Kernels Methods in Machine Learning

• Perceptron. Geometric Margins.
• Support Vector Machines (SVMs).

Maria-Florina Balcan
02/03/2016
Quick Recap about Perceptron andMargins
The Online Learning Model

- Example arrive *sequentially*.
- We need to make a prediction.
  Afterwards observe the outcome.

For \( i=1, 2, \ldots \):

- **Phase i:**
  - Online Algorithm
  - Example \( x_i \)
  - Prediction \( h(x_i) \)
  - Observe \( c^*(x_i) \)

**Mistake bound model**

- Analysis wise, make **no distributional assumptions**.
- **Goal:** **Minimize** the number of mistakes.
**Perceptron Algorithm in Online Model**

WLOG homogeneous linear separators [$w_0 = 0$].

- Set $t=1$, start with the all zero vector $w_1$.
- Given example $x$, predict $+$ iff $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive, $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, $w_{t+1} \leftarrow w_t - x$

**Note 1:** $w_t$ is weighted sum of incorrectly classified examples

$$w_t = a_{i_1} x_{i_1} + \cdots + a_{i_k} x_{i_k}$$

So, $w_t \cdot x = a_{i_1} x_{i_1} \cdot x + \cdots + a_{i_k} x_{i_k} \cdot x$

**Note 2:** Number of mistakes ever made depends only on the geometric margin of examples seen.

- No matter how long the sequence is or how high dimension $n$ is!
**Geometric Margin**

**Definition:** The *margin* of example $x$ w.r.t. a linear sep. $w$ is the distance from $x$ to the plane $w \cdot x = 0$.

If $||w|| = 1$, margin of $x$ w.r.t. $w$ is $|x \cdot w|$.
**Geometric Margin**

**Definition:** The margin of example $x$ w.r.t. a linear separator $w$ is the distance from $x$ to the plane $w \cdot x = 0$.

**Definition:** The margin $\gamma_w$ of a set of examples $S$ w.r.t. a linear separator $w$ is the smallest margin over points $x \in S$.

**Definition:** The margin $\gamma$ of a set of examples $S$ is the maximum $\gamma_w$ over all linear separators $w$. 

![Diagram of geometric margin with points and plane separation](image-url)
Poll time
Theorem: If data linearly separable by margin $\gamma$ and points inside a ball of radius $R$, then Perceptron makes $\leq \left(\frac{R}{\gamma}\right)^2$ mistakes.

- No matter how long the sequence is, how high dimension $n$ is!
**Margin Important Theme in ML**

- **If large margin, # mistakes Peceptron makes is small** *(independent on the dim of the ambient space)*!

- **Large margin can help prevent overfitting.**
  - **If large margin** $\gamma$ **and if alg. produces a large margin classifier, then amount of data needed depends only on** $R/\gamma$ [Bartlett & Shawe-Taylor ’99].
So far, talked about margins in the context of (nearly) linearly separable datasets
What if Not Linearly Separable

Problem: data not linearly separable in the most natural feature representation.

Example: vs

No good linear separator in pixel representation.

Solutions:

• “Learn a more complex class of functions”
  • (e.g., decision trees, neural networks, boosting).

• “Use a Kernel” (a neat solution that attracted a lot of attention)

• “Use a Deep Network”

• “Combine Kernels and Deep Networks”
Overview of Kernel Methods

What is a Kernel?

A kernel $K$ is a legal def of dot-product: i.e. there exists an implicit mapping $\Phi$ s.t. $K(x, y) = \Phi(x) \cdot \Phi(y)$

E.g., $K(x, y) = (x \cdot y + 1)^d$

$\phi: \text{(n-dimensional space)} \rightarrow \text{n^d-dimensional space}$

Why Kernels matter?

• Many algorithms interact with data only via dot-products.

• So, if replace $x \cdot z$ with $K(x, z)$ they act implicitly as if data was in the higher-dimensional $\Phi$-space.

