Machine Learning: Yoshinari Fujinuma

University of Colorado Boulder

LECTURE 19

Slides adapted from Jordan Boyd-Graber, Chris Ketelsen

Logistics

- Homework 3 is due today
- Homework 4 is available
- Project proposal is due on Friday

Overview

Kernels

Examples

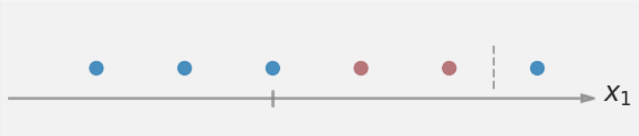
Outline

Kernels

Examples

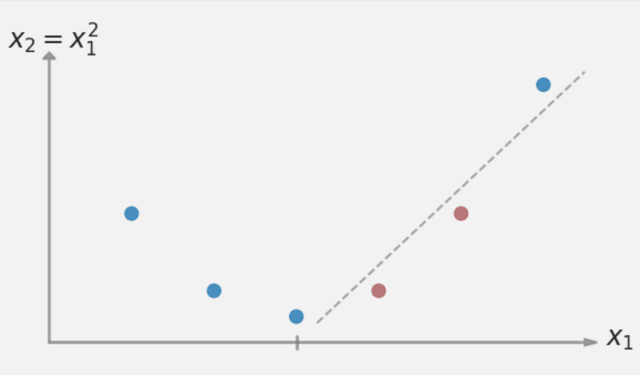
Can you solve this with linear separator?

What can we do if the data is clearly not linearly separable?



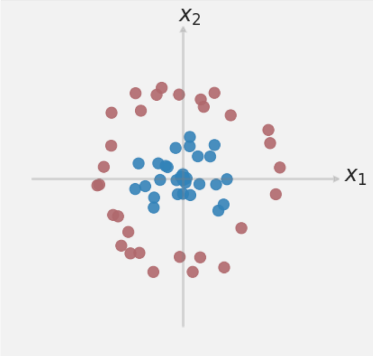
Can you solve this with linear separator?

Add a dimension.



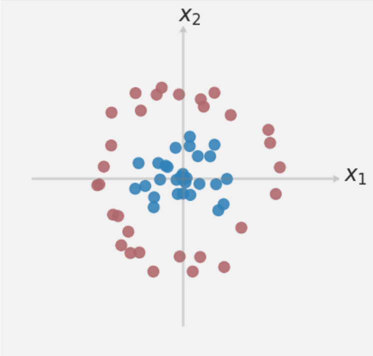
What about this?

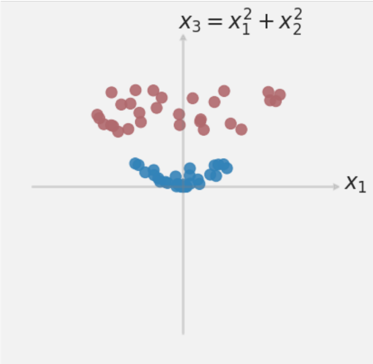
Definitely not separable in two dimensions.



What about this?

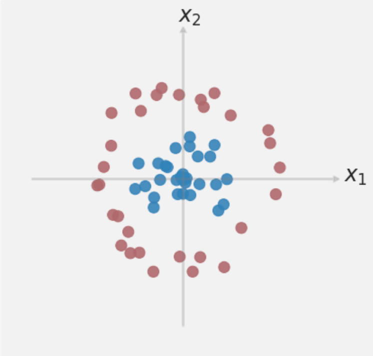
Definitely not separable in two dimensions.
But in three dimensions, it becomes easily separable.

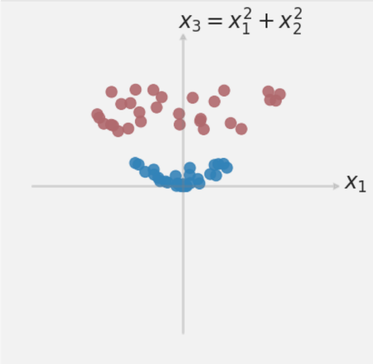




Derived features

We started with the original feature vector, $\boldsymbol{x} = (x_1, x_2)$,
and we created a new derived feature vector, $\phi(\boldsymbol{x}) = (x_1, x_2, x_1^2 + x_2^2)$.





What's special about SVMs?

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i}^T \boldsymbol{x_j})$$

What's special about SVMs?

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i}^T \boldsymbol{x_j})$$

- This dot product is basically just how much $\boldsymbol{x}_i$ looks like $\boldsymbol{x}_j$. Can we generalize that?
- Kernels!

What's special about SVMs?

Soft-margin SVM

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i}^T \boldsymbol{x_j})$$

Soft-margin SVM with feature mapping ϕ

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\phi(\boldsymbol{x_i})^T \phi(\boldsymbol{x_j}))$$

Soft-margin SVM with a kernel K

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

Why Do We Need Kernels?

