^T z}{\sigma^2}}$ well defined by rule 7
- $k_4(x,z) = e^{\frac{-x^T x}{\sigma^2}} k_3(x,z) e^{\frac{-z^T z}{\sigma^2}} = e^{\frac{-x^T x}{\sigma^2}} e^{\frac{2x^T z}{\sigma^2}} e^{\frac{z^T z}{\sigma^2}} = e^{\frac{-x^T x + 2x^T z - z^T z}{\sigma^2}} = e^{\frac{-(x-z)^2}{\sigma^2}} = k_{RBF}(x,z)$

well defined by rule 6 with $f(x) = e^{\frac{-x^T x}{\sigma^2}}$

Kernelized an algorithm

- (In practice) an algorithm can be kernelized in **three** steps:
  - Prove that the solution lies in the span of the training points (i.e. $w = \sum_{i=1}^{n} \alpha_i x_i$ for some α_i)
  - Rewrite the algorithm and the classifier so that all training or testing inputs x_i are ***only accessed*** in inner-products with other Inputs, e.g. $x_i^T x_j$.
  - Define a kernel function and substitute $k(x_i, x_j)$ with $x_i^T x_j$.

Kernelize Linear Regression

- For linear regression, we try to minimize the squared loss:

$$\min_{w} \sum_{i=1}^{n} (w^T x_i - y_i)^2$$

  - The hyperplane is defined by w.
  - When testing, simply $h(x) = w^T x$.
  - Since $X = [x_1, \dots, x_n]$ and $y = [y_1, \dots, y_n]^T$, the closed form solution based on OLS can be written in closed form: $w = (XX^T)^{-1}Xy$

Kernelize Linear Regression

- Recap from last lecture that the solution w is a linear combination of training inputs

$$w = \sum_{i=1}^{n} \alpha_i x_i = X\vec{\alpha}$$

- α must always exist if the loss is minimized by gradient descent and the initial vector is set to $\vec{0}$.

- During testing, a test point z is only accessed through inner-products with training inputs

$$h(z) = w^T z = \sum_{i=1}^{n} \alpha_i x_i^T z$$

substituting $k(x, z)$ for any inner-product x^Tz

Kernelize Linear Regression

Theorem. Kernelized ordinary least squares has the solution $\vec{\alpha} = K^{-1}y$

- Proof

$X\vec{\alpha} = w = (XX^T)^{-1}Xy$ | least square results

$(X^TX)(X^TX)\vec{\alpha} = X^T(XX^T(XX^T)^{-1})Xy$ | multiply from left by X^TXX^T

$K^2\vec{\alpha} = Ky$ | substitute $K = X^TX$

$\vec{\alpha} = K^{-1}y$ | multiply from left by $(K^{-1})^2$

Kernel regression can be extended to the kernelized version of Ridge regression. The solution is $\vec{\alpha} = (K + \tau^2 I)^{-1}y$

In practice a small value of $\tau^2 > 0$ increases stability, especially if K is invertible.

Kernelize SVM

The **primal** SVM is a quadratic programming problem:

$$\min_{w,h} \mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{n}\xi_i$$

$$\textbf{s.t. } \forall i,\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

The ***dual*** form (not required for understanding)

$$\min_{\alpha_1,\ldots,\alpha_n} \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j K_{ij} - \sum_{i=1}^{n}\alpha_i$$

$$\textbf{s.t. } 0 \leq \alpha_i \leq C$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0$$

Kernelize SVM

- We have $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \phi(\mathbf{x}_i)$

We never compute this

- And $h(\mathbf{x}) = sign\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x_i}, \mathbf{x}) + b\right)$
- For the primal formulation, only support vectors satisfy the constraint with equality $y_i(\mathbf{w}^T \phi(x_i) + b) = 1.$
- In the dual formulation, the support vectors will have $\alpha_i > 0$ and other training inputs have $\alpha_i = 0.$
- During test, only need to compute the sum in $h(x)$ over the support vectors and all the others with $\alpha_i = 0$can be <u>*discarded*</u> after training.

