Kernels + Support Vector Machines (SVMs)

SVM Readings:
Murphy 14.5
Bishop 7.1
HTF 12 - 12.38
Mitchell --

Matt Gormley
Lecture 12
February 27, 2016
Reminders

• Homework 4: Perceptron / Kernels / SVM
  – Due: Fri, Mar. 03 at 11:59pm

• Midterm Exam (Evening Exam)
  – Tue, Mar. 07 at 7:00pm – 9:30pm
  – See Piazza for details about location

• Grading

9 days for HW4
Outline

• **Kernels**
  – Kernel Perceptron
  – Kernel as a dot product
  – Gram matrix
  – Examples: Polynomial, RBF

• **Support Vector Machine (SVM)**
  – Background: Constrained Optimization, Linearly Separable, Margin
  – SVM Primal (Linearly Separable Case)
  – SVM Primal (Non-linearly Separable Case)
  – SVM Dual
KERNELS
Kernels: Motivation

Most real-world problems exhibit data that is not linearly separable.

Example: pixel representation for Facial Recognition:

Q: When your data is not linearly separable, how can you still use a linear classifier?

A: Preprocess the data to produce nonlinear features
Kernels: Motivation

• Motivation #1: Inefficient Features
  – Non-linearly separable data requires high dimensional representation
  – Might be prohibitively expensive to compute or store

• Motivation #2: Memory-based Methods
  – k-Nearest Neighbors (KNN) for facial recognition allows a distance metric between images -- no need to worry about linearity restriction at all
Kernels

Whiteboard

– Kernel Perceptron
– Kernel as a dot product
– Gram matrix
– Examples: RBF kernel, string kernel
Kernel Methods

• **Key idea:**
  1. **Rewrite** the algorithm so that we only work with *dot products* \( x^T z \) of feature vectors
  2. **Replace** the *dot products* \( x^T z \) with a *kernel function* \( k(x, z) \)

• The kernel \( k(x, z) \) can be **any** legal definition of a dot product:

\[
k(x, z) = \varphi(x)^T\varphi(z) \text{ for any function } \varphi : \mathcal{X} \to \mathbb{R}^D
\]

So we only compute the \( \varphi \) dot product **implicitly**

• This **“kernel trick”** can be applied to many algorithms:
  – classification: perceptron, SVM, …
  – regression: ridge regression, …
  – clustering: k-means, …
**Q:** These are just non-linear features, right?
**A:** Yes, but...

**Q:** Can’t we just compute the feature transformation $\varphi$ explicitly?
**A:** That depends...

**Q:** So, why all the hype about the kernel trick?
**A:** Because the **explicit features** might either be **prohibitively expensive** to compute or **infinite length** vectors.
Example: Polynomial Kernel

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

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

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

$$= (x_1z_1 + x_2z_2)^2 = (x \cdot z)^2 = K(x, z)$$
Example: Polynomial Kernel

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

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

Polynomial kernel degreee $d$, $k(x, z) = (x^T z)^d = \phi(x) \cdot \phi(z)$

- $x_1^d$, $x_1 x_2 \ldots x_d$, $x_1^2 x_2 \ldots x_{d-1}$
- Total number of such feature is $\binom{d+n-1}{d} = \frac{(d+n-1)!}{d! (n-1)!}$
- $d = 6, n = 100$, there are 1.6 billion terms

$k(x, z) = (x^T z)^d = \phi(x) \cdot \phi(z)$
**Kernel Examples**

**Side Note:** The feature space might not be unique!

**Explicit representation #1:**

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

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

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

**Explicit representation #2:**

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

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

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

These two different feature representations correspond to the same kernel function!
## Kernel Examples

<table>
<thead>
<tr>
<th>Name</th>
<th>Kernel Function (implicit dot product)</th>
<th>Feature Space (explicit dot product)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$K(x, z) = x^T z$</td>
<td>Same as original input space</td>
</tr>
<tr>
<td>Polynomial (v1)</td>
<td>$K(x, z) = (x^T z)^d$</td>
<td>All polynomials of degree $d$</td>
</tr>
<tr>
<td>Polynomial (v2)</td>
<td>$K(x, z) = (x^T z + 1)^d$</td>
<td>All polynomials up to degree $d$</td>
</tr>
<tr>
<td>Gaussian</td>
<td>$K(x, z) = \exp\left(-\frac{</td>
<td></td>
</tr>
<tr>
<td>Hyperbolic Tangent (Sigmoid)</td>
<td>$K(x, z) = \tanh(\alpha x^T z + c)$</td>
<td>(With SVM, this is equivalent to a 2-layer neural network)</td>
</tr>
</tbody>
</table>
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.010000)

