CS 4824/ECE 4424: Kernels

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

- Generalized linear models (up to ~2010)
  - Fixed basis functions
  - Hypothesis space is limited
  - Easy to optimize (usually convex)

- Neural networks (2010 onwards)
  - Adaptive basis functions
  - Rich hypothesis space
  - Hard to optimize (usually non-convex)
How to extend generalized linear models to have richer hypothesis?
How to generalize linear models for linearly non-separable data?

- Use features of features of features of features....

\[ \phi(x) = \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ x_1 x_2 \\ \vdots \\ x_2 x_3 \\ \vdots \\ x_1^2 \\ \vdots \\ x_2^2 \\ \vdots \end{pmatrix} \]

- **Challenge**: Feature space can get really large really quickly!
Non-linear features: 1D input

- Datasets that are linearly separable with some noise work out great:

- But what are we going to do if the dataset is just too hard?
Non-linear features: 1D input

- Datasets that are linearly separable with some noise work out great:

- But what are we going to do if the dataset is just too hard?

- How about... mapping data to a higher-dimensional space:
Mapping to higher dimensional space

Linearly non-separable
Mapping to higher dimensional space
Mapping to higher dimensional space

Map to 3D

Linearly separable
Input feature space

\[ \text{Polynomial of degree } d \]

What can go wrong?

\[ w \cdot \phi(x) \]
Higher order polynomials

- Number of terms = \( \binom{d + m - 1}{d} = \frac{(d + m - 1)!}{d!(m - 1)!} \)
- where \( m \) = dimension of input features, \( d \) = degree of polynomial
- Grows fast!
  - \( m = 100, d = 6 \)
  - \( \sim 1.6 \text{ billion terms} \)
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
  - More computationally expensive to train
  - 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

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

- Replace dot product $\phi(x) \cdot \phi(z)$ by kernel function $k(x, z)$ which computes the dot product implicitly.
Example of Kernel function

- Consider two examples $x = \{x_1, x_2\}$ and $z = \{z_1, z_2\}$
  
  $\quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$

- Let’s assume we are given a function $k$ (kernel) that takes as inputs $x$ and $z$
  
  $k(x, z) = (x \cdot z)^2$

- The above $k$ implicitly defines a mapping $\phi$ to a higher dimensional space

- Cool! taking a dot product and an exponential gives same results as mapping into high dimensional space and then taking dot product

- The above $k$ implicitly defines a mapping $\phi$ to a higher dimensional space
Kernel function

\[ k(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle \]

- But, it isn’t obvious yet how we will incorporate it into actual learning algorithms.

We will do that next...
“Kernelizing” learning algorithms

- **Key idea**: map to higher dimensional space
  - If \( x \) is in \( \mathbb{R}^n \), then \( \phi(x) \) is in \( \mathbb{R}^m \) for \( m > n \)
  - We can now learn feature weights \( w \) in \( \mathbb{R}^m \) and
    - predict \( y \) by computing \( w \cdot \phi(x) \)
  - Linear function in the higher dimensional space will be non-linear in the original space
Question to think about…

How can we “Kernelize” perceptron learning algorithm?