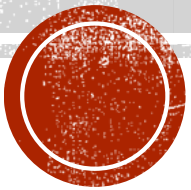
KERNELS



CS 412 Introduction to Machine Learning

Prof. Zheleva

February 22, 2017

Reading Assignment: CIML: 11.1-11.4

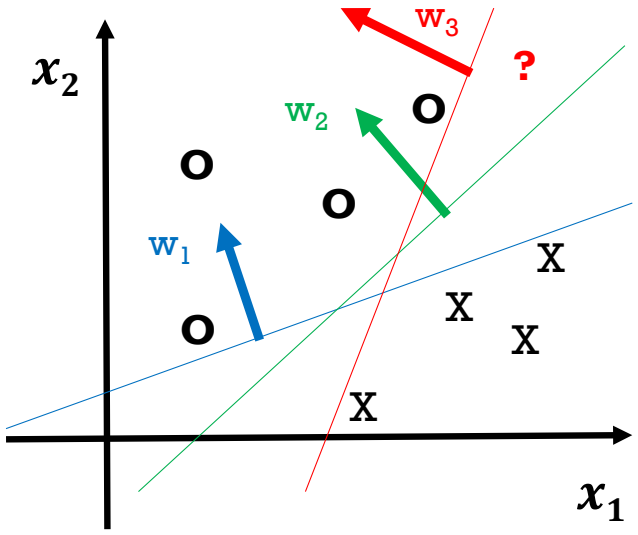
LAST TIME: LINEAR MODELS

- Subgradients
  - When functions are not differentiable
- Closed form solutions to optimization problems
- (Linear) Support Vector Machines
  - Computing and optimizing the margin

LINEAR MODELS

- A linear classifier is a hyperplane that separates positive from negative examples (**w**,b)
  - The prediction is a linear combination of feature values x
  - Examples: perceptron, linear SVM
- At test time, we check on what side of the hyperplane examples fall

$$\hat{y} = sign(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b})$$



LEARNING A LINEAR CLASSIFIER AS AN OPTIMIZATION PROBLEM

- Structural risk minimization framework
  - Tradeoff between low training error and a solution that is simple

$$\min_{\boldsymbol{w},b} L(\boldsymbol{w},b) = \min_{\boldsymbol{w},b}\left(\sum_{n=1}^{N} l(y_n, \boldsymbol{w}\cdot x_n + b) + \lambda R(\boldsymbol{w},b)\right)$$

Objective function

Loss function
Measures how well classifier fits training data

Regularizing function
Helps with avoiding overfitting
Penalizes complex functions
λ is a hyperparameter

Variables over which we are optimizing the objective function

APPROXIMATING THE 0/1 LOSS WITH CONVEX SURROGATE LOSS FUNCTIONS

- Examples (with b=0)

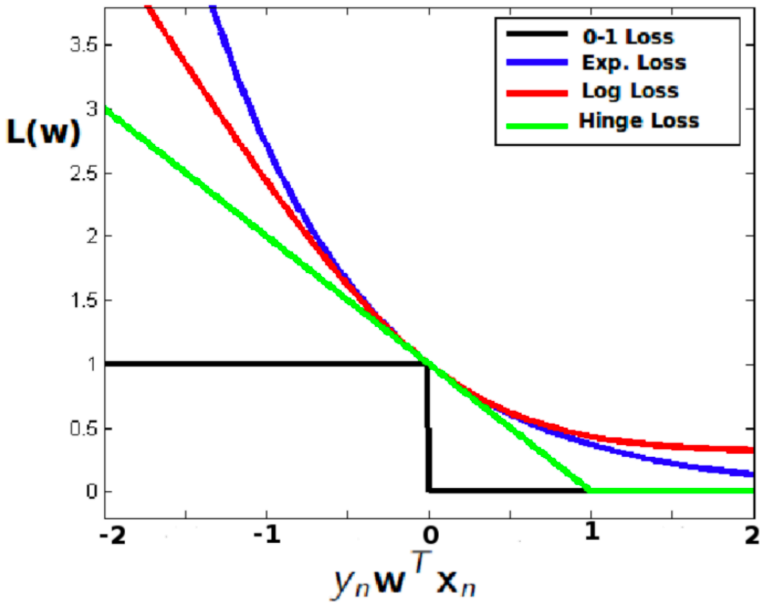
$$\text{Zero/one:} \quad \ell^{(0/1)}(y,\hat{y}) = \mathbf{1}[y\hat{y} \leq 0]$$