• If data is linearly separable by large margin in the $\Phi$-space, then good sample complexity.
Kernels

Definition

$K(\cdot, \cdot)$ is a kernel if it can be viewed as a legal definition of inner product:

- $\exists \phi: X \to \mathbb{R}^N$ s.t. $K(x, z) = \phi(x) \cdot \phi(z)$
  - Range of $\phi$ is called the $\Phi$-space.
  - $N$ can be very large.
- But think of $\phi$ as implicit, not explicit!!!
Example

For $n=2$, $d=2$, the kernel $K(x, z) = (x \cdot z)^d$ corresponds to

$$(x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$$
Example

\( \phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3, (x_1, x_2) \rightarrow \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \)

\( \phi(x) \cdot \phi(z) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2) \)

\[ = (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z) \]
Kernels

Definition

\( \mathcal{K}(\cdot,\cdot) \) is a kernel if it can be viewed as a legal definition of inner product:

- \( \exists \phi : X \to \mathbb{R}^N \) s.t. \( \mathcal{K}(x,z) = \phi(x) \cdot \phi(z) \)
  - Range of \( \phi \) is called the \( \Phi \)-space.
  - \( N \) can be very large.
- But think of \( \phi \) as \textit{implicit}, not explicit!!!!
**Example**

**Note:** feature space might not be unique.

\[ \phi: \mathbb{R}^2 \to \mathbb{R}^3, (x_1, x_2) \to \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \]

\[ \phi(x) \cdot \phi(z) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \cdot (z_1^2, z_2^2, \sqrt{2}z_1z_2) \]

\[ = (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z) \]

\[ \phi: \mathbb{R}^2 \to \mathbb{R}^4, (x_1, x_2) \to \Phi(x) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \]

\[ \phi(x) \cdot \phi(z) = (x_1^2, x_2^2, x_1x_2, x_2x_1) \cdot (z_1^2, z_2^2, z_1z_2, z_2z_1) \]

\[ = (x \cdot z)^2 = K(x, z) \]
Avoid explicitly expanding the features

Feature space can grow really large and really quickly....

Crucial to think of $\phi$ as implicit, not explicit!!!!

- Polynomial kernel degree $d$, $k(x, z) = (x^\top z)^d = \phi(x) \cdot \phi(z)$
  - $x_1^d, x_1 x_2 \ldots x_d, x_1^2 x_2 \ldots x_{d-1}$
  - Total number of such feature is
    $$(d + n - 1 \choose d) = \frac{(d + n - 1)!}{d!(n - 1)!}$$
  - $d = 6, n = 100$, there are $1.6$ billion terms

$O(n)$ computation!
Kernelizing a learning algorithm

• If all computations involving instances are in terms of inner products then:
  
  ▪ Conceptually, work in a very high diml space and the alg’s performance depends only on linear separability in that extended space.
  
  ▪ Computationally, only need to modify the algo by replacing each $x \cdot z$ with a $K(x, z)$.

• Examples of kernalizable algos:
  
  • classification: Perceptron, SVM.
  
  • regression: linear, ridge regression.
  
  • clustering: k-means.
Kernelizing the Perceptron Algorithm

- Set $t=1$, start with the all zero vector $w_1$.
- Given example $x$, predict + iff $w_t \cdot x \geq 0$
- On a mistake, update as follows:
  - Mistake on positive, $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, $w_{t+1} \leftarrow w_t - x$

Easy to kernelize since $w_t$ is weighted sum of incorrectly classified examples $w_t = a_{i_1}x_{i_1} + \cdots + a_{i_k}x_{i_k}$

Replace $w_t \cdot x = a_{i_1}x_{i_1} \cdot x + \cdots + a_{i_k}x_{i_k} \cdot x$ with $a_{i_1}K(x_{i_1}, x) + \cdots + a_{i_k}K(x_{i_k}, x)$

Note: need to store all the mistakes so far.
Kernelizing the Perceptron Algorithm

- Given $x$, predict + iff
  \[ \phi(x_{i_{t-1}}) \cdot \phi(x) \geq 0 \]

- On the $t$th mistake, update as follows:
  - Mistake on positive, set $a_{i_t} \leftarrow 1$; store $x_{i_t}$
  - Mistake on negative, $a_{i_t} \leftarrow -1$; store $x_{i_t}$

Perceptron $w_t = a_{i_1}x_{i_1} + \cdots + a_{i_k}x_{i_k}$

$w_t \cdot x = a_{i_1}x_{i_1} \cdot x + \cdots + a_{i_k}x_{i_k} \cdot x \Rightarrow a_{i_1}K(x_{i_1}, x) + \cdots + a_{i_k}K(x_{i_k}, x)$

Exact same behavior/prediction rule as if mapped data in the $\phi$-space and ran Perceptron there!

