CS 4824/ECE 4424: Kernel Perceptron

Acknowledgement:
Many of these slides are derived from Tom Mitchell, Pascal Poupart, Pieter Abbeel, Eric Eaton, Carlos Guestrin, William Cohen, and Andrew Moore.

Kernel Methods

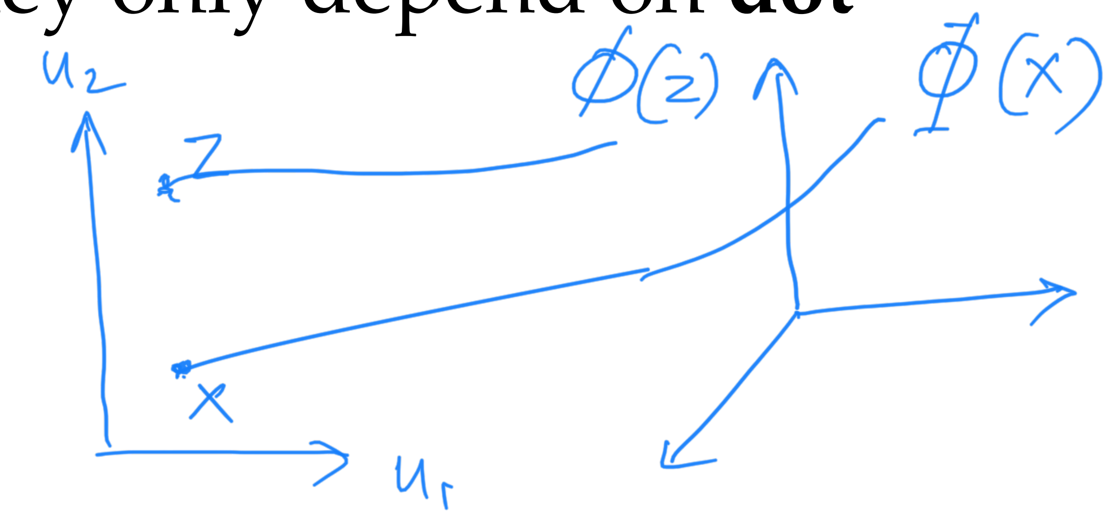
- Goal: keep advantages of linear models, but make them capture non-linear patterns in data!

- **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

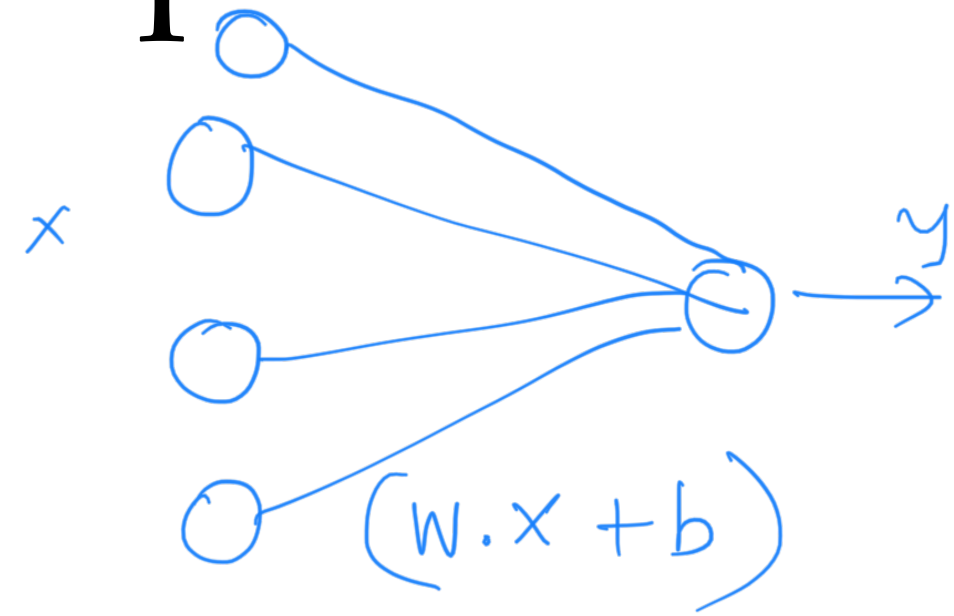
- Rewrite learning algorithms so they only depend on **dot products between two examples**



- Replace dot product $\boxed{\phi(x)^T\phi(z)}$
by **kernel function** $\boxed{k(x, z)}$
which computes the dot product **implicitly**

How?