$$\text{Hinge:} \quad \ell^{(\text{hin})}(y,\hat{y}) = \max\{0, 1 - y\hat{y}\}$$

$$\text{Logistic:} \quad \ell^{(\log)}(y,\hat{y}) = \frac{1}{\log 2} \log\left(1 + \exp[-y\hat{y}]\right)$$

$$\text{Exponential:} \quad \ell^{(\exp)}(y,\hat{y}) = \exp[-y\hat{y}]$$

$$\text{Squared:} \quad \ell^{(\text{sqr})}(y,\hat{y}) = (y - \hat{y})^2$$



- All are convex upper bounds on 0/1 loss
- Some smooth, some more sensitive to outliers

OPTIMIZATION AS MATRIX OPERATIONS

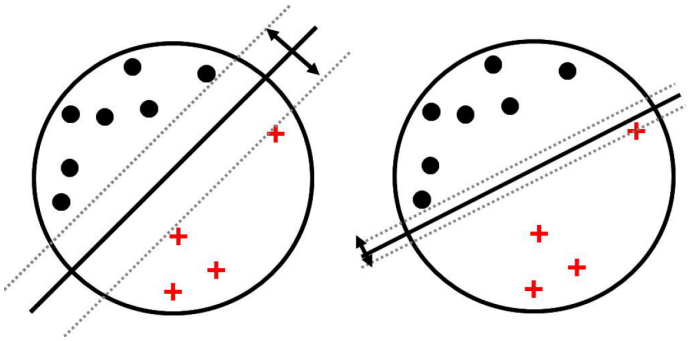
$$\hat{\boldsymbol{Y}} = [\boldsymbol{X}\boldsymbol{w}]_n = \sum_d \boldsymbol{X}_{n,d} w_d$$

$$\underbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,D} \\ x_{2,1} & x_{2,2} & \dots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,D} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} \sum_d x_{1,d} w_d \\ \sum_d x_{2,d} w_d \\ \vdots \\ \sum_d x_{N,d} w_d \end{bmatrix}}_{\hat{\mathbf{Y}}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\hat{Y}}$$

HARD-MARGIN VS SOFT-MARGIN SVM

- Hard-margin SVM: if *data not linearly separable*, feasible region is empty
  - No parameters can satisfy the constraints
- Soft-margin SVM: use slack parameters

$$\min_{w,b,\xi} \underbrace{\frac{1}{\gamma(w,b)}}_{\text{Large margin}} + C \underbrace{\sum_n \xi_n}_{\text{Small slack:}}$$



Amount you need to pay, so a point is not misclassified

$$\text{Subject to } y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 - \xi_n \quad (\forall n)$$
$$\xi_n \geq 0 \quad (\forall n)$$

SVM OPTIMIZATION

- Recovering slacks when given **w** and b

$$\xi_n = \begin{cases} 0 & if \;\; y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) \geq 1 \\ 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b) & otherwise \end{cases}$$