Kernelize SVM

- Solving b for classification:
  - For support vectors, we have $\alpha_i > 0$. Then

$$y_i(\mathbf{w}^T\phi(x_i) + b) = 1 \qquad y_i(\mathbf{w}^T\phi(x_i) + b) = 1$$

$$y_i\left(\sum_j y_j\alpha_j k(\mathbf{x}_j, \mathbf{x}_i) + b\right) = 1 \quad y_i\left(\sum_j y_j\alpha_j k(\mathbf{x}_j, \mathbf{x}_i) + b\right) = 1$$

$$y_i - \sum_j y_j\alpha_j k(\mathbf{x}_j, \mathbf{x}_i) = b \quad y_i - \sum_j y_j\alpha_j k(\mathbf{x}_j, \mathbf{x}_i) = b$$

Such that we can solve for b from the support vectors.

Kernel SVM vs. nearest neighbor

- For binary classification problem ($y_i \in \{+1, -1\}$), we can write the decision function for a test point z as

$$h(\mathbf{z}) = sign\left(\sum_{i=1}^{n} y_i \delta^{nn}(\mathbf{x}_i, \mathbf{z})\right)$$

where $\delta^{nn}(\mathbf{z}, \mathbf{x}_i) \in \{0, 1\}$ with $\delta^{nn}(\mathbf{z}, \mathbf{x}_i) = 1$ only if x_i is one of the k nearest neighbors of test point z.

- The SVM decision function

$$h(\mathbf{z}) = sign\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{z}) + b\right)$$

is actually a soft version of KNN.

Kernel Method

- Once we construct the kernel space, rather than run SVM on original x_i, we can run it on $\Phi(x_i)$.
  - corresponding to find non-linear separator in input space
- What if $\Phi(x_i)$ really big and how can I construct it?
  - use Kernels to compute it implicitly !

Kernel Method

- Find kernel *K* such that

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$$

- Computing $K(x_i, x_j)$ should be efficient, much more so than computing $\Phi(x_i)$ and $\Phi(x_j)$. So we don't need to compute them ☺

- Use $K(x_i, x_j)$ in SVM algorithm (or other inner product place) rather than $\langle x_i, x_j \rangle$

Polynomial Kernel

- Let $\mathbf{x}_i = [x_{i1}, x_{i2}]$ and $\mathbf{x}_j = [x_{j1}, x_{j2}]$
  Consider the following function:

$$
\begin{aligned}
K(\mathbf{x}_i, \mathbf{x}_j) &= \langle \mathbf{x}_i, \mathbf{x}_j \rangle^2 \\
&= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 \\
&= (x_{i1}^2 x_{j1}^2 + x_{i2}^2 x_{j2}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2}) \\
&= \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle
\end{aligned}
$$

where

$$
\begin{aligned}
\Phi(\mathbf{x}_i) &= [x_{i1}^2, x_{i2}^2, \sqrt{2}x_{i1}x_{i2}] \\
\Phi(\mathbf{x}_j) &= [x_{j1}^2, x_{j2}^2, \sqrt{2}x_{j1}x_{j2}]
\end{aligned}
$$

Kernelization (one example)

$$\phi\phi^T = K = \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & \vdots & \phi(x_1)^T\phi(x_n) \\ \phi(x_2)^T\phi(x_1) & \phi(x_2)^T\phi(x_2) & \vdots & \phi(x_2)^T\phi(x_n) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(x_m)^T\phi(x_1) & \phi(x_m)^T\phi(x_2) & & \phi(x_m)^T\phi(x_n) \end{bmatrix} = \begin{bmatrix} k(x_1,x_1) & k(x_1,x_1) & \vdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \vdots & k(x_2,x_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_m,x_1) & k(x_m,x_2) & & k(x_m,x_n) \end{bmatrix}$$

Suppose $x = [x_1, x_2]^T$, $\emptyset: x \to \emptyset(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$

Don't get confused here, $\emptyset(x_1)$ here x_1 is a training data point; for $[x_1^2, \sqrt{2}x_1x_2, x_2^2]$, x_1 is the first dimension.

$$\emptyset(x)^T\emptyset(z) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}^T \begin{bmatrix} z_1^2 \\ \sqrt{2}z_1z_2 \\ z_2^2 \end{bmatrix} = x_1^2z_1^2 + 2x_1x_2z_1z_2 + x_2^2z_2^2 = (x_1z_1 + x_2z_2)^2$$