Do this implicitly, so computational savings!!!!!
Generalize Well if Good Margin

- If data is linearly separable by margin in the $\phi$-space, then small mistake bound.

- If margin $\gamma$ in $\phi$-space, then Perceptron makes $\left(\frac{R}{\gamma}\right)^2$ mistakes.
Kernels: More Examples

- Linear: $K(x, z) = x \cdot z$

- Polynomial: $K(x, z) = (x \cdot z)^d$ or $K(x, z) = (1 + x \cdot z)^d$

- Gaussian: $K(x, z) = \exp \left[ -\frac{||x-z||^2}{2\sigma^2} \right]$

- Laplace Kernel: $K(x, z) = \exp \left[ -\frac{||x-z||}{2\sigma^2} \right]$

- Kernel for non-vectorial data, e.g., measuring similarity between sequences.
Properties of Kernels

Theorem (Mercer)

\( K \) is a kernel if and only if:

- \( K \) is symmetric

- For any set of training points \( x_1, x_2, ..., x_m \) and for any \( a_1, a_2, ..., a_m \in \mathbb{R} \), we have:

\[
\sum_{i,j} a_i a_j K(x_i, x_j) \geq 0
\]

\[ a^T K a \geq 0 \]

I.e., \( K = (K(x_i, x_j))_{i,j=1,...,n} \) is positive semi-definite.
Kernel Methods

• Offer great modularity.

• No need to change the underlying learning algorithm to accommodate a particular choice of kernel function.

• Also, we can substitute a different algorithm while maintaining the same kernel.
Kernel, Closure Properties

Easily create new kernels using basic ones!

Fact: If $K_1(\cdot;\cdot)$ and $K_2(\cdot;\cdot)$ are kernels $c_1 \geq 0, c_2 \geq 0,$

then $K(x, z) = c_1 K_1(x, z) + c_2 K_2(x, z)$ is a kernel.

Key idea: concatenate the $\phi$ spaces.

\[ \phi(x) = (\sqrt{c_1} \phi_1(x), \sqrt{c_2} \phi_2(x)) \]

\[ \phi(x) \cdot \phi(z) = c_1 \phi_1(x) \cdot \phi_1(z) + c_2 \phi_2(x) \cdot \phi_2(z) \]

$K_1(x, z)$ $K_2(x, z)$
Kernel, Closure Properties

Easily create new kernels using basic ones!

**Fact:** If $K_1(\cdot, \cdot)$ and $K_2(\cdot, \cdot)$ are kernels,
then $K(x, z) = K_1(x, z)K_2(x, z)$ is a kernel.

**Key idea:** $\phi(x) = (\phi_{1,i}(x) \phi_{2,j}(x))_{i \in \{1,\ldots,n\}, j \in \{1,\ldots,m\}}$

\[
\phi(x) \cdot \phi(z) = \sum_{i,j} \phi_{1,i}(x) \phi_{2,j}(x) \phi_{1,i}(z) \phi_{2,j}(z)
= \sum_i \phi_{1,i}(x) \phi_{1,i}(z) \left( \sum_j \phi_{2,j}(x) \phi_{2,j}(z) \right)
= \sum_i \phi_{1,i}(x) \phi_{1,i}(z) K_2(x, z) = K_1(x, z) K_2(x, z)
\]
Kernels, Discussion

• If all computations involving instances are in terms of inner products then:
  - Conceptually, work in a very high diml space and the alg’s performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each $x \cdot z$ with a $K(x, z)$.

• Lots of Machine Learning algorithms are kernelizable:
  - classification: Perceptron, SVM.
  - regression: linear regression.
  - clustering: k-means.
Kernels, Discussion

- If all computations involving instances are in terms of inner products then:
  - Conceptually, work in a very high diml space and the alg’s performance depends only on linear separability in that extended space.
  - Computationally, only need to modify the algo by replacing each $x \cdot z$ with a $K(x, z)$.

How to choose a kernel:

- Kernels often encode domain knowledge (e.g., string kernels)

- Use Cross-Validation to choose the parameters, e.g., $\sigma$ for Gaussian Kernel $K(x, z) = \exp\left[-\frac{|x-z|^2}{2 \sigma^2}\right]$

- Learn a good kernel; e.g., [Lanckriet-Cristianini-Bartlett-El Ghaoui-Jordan’04]