- SVM as unconstrained optimization problem

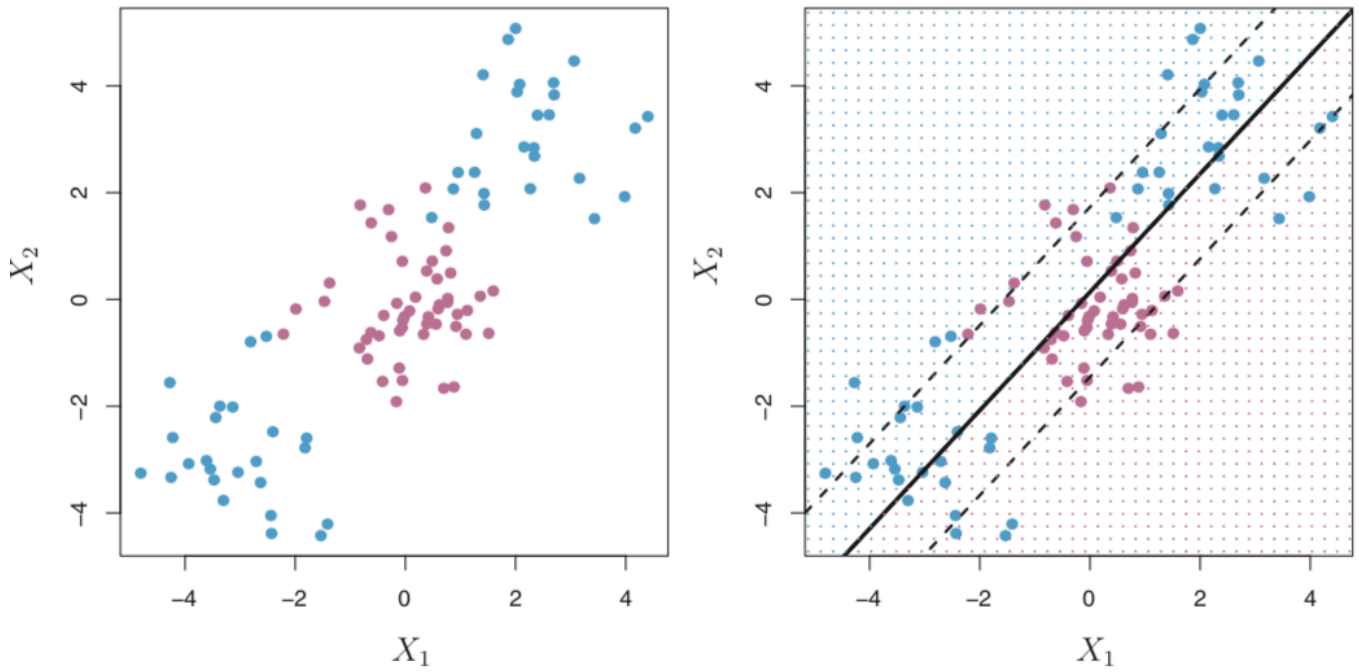
$$\min_{w,b,\xi} \; \underbrace{\frac{1}{2}||\boldsymbol{w}|\Big|^2}_{\text{Large margin}} + C \underbrace{\sum_n l^{(hinge)}(\; y_n(\boldsymbol{w} \cdot \boldsymbol{x_n} + b))}_{\text{Small slack:}}$$

TODAY: KERNELS

- Problem: linear classifiers
  - Easy to implement and easy to optimize
  - But limited to linear decision boundaries

- What can we do about it?
  - Today's topic: Kernels

2D DATA AND A LINEAR SVM



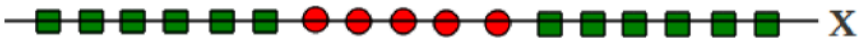
KERNEL METHODS

- Goal: keep advantages of linear models, but make them capture non-linear patterns in data!

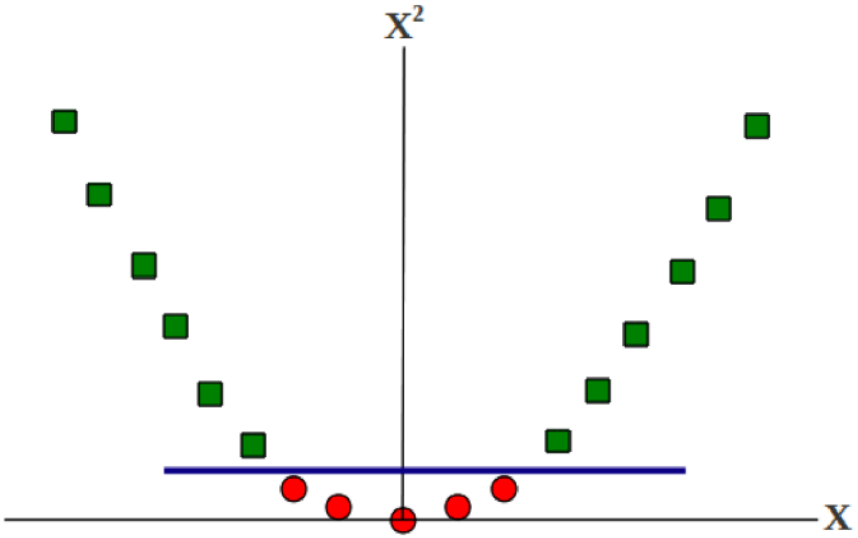
- How?
  - Use features of features
  - Map data to higher dimensions where it exhibits linear patterns