Polynomial Kernel

- Defined as $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i, \mathbf{x}_j \rangle^d$
  - $\Phi(\mathrm{x})$ contains all monomials of degree d
  - 10^{10} monomials of degree 5
  - Never explicitly compute $\Phi(\mathrm{x})$
- Variation $K(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + 1)^d$
  - adds all lower-order monomials (degrees $1, \ldots, d$)

Kernel Trick

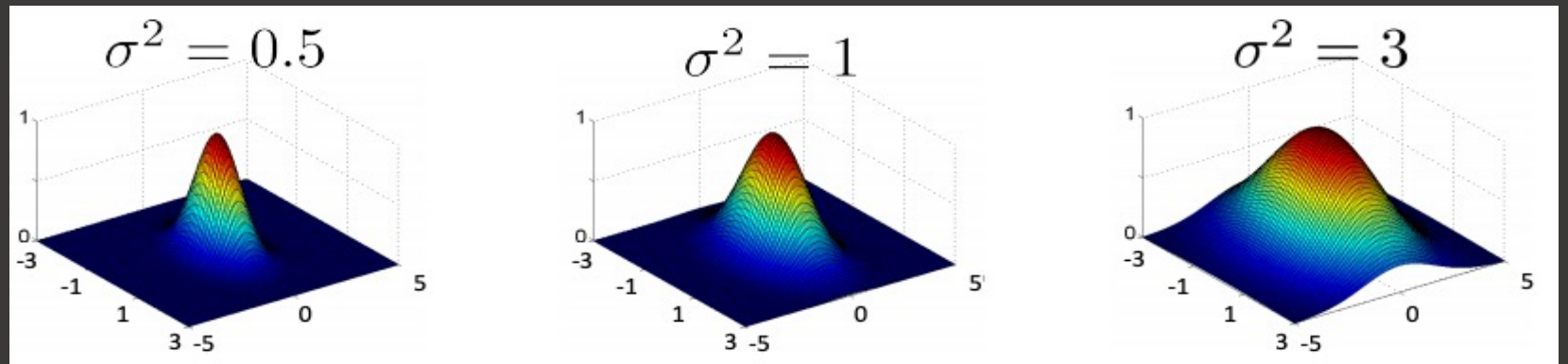
- Given an algorithm which is formulated in terms of a positive definite kernel K, one can construct an alternative algorithm by replacing K with another positive definite kernel K'