RBF Kernel: \( K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2) \)
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.010000)

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2) \]
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.020000)

RBF Kernel: \( K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|^2) \)
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.040000)

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma ||x^{(i)} - x^{(j)}||^2_2) \]
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.080000)

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2) \]
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.160000)

RBF Kernel: $K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2)$
RBF Kernel:

\[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|^2_2) \]
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=0.640000)

RBF Kernel: \( K(x^{(i)}, x^{(j)}) = \exp(-\gamma \| x^{(i)} - x^{(j)} \|_2^2) \)
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=1.280000)

RBF Kernel: $K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|^2_2)$
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=2.560000)

RBF Kernel: \( K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2) \)
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=5.120000)

RBF Kernel: \( K(x^{(i)}, x^{(j)}) = \exp(-\gamma \| x^{(i)} - x^{(j)} \|_2^2) \)
RBF Kernel Example

Classification with SVM (kernel=rbf, gamma=10.000000)

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \| x^{(i)} - x^{(j)} \|^2) \]
RBF Kernel Example

KNN vs. SVM

Classification with KNN (k = 100, weights = 'uniform')

Classification with SVM (kernel=rbf, gamma=0.001000)

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma ||x^{(i)} - x^{(j)}||^2_2) \]
RBF Kernel Example

KNN vs. SVM

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2) \]
RBF Kernel Example

KNN vs. SVM

RBF Kernel: \[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|_2^2) \]
RBF Kernel Example

KNN vs. SVM

RBF Kernel:

\[ K(x^{(i)}, x^{(j)}) = \exp(-\gamma \|x^{(i)} - x^{(j)}\|^2_2) \]
Example: String Kernel

Setup:
– Input instances $\mathbf{x}$ are strings of characters (e.g. $\mathbf{x}^{(3)} = ['s', 'a', 't'], \mathbf{x}^{(7)} = ['c', 'a', 't']$
– Want indicator features for the presence / absence of each possible substring up to length $K$

Questions:
1. What is the best runtime of a single Standard Perceptron update?
2. What is the best runtime of a single Kernel Perceptron update?
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]
SUPPORT VECTOR MACHINE (SVM)
SVM: Optimization Background

Whiteboard

– Constrained Optimization
– Linear programming
– Quadratic programming
– Example: 2D quadratic function with linear constraints
Quadratic Program

Simple Quadratic Program
Quadratic Program
Quadratic Program
Quadratic Program

Simple Quadratic Program
Quadratic Program

Simple Quadratic Program
SVM

Whiteboard

– SVM Primal (Linearly Separable Case)
– SVM Primal (Non-linearly Separable Case)
SVM QP

SVM Quadratic Program

Classification with SVM (w=[-2.00, 3.00])
SVM QP
SVM QP
SVM QP

SVM Quadratic Program

Classification with SVM (w=[1.28, 1.62])
SVM QP

SVM Quadratic Program

Classification with SVM (w=[1.28, 1.60])
Support Vector Machines (SVMs)

**Input:** \( S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \);

Find \[ \text{argmin}_{w, \xi_1, \ldots, \xi_m} \|w\|^2 + C \sum_i \xi_i \text{ s.t.:} \]

- For all \( i \), \( y_i w \cdot x_i \geq 1 - \xi_i \)
  \[ \xi_i \geq 0 \]

Which is equivalent to:

**Can be kernelized!!!**

**Lagrangian Dual**

Input: \( S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \);

Find \[ \text{argmin}_\alpha \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j x_i \cdot x_j - \sum_i \alpha_i \text{ s.t.:} \]

- For all \( i \), \[ 0 \leq \alpha_i \leq C_i \]
  \[ \sum_i y_i \alpha_i = 0 \]
**SVMs (Lagrangian Dual)**

**Input:** $S=\{(x_1, y_1), \ldots, (x_m, y_m)\}$

**Find** $\text{argmin}_{\alpha} \frac{1}{2} \sum_i \sum_j y_i y_j \alpha_i \alpha_j x_i \cdot x_j - \sum_i \alpha_i$ s.t.:

- For all $i$, $0 \leq \alpha_i \leq C_i$
- $\sum_i y_i \alpha_i = 0$

- Final classifier is: $w = \sum_i \alpha_i y_i x_i$

- The points $x_i$ for which $\alpha_i \neq 0$ are called the “support vectors”
SVM Takeaways

• Maximizing the margin of a linear separator is a **good training criteria**
• Support Vector Machines (SVMs) learn a **max-margin linear classifier**
• The SVM optimization problem can be solved with **black-box Quadratic Programming (QP) solvers**
• Learned decision boundary is defined by its **support vectors**