CLASSIFYING NON-LINEARLY SEPARABLE DATA WITH A LINEAR CLASSIFIER

- Example 1: Non-linearly separable data in 1D

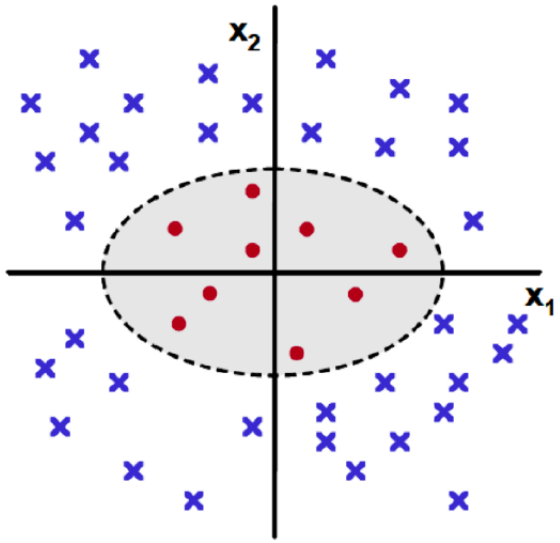


- Becomes linearly separable in new 2D space
- Defined by the following mapping: $(x) \rightarrow (x, x^2)$



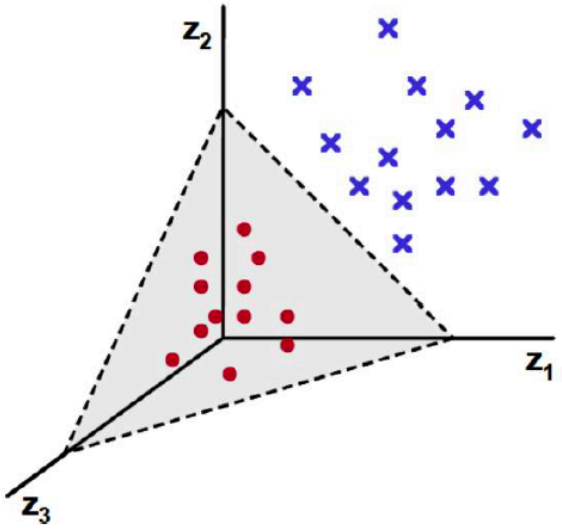
CLASSIFYING NON-LINEARLY SEPARABLE DATA WITH A LINEAR CLASSIFIER

- Example 2: Non-linearly separable data in 2D



- Becomes linearly separable in 3D defined by following mapping:

$$\boldsymbol{x} = (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



DEFINING FEATURE MAPPINGS

- Map an original feature vector $\boldsymbol{x} = (x_1, x_2, x_3, \dots, x_D)$ to an expanded version $\boldsymbol{\phi}(\boldsymbol{x})$
- Example: quadratic feature mapping represents feature combinations

$$\begin{aligned}\boldsymbol{\phi}(\boldsymbol{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \dots, \sqrt{2}x_D,\\ x_1^2, x_1x_2, x_1x_3, \dots, x_1x_D,\\ x_2x_1, x_2^2, x_2x_3, \dots, x_2x_D,\\ x_3x_1, x_3x_2, x_3^2, \dots, x_3x_D,\\ \dots,\\ x_Dx_1, x_Dx_2, x_Dx_3, \dots, x_D^2)\end{aligned}$$

FEATURE MAPPINGS

- Pros: can help turn non-linear classification problem into linear problem

- Cons: "feature explosion" creates issues when training linear classifier in new feature space
  - Computational: More computationally expensive to train
  - Statistical: More training examples needed to avoid overfitting

KERNEL METHODS

- Goal: keep advantages of linear models, but make them capture non-linear patterns in data!