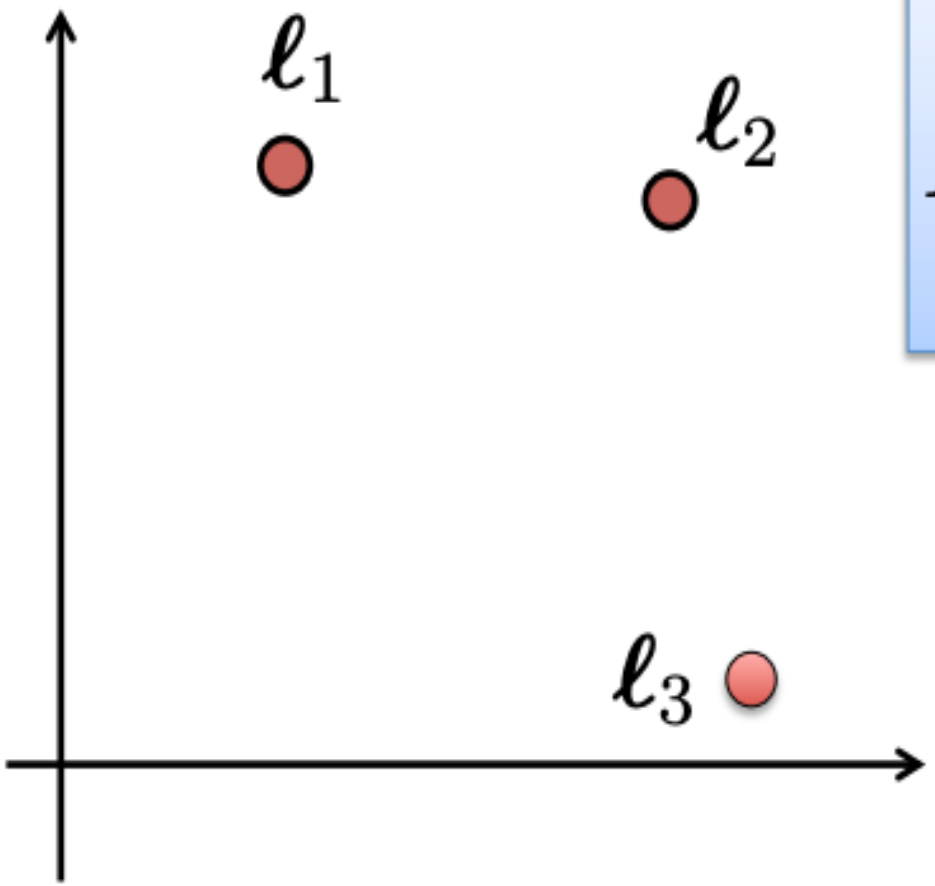
Gaussian Kernel

- Also called Radial Basis Function (RBF) kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

  - Has value 1 when $\mathbf{x}_i = \mathbf{x}_j$
  - Value falls off to 0 with increasing distance
  - Need to do feature scaling **before** using Gaussian Kernel





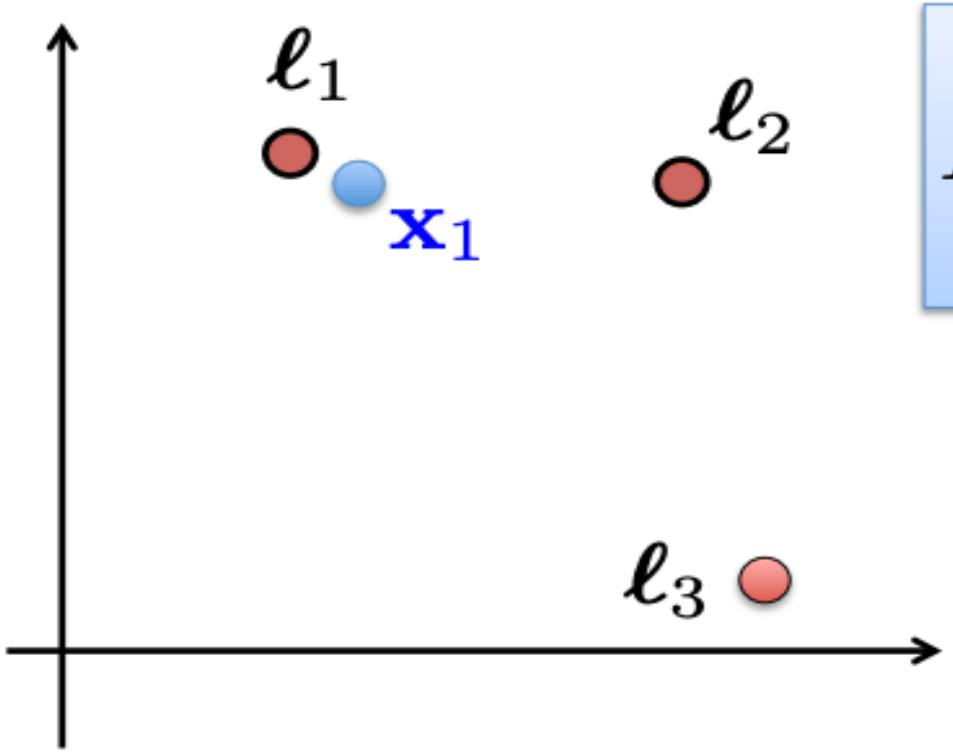
$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \boldsymbol{\ell}_1) + \theta_2 K(\mathbf{x}, \boldsymbol{\ell}_2) + \theta_3 K(\mathbf{x}, \boldsymbol{\ell}_3) \geq 0$