Recall perceptron



- Let $y \in \{-1, 1\}\ \forall y$
- Initialize weights $\mathbf{w} = 0, b = 0$
- Run through the training data $i = 1,\ldots,n$
  - if $\boxed{y^i(\mathbf{w} \,.\, \mathbf{x}^i + b) < 0}$ checks if x^i is misclassified
    - $\mathbf{w} \leftarrow \mathbf{w} + y^i \mathbf{x}^i$ $w \leftarrow w - x^i$ if y is -1
    - $\boxed{b \leftarrow b + y^i}$ $w \leftarrow w + x^i$ if y is $+1$

"Kernelizing" the perceptron

- **Naïve approach**: let's explicitly train a perceptron in the new feature space

- Let $y \in \{-1, 1\}\ \forall y$

$x^i \longrightarrow \phi(x^i)$

- Initialize weights $\mathbf{w} = 0$, $b = 0$
- Run through the training data $i = 1,\ldots,n$
  - if $y^i(\mathbf{w} \cdot \phi(\mathbf{x}^i) + b) < 0$
    - $\mathbf{w} \leftarrow \mathbf{w} + y^i\phi(\mathbf{x}^i)$
    - $b \leftarrow b + y^i$

if $\phi(x_i)$ is misclassified

if $y = +1$ then $w \leftarrow w + \phi(x^i)$

if $y = -1$ then $w \leftarrow w - \phi(x^i)$

- And then summarize

$$W = \sum_{j=1}^{n} \boxed{\alpha_j} \boxed{y^j \phi(x^j)}$$

α_j is the number of misclassification on j^{th} traing example

- For making prediction on a new example $\mathbf{x}^i$

$$W \cdot \phi(x^i) + b = \sum_{j=1}^{n} \left(\alpha_j y^j \boxed{\phi(x_j) \cdot \phi(x_i)} \right) + b$$

"Kernelizing" the perceptron

- **Naïve approach**: let's explicitly train a perceptron in the new feature space

- Let $y \in \{-1, 1\}\ \forall y$

- Initialize weights $\mathbf{w} = 0, b = 0$
- Run through the training data $i = 1,\ldots,n$
  - if $y^i(\mathbf{w}\ .\ \phi(\mathbf{x}^i) + b) < 0$
    - $\mathbf{w} \leftarrow \mathbf{w} + y^i\phi(\mathbf{x}^i)$
    - $b \leftarrow b + y^i$

- And then summarize
- $\mathbf{w} = \sum_{j=1}^{n} \alpha_j y^j \phi(\mathbf{x}^j)$, where α_j is the number of misclassifications
- For making prediction on a new example $\mathbf{x}^i$
- $\mathbf{w}\ .\ \phi(\mathbf{x}^i) + b = \sum_{j=1}^{n} \alpha_j y^j \phi(\mathbf{x}^j)\ .\ \phi(\mathbf{x}^i) + b = \sum_{j=1}^{n} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^i) + b$

Kernel perceptron

- Let $y \in \{-1, 1\}\ \forall y$

n is the number of data points

- Initialize $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 0,\ b = 0$
- Run through the training data $i = 1, \ldots, n$
  - if $\underbrace{y^i \left(\sum_{j=1}^{n} \alpha_j y^j \boxed{\phi(x^j) \cdot \phi(x^i)} + b \right) < 0}$

$\alpha_i \leftarrow \alpha_i + 1$

$b \leftarrow b + y^i$

$\boxed{K(x^j, x^i)}$

- So we can plug in kernel functions and **implicitly** operate in the higher-dimensional space

Kernel perceptron

- Let $y \in \{-1, 1\} \; \forall y$

- Initialize $\alpha_1 = \ldots = \alpha_n = 0, b = 0$
- Run through the training data $i = 1,\ldots,n$
  - if $y^i(\sum_{j=1}^{n} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^i) + b) < 0$
    - $\alpha_i \leftarrow \alpha_i + 1$
    - $b \leftarrow b + y^i$