- How?
  - By mapping data to higher dimensions where it exhibits linear patterns
  - **By rewriting linear models so that the mapping never needs to be explicitly computed**

THE KERNEL TRICK

1. Rewrite learning algorithms so they only depend on dot products between two examples
2. Replace dot product $\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z})$

   by kernel function k(x,z)

   which computes the dot product implicitly

- Intuition: kernel function quantifies the similarity of two examples
  - For example, the linear kernel $\mathbf{k}(\mathbf{x}, \mathbf{z}) = \boldsymbol{x} \cdot \boldsymbol{z}$ is equivalent to Pearson correlation

KERNEL FUNCTION EXAMPLE 1

- Consider two examples $\boldsymbol{x} = (x_1, x_2)$ and $\boldsymbol{z} = (z_1, z_2)$
- Let's assume we are given a quadratic kernel k that takes as inputs $\boldsymbol{x}$ and $\boldsymbol{z}$

$$\begin{aligned}\boldsymbol{k}(\boldsymbol{x}, \boldsymbol{z}) &= (\boldsymbol{x} \cdot \boldsymbol{z})^2 \\ &= (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2 z_2^2 \\ &= \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right) \cdot (z_1^2, \sqrt{2} z_1 z_2, z_2^2) \\ &= \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z})\end{aligned}$$

- k implicitly defines a mapping $\boldsymbol{\phi}$ to a higher dimensional space

$$\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$$

KERNEL FUNCTION EXAMPLE 2

$$\begin{aligned}\boldsymbol{\phi}(\boldsymbol{x}) =< 1, &\sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_3, \ldots, \sqrt{2}x_D,\\ &x_1^2, x_1x_2, x_1x_3, \ldots, x_1x_D,\\ &x_2x_1, x_2^2, x_2x_3, \ldots, x_2x_D,\\ &x_3x_1, x_3x_2, x_3^2, \ldots, x_3x_D,\\ &\ldots,\\ &x_Dx_1, x_Dx_2, x_Dx_3, \ldots, x_D^2 >\end{aligned}$$

What is the function k(x,z) that can implicitly compute the dot product $\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{z})$?

$$\begin{aligned}\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z}) &= 1 + 2x_1z_1 + 2x_2z_2 + \cdots + 2x_Dz_D + x_1^2z_1^2 + \cdots + x_1x_Dz_1z_D +\\ &\quad \cdots + x_Dx_1z_Dz_1 + x_Dx_2z_Dz_2 + \cdots + x_D^2z_D^2 \qquad (11.2)\\ &= 1 + 2\sum_d x_dz_d + \sum_d\sum_e x_dx_ez_dz_e \qquad (11.3)\\ &= 1 + 2\boldsymbol{x}\cdot\boldsymbol{z} + (\boldsymbol{x}\cdot\boldsymbol{z})^2 \qquad (11.4)\\ &= (1 + \boldsymbol{x}\cdot\boldsymbol{z})^2 \qquad (11.5)\end{aligned}$$

KERNELS: FORMALLY DEFINED

- Each kernel has an associated feature mapping $\boldsymbol{\phi}$
- $\boldsymbol{\phi}$ takes input $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ (input space) and maps it to $\mathcal{F}$ (feature space)
- Kernel k(**x**,**z**) takes two inputs and gives their similarity in $\mathcal{F}$ space

$$\phi: \boldsymbol{\mathcal{X}} \rightarrow \mathcal{F}$$
$$k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}, \qquad k(\boldsymbol{x}, \boldsymbol{z}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{z})$$

- $\mathcal{F}$ needs to be a vector space with a dot product defined in it
  - Also called Hilbert space

MERCER'S CONDITION

- Can any function be used as a kernel function?
  - No. It must satisfy Mercer's condition

- For k to be a kernel function
  - There must exist a Hilbert Space $\mathcal{F}$ for which k defines a dot product
  - The above is true if K is a positive semi-definite (psd) function
  - For all square integrable functions f, the following holds:

$$\iint f(x)k(x,z)f(z)dxdz$$

Non-trivial to show

CONSTRUCTING KERNEL COMBINATIONS

- If k_1 and k_2 are kernel functions, then the following are as well
  - Direct sum $k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z}) + k_2(\boldsymbol{x}, \boldsymbol{z})$
  - Scalar product: $k(\boldsymbol{x}, \boldsymbol{z}) = \alpha k_1(\boldsymbol{x}, \boldsymbol{z})$
  - Direct product: $k(\boldsymbol{x}, \boldsymbol{z}) = k_1(\boldsymbol{x}, \boldsymbol{z}) k_2(\boldsymbol{x}, \boldsymbol{z})$
- Can use Mercer's condition to prove