Example based on Andrew Ng



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

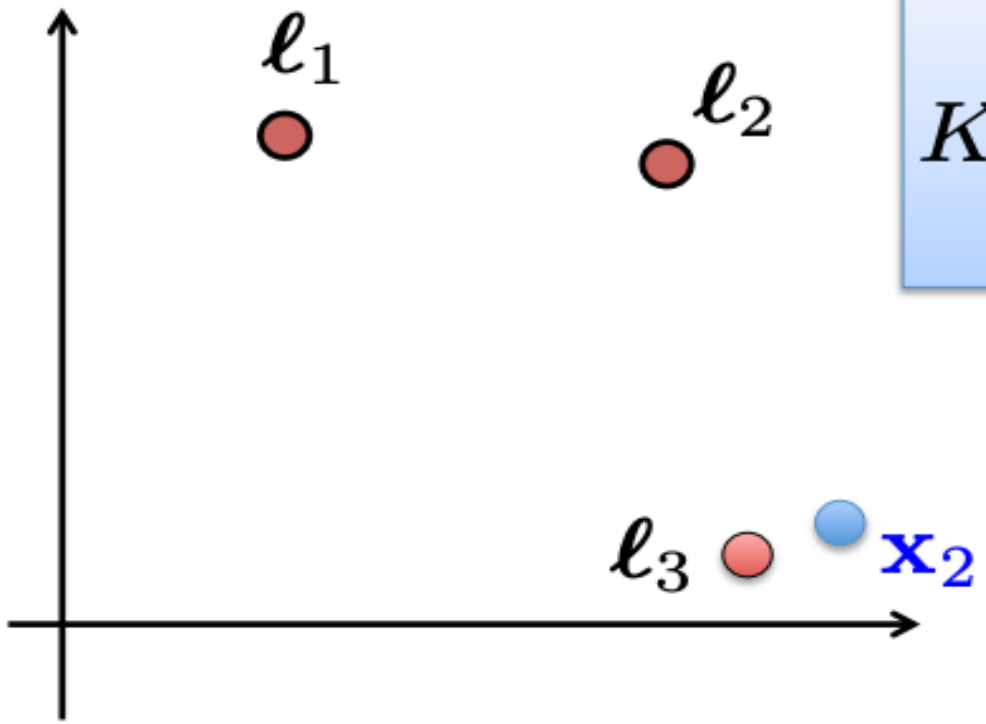
$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \boldsymbol{\ell}_1) + \theta_2 K(\mathbf{x}, \boldsymbol{\ell}_2) + \theta_3 K(\mathbf{x}, \boldsymbol{\ell}_3) \geq 0$

- For $\mathbf{x}_1$, we have $K(\mathbf{x}_1, \ell_1) \approx 1$, other similarities ≈ 0

$$\begin{aligned} &\theta_0 + \theta_1(1) + \theta_2(0) + \theta_3(0) \\ &= -0.5 + 1(1) + 1(0) + 0(0) \\ &= 0.5 \geq 0 \text{ , so predict +1} \end{aligned}$$