- So we can plug in kernel functions and **implicitly** operate in the higher-dimensional space

Kernel perceptron: primal and dual forms

Primal form

- Let $y \in \{-1, 1\}\ \forall y$
- Initialize weights $\mathbf{w} = 0, b = 0$
- Run through the training data $i = 1,\ldots,n$
  - if $y^i(\mathbf{w} \cdot \phi(\mathbf{x}^i) + b) < 0$
    - $\mathbf{w} \leftarrow \mathbf{w} + y^i \phi(\mathbf{x}^i)$
    - $b \leftarrow b + y^i$

Dual form

- Let $y \in \{-1, 1\}\ \forall y$
- Initialize $\alpha_1 = \ldots = \alpha_n = 0, b = 0$
- Run through the training data $i = 1,\ldots,n$
  - if $y^i(\sum_{j=1}^{n} \alpha_j y^j k(\mathbf{x}^j, \mathbf{x}^i) + b) < 0$
    - $\alpha_i \leftarrow \alpha_i + 1$
    - $b \leftarrow b + y^i$

Commonly used Kernels

- Polynomial Kernel of degree exactly d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^d$$

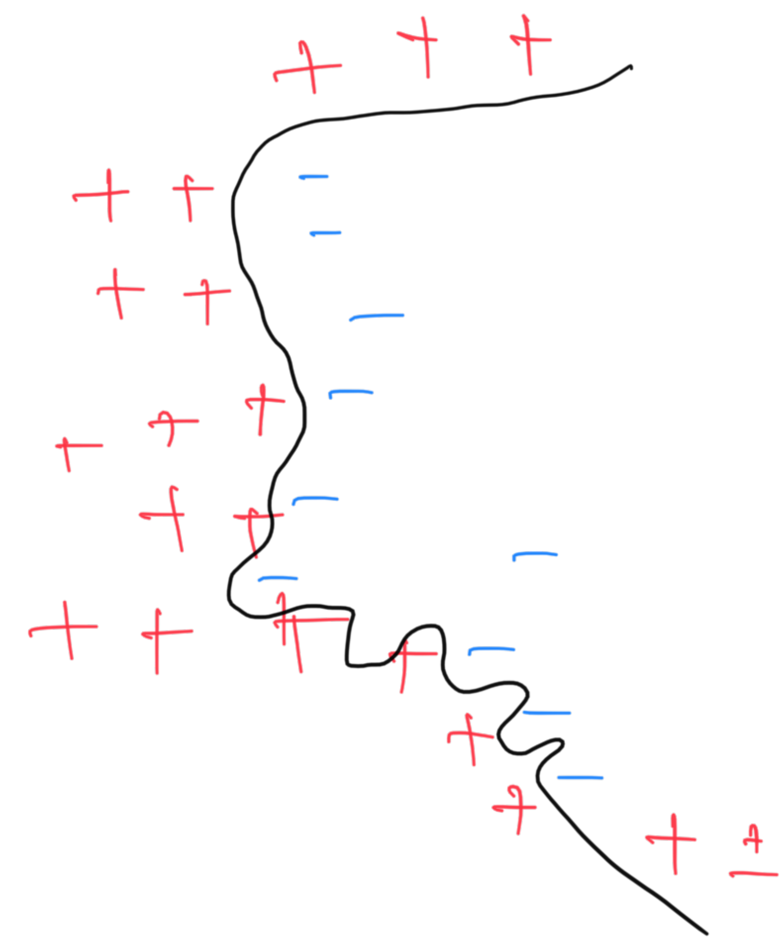
- Polynomial Kernel of degree up to d

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$$

- Gaussian Kernel

$$\boxed{K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||^2}{2\sigma^2}\right)}$$

- …and many others



Which functions can be Kernels?

- not all functions
- for some definitions of k(x,z) there is no corresponding projection φ(x)

- Well developed theory on this, including how to construct new kernels from existing ones

- Initially kernels were defined over data points in Euclidean space, but more recently over strings, over trees, over graphs, ...

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**

Discussion

- Other algorithms can be "Kernelized":
  - Logistic regression
  - We'll talk about Support Vector Machines next…

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