COMMONLY USED KERNEL FUNCTIONS

Linear (trivial) Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z} \text{ (mapping function } \phi \text{ is identity - no mapping)}$$

Quadratic Kernel:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2 \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^2$$

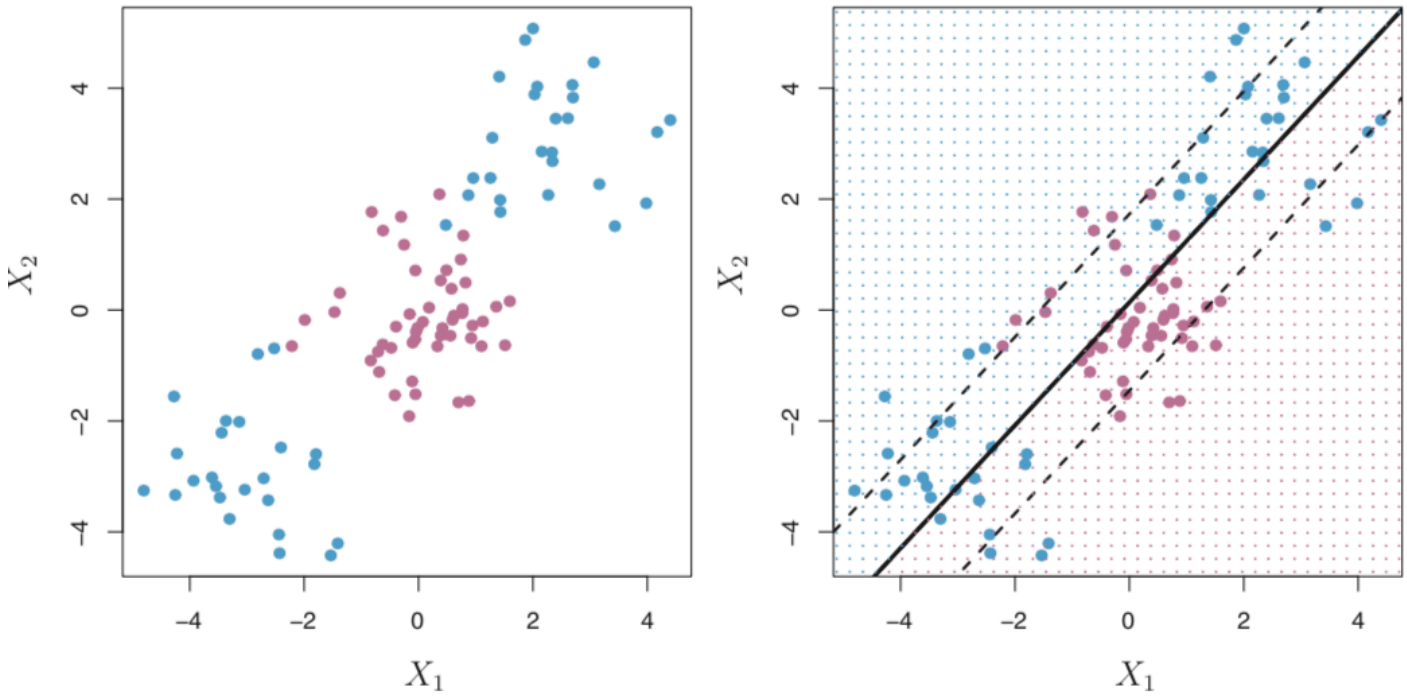
Polynomial Kernel (of degree d):

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^d \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^d$$