Kernel is a generalization of dot product

- Dot product tells us about the **similarity** of the two vectors
- Kernel can be viewed as a similarity measure but with non-linear combination of features

$$K(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$$

So why do we want kernels instead of just defining a projection ϕ?

Why Do We Need Kernels?

Assume d-dimensional feature vectors $\boldsymbol{x}$ and $\boldsymbol{z}$
Example of a kernel:

$$
\begin{aligned}
K(\boldsymbol{x},\boldsymbol{z}) &= (\boldsymbol{x}^T\boldsymbol{z})^2 \\
&= (\textstyle\sum_{i=1}^{p} x_i z_i)^2 \\
&= (\textstyle\sum_{i=1}^{p} x_i z_i)(\textstyle\sum_{j=1}^{p} x_j z_j) \\
&= \textstyle\sum_{i=1}^{p}\sum_{j=1}^{p} x_i z_i x_j z_j \\
&= \textstyle\sum_{i=1}^{p}\sum_{j=1}^{p} x_i x_j z_i z_j \\
&= \phi(\boldsymbol{x})^T\phi(\boldsymbol{z})
\end{aligned}
$$

Why Do We Need Kernels?

Suppose $d = 3$. Then $\boldsymbol{x} = (x_1, x_2, x_3)$ and $\boldsymbol{z} = (z_1, z_2, z_3)$.
Using $K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z})^2 = \sum_{i=1}^{p} \sum_{j=1}^{p} x_i x_j z_i z_j = \phi(\boldsymbol{x})^T \phi(\boldsymbol{z})$, where $\phi(\boldsymbol{x})$ is

$$\phi(\boldsymbol{x}) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

So we need to keep track of $d^2 = 9$ features if we explicitly use ϕ.

Why Do We Need Kernels?

If we use the kernel $K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z})^2$ where $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^d$, then

- computation of ϕ requires $O(p^2)$ in time and space
- computation of $(\boldsymbol{x}^T \boldsymbol{z})^2$ requires $O(p)$ in time

Evaluating using kernels is lot cheaper especially when d is large.

What's a kernel?

- A function $K : \mathcal{X} \times \mathcal{X} \mapsto \mathrm{R}$ is a kernel over $\mathcal{X}$.
- This is equivalent to taking the dot product $\phi(\boldsymbol{x})^T \phi(\boldsymbol{x}')$ for some mapping
- Mercer's Theorem: So long as the function is continuous and symmetric, then K admits an expansion of the form

$$K(\boldsymbol{x}, \boldsymbol{x}') = \sum_n a_n \phi_n(\boldsymbol{x}) \phi_n(\boldsymbol{x}')$$

- The computational cost is just in computing the kernel

Kernel Matrix

The important property of the kernel matrix $\boldsymbol{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ is symmetric positive semidefinite.

$$\boldsymbol{K}^T = \boldsymbol{K} \text{ (symmetric)}$$

$$\forall \boldsymbol{x}, \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0 \text{ (positive semidefinite)}$$

Also known as Gram matrix.

Example of Kernels

Polynomial Kernel

$$K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^d$$

where c is a constant, d is the degree of polynomial

(Gaussian) Radial Basis Kernel (RBF Kernel)

$$K(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(-\gamma \left\|\boldsymbol{x}' - \boldsymbol{x}\right\|^2\right)$$

where γ is a hyperparameter.

- if $\boldsymbol{x} = \boldsymbol{x}'$, then $K(\boldsymbol{x}, \boldsymbol{x}') = 1$
- if $\boldsymbol{x}$ is very different from $\boldsymbol{x}'$, then $K(\boldsymbol{x}, \boldsymbol{x}') \approx 0$

How does it affect optimization

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j (\boldsymbol{x_i}^T \boldsymbol{x_j}) \qquad \max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{i=1}^{m} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x_i}, \boldsymbol{x_j})$$

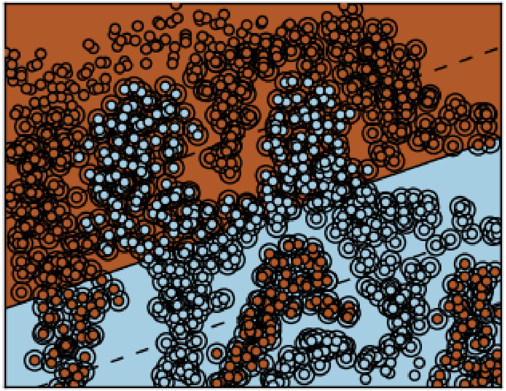
- Replace all dot product with kernel evaluations $K(x_1, x_2)$
- Makes computation more expensive, overall structure is the same