Example based on Andrew Ng



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

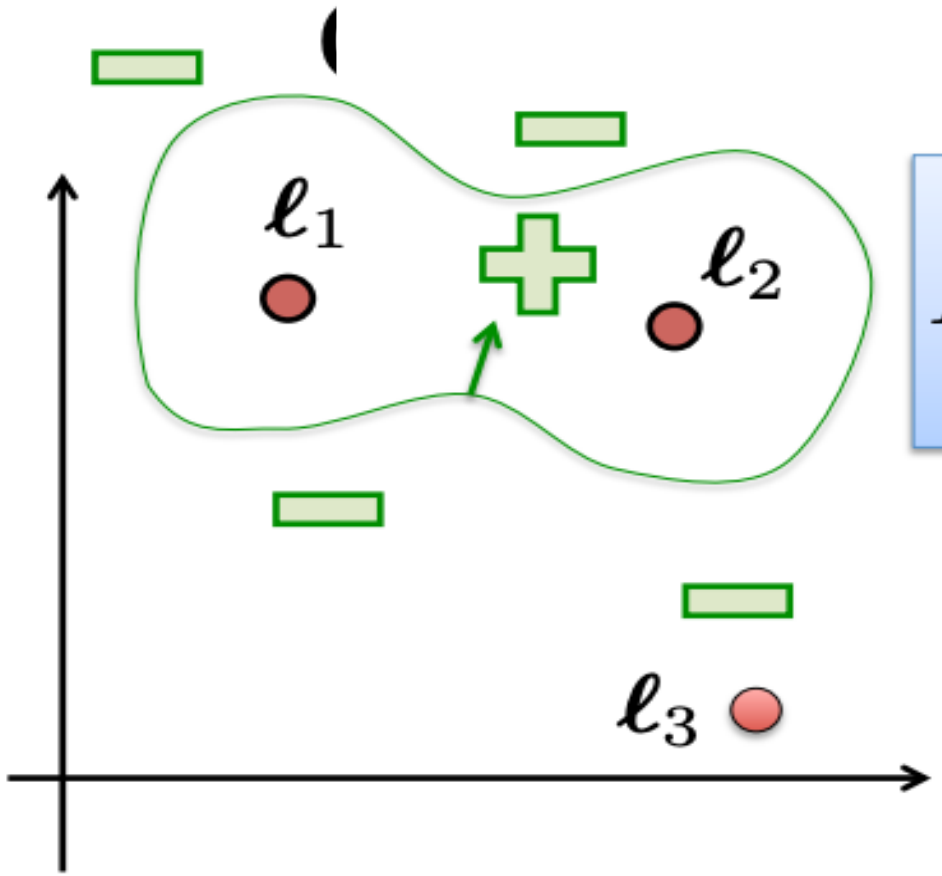
$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \boldsymbol{\ell}_1) + \theta_2 K(\mathbf{x}, \boldsymbol{\ell}_2) + \theta_3 K(\mathbf{x}, \boldsymbol{\ell}_3) \geq 0$

- For $\mathbf{x}_2$, we have $K(\mathbf{x}_2, \ell_3) \approx 1$, other similarities ≈ 0

$$\begin{aligned} &\theta_0 + \theta_1(0) + \theta_2(0) + \theta_3(1) \\ &= -0.5 + 1(0) + 1(0) + 0(1) \\ &= -0.5 < 0 \text{, so predict -1} \end{aligned}$$

Example based on Andrew Ng



$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\sigma^2}\right)$$

Imagine we've learned that:

$$\boldsymbol{\theta} = [-0.5, 1, 1, 0]$$

Predict +1 if $\theta_0 + \theta_1 K(\mathbf{x}, \boldsymbol{\ell}_1) + \theta_2 K(\mathbf{x}, \boldsymbol{\ell}_2) + \theta_3 K(\mathbf{x}, \boldsymbol{\ell}_3) \geq 0$

Rough sketch of decision surface

Example based on Andrew Ng

Other Kernels

- Sigmoid Kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha \mathbf{x}_i^T \mathbf{x}_j + c)$
  - Neural networks use sigmoid as activation function
  - SVM with a sigmoid kernel is equivalent to 2-layer Perceptron
- Cosine Similarity Kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$
  - Common choice for text analysis
  - L2 norm projects vectors onto the unit sphere; their dot product is the cosine of the angle between the vectors.

Other Kernels

- Chi-squared Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\gamma \sum_k \frac{(x_{ik} - x_{jk})^2}{x_{ik} + x_{jk}}\right)$$

  - Chi-squared measures distance between probability distributions
  - Data is assumed to be non-negative, often with with $L1$ norm of 1.
- String kernel; Tree kernel; Graph kernel,…

Other Kernels

- Chi-squared Kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = exp\left(-\gamma \sum_k \frac{(x_{ik} - x_{jk})^2}{x_{ik} + x_{jk}}\right)$$

  - Chi-squared measures distance between probability distributions
  - Data is assumed to be non-negative, often with with $L1$ norm of 1.
- String kernel; Tree kernel; Graph kernel,…

Some Math about Kernels

- What does it mean to be a kernel?

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \text{ for some } \Phi$$

- What does it take to be a kernel?
  - The Gram matrix $G_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - Symmetric matrix
  - Positive semi-definite matrix
    $z^T G z \geq 0$ for every non-zero vector $z \in \mathbb{R}^d$