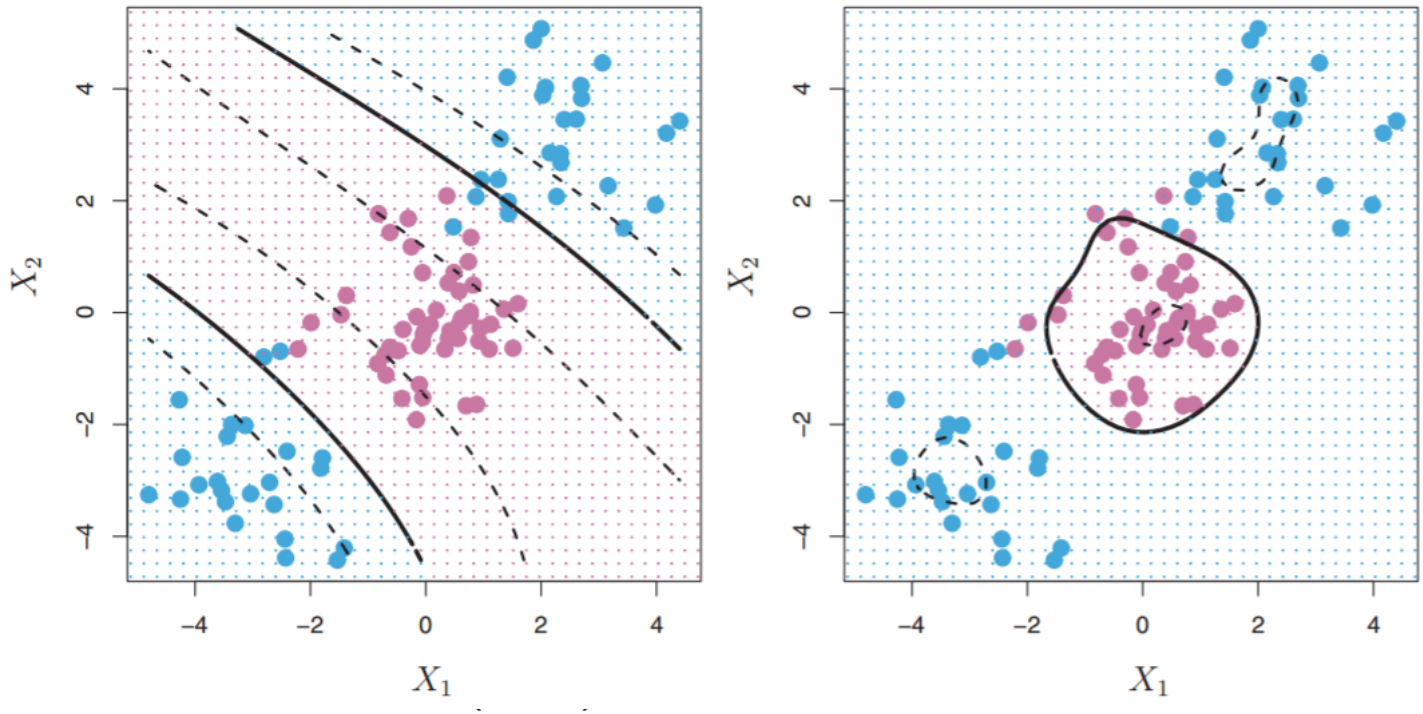
Radial Basis Function (RBF) Kernel:

$$k(\mathbf{x}, \mathbf{z}) = \exp[-\gamma ||\mathbf{x} - \mathbf{z}||^2]$$

2D DATA AND A LINEAR KERNEL



DEGREE 3 POLYNOMIAL AND RADIAL KERNELS



$$k(\mathbf{x}, \mathbf{z}) = \exp[-\gamma||\mathbf{x} - \mathbf{z}||^2]$$

How can you decide which kernel to use?

THE KERNEL TRICK

- Rewrite learning algorithms so they only depend on **dot products between two examples**
- Replace dot product $\boldsymbol{\phi}(x)^T\boldsymbol{\phi}(z)$

  by kernel function k(x,z)

  which computes the dot product **implicitly**

"KERNELIZING" THE PERCEPTRON

Algorithm 5 PERCEPTRONTRAIN(**D**, *MaxIter*)

1: $w_d \leftarrow 0$, for all $d = 1 \ldots D$ // initialize weights
2: $b \leftarrow 0$ // initialize bias
3: **for** $iter = 1 \ldots MaxIter$ **do**
4: **for all** $(x, y) \in \mathbf{D}$ **do**
5: $a \leftarrow \sum_{d=1}^{D} w_d \, x_d + b$ // compute activation for this example
6: **if** $ya \leq 0$ **then**
7: $w_d \leftarrow w_d + yx_d$, for all $d = 1 \ldots D$ // update weights
8: $b \leftarrow b + y$ // update bias
9: **end if**
10: **end for**
11: **end for**
12: **return** $w_0, w_1, \ldots, w_D, b$