Outline

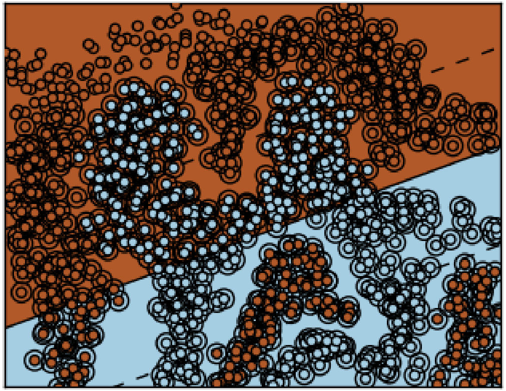
Kernels

Examples

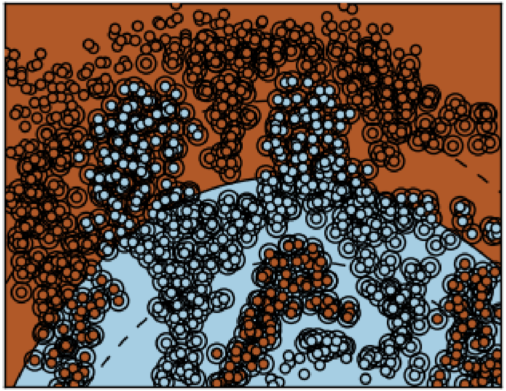
Linear Decision Boundary Doesn't Work



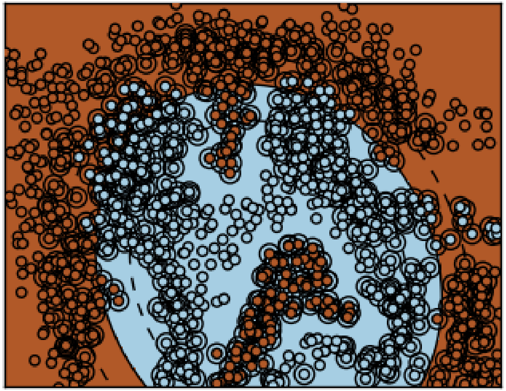
Polynomial Kernel $d = 1, c = 5$



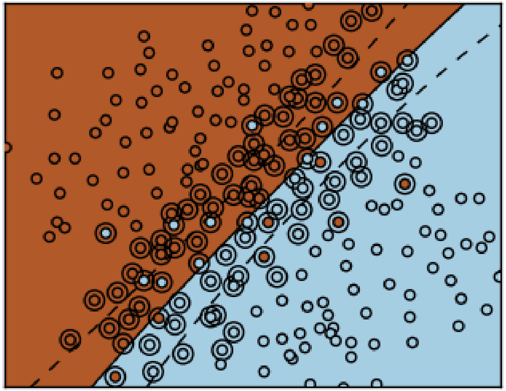
Polynomial Kernel $d = 2, c = 5$



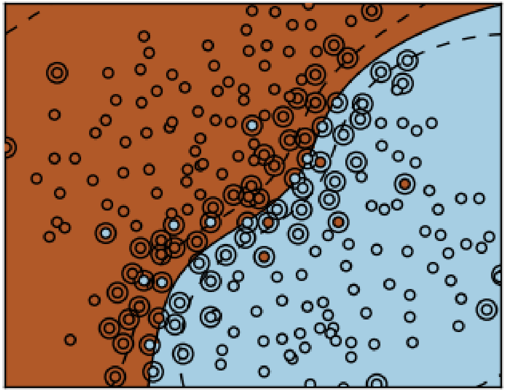
Polynomial Kernel $d = 3, c = 5$



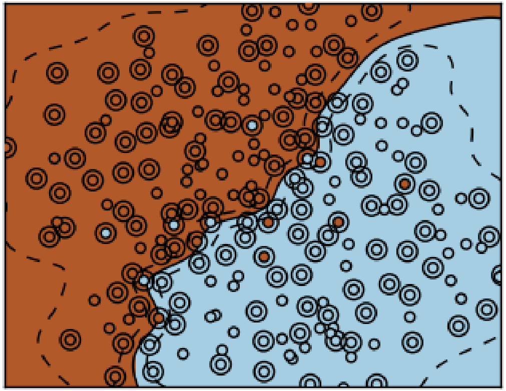
RBF Kernel $\gamma = 1$



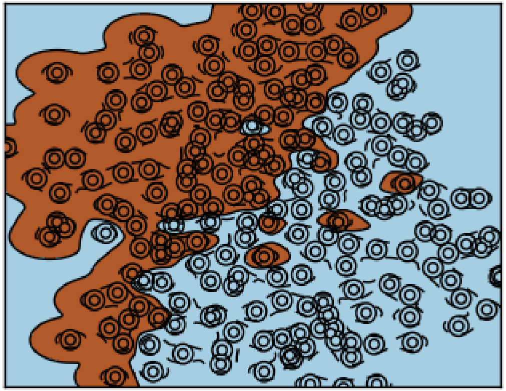
RBF Kernel $\gamma = 10$



RBF Kernel $\gamma = 100$



RBF Kernel $\gamma = 1000$



Recap

- Kernels: applicable to wide range of data, inner product trick keeps method simple