"KERNELIZING" THE PERCEPTRON

- Goal: Remove the explicit dependence of this algorithm on ϕ and w

Algorithm 29 PERCEPTRONTRAIN(**D**, *MaxIter*)

```
1:  w ← 0, b ← 0                                   // initialize weights and bias
2:  for iter = 1 ... MaxIter do
3:      for all (x,y) ∈ D do
4:          a ← w · φ(x) + b                       // compute activation for this example
5:          if ya ≤ 0 then
6:              w ← w + y φ(x)                     // update weights
7:              b ← b + y                          // update bias
8:          end if
9:      end for
10: end for
11: return w, b
```

Can we apply the kernel trick?
Not yet, we need to rewrite the algorithm using dot products between examples

"KERNELIZING" THE PERCEPTRON

- Perceptron Representer Theorem: During a run of the perceptron algorithm, the weight vector **w** can always be represented as a linear combination of the expanded training data $\boldsymbol{\phi}(\mathrm{x}_1), \boldsymbol{\phi}(\mathrm{x}_2), \ldots, \boldsymbol{\phi}(\mathrm{x}_\mathrm{N})$"
- Proof by induction
  - Suppose it's true before the kth update, and the update is on example n

$$\boldsymbol{w} = \sum_i \alpha_i \phi(\mathbf{x}_\mathrm{i})$$

  - The new weight vector is

$$\boldsymbol{w} = \sum_i \alpha_i \phi(\mathbf{x}_\mathrm{i}) + y_n \phi(\mathbf{x}_\mathrm{n}) = \sum_i (\alpha_i + y_n [i = n]) \phi(\mathbf{x}_\mathrm{i})$$

"KERNELIZING" THE PERCEPTRON

- We can use the perceptron representer theorem to compute activations as a **dot product** between examples

$$\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}) + b = \left(\sum_n \alpha_n \boldsymbol{\phi}(\boldsymbol{x}_n) \right) \cdot \boldsymbol{\phi}(\boldsymbol{x}) + b \qquad \text{definition of } \boldsymbol{w} \tag{11.6}$$

$$= \sum_n \alpha_n \left[\boldsymbol{\phi}(\boldsymbol{x}_n) \cdot \boldsymbol{\phi}(\boldsymbol{x}) \right] + b \qquad \text{dot products are linear} \tag{11.7}$$

"KERNELIZING" THE PERCEPTRON

Algorithm 30 KERNELIZEDPERCEPTRONTRAIN(**D**, *MaxIter*)

```
1:  α ← 0, b ← 0                                   // initialize coefficients and bias
2:  for iter = 1 ... MaxIter do
3:      for all (x_n, y_n) ∈ D do
4:          a ← Σ_m α_m φ(x_m) · φ(x_n) + b        // compute activation for this example
5:          if y_n a ≤ 0 then
6:              α_n ← α_n + y_n                    // update coefficients
7:              b ← b + y                          // update bias
8:          end if
9:      end for
10: end for
11: return α, b
```

Same training algorithm but no explicit reference to weights
Only depends on dot products between examples

We can apply the kernel trick!

KERNEL METHODS

- Goal: keep advantages of linear models, but make them capture non-linear patterns in data
- How?
  - By mapping data to higher dimensions where it exhibits linear patterns
  - By rewriting linear models so that the (higher dimensional) mapping never needs to be explicitly computed
    - Making the representation sparser and the computation more efficient!

DISCUSSION

- Other algorithms can be "kernelized":
  - We'll talk about kernels and SVM next

- Do Kernels address all the downsides of "feature explosion"?
  - Helps reduce computation cost during training
  - But overfitting remains an issue

SUMMARY

- Kernel functions
  - What they are, why they are useful, how they relate to feature combinations
- Kernelized perceptron
  - You should be able to derive it and implement it

ANNOUNCEMENTS

- Homework 3
  - Released by the end of this week

- My office hours next week
  - Wednesday 2-4pm – cancelled (on travel)
  - Available Monday 3-5pm or by appointment

ACKNOWLEDGEMENTS

- These slides use materials by Marine Carpuat and Piyush Rai