Kernel Methods

David Rosenberg

New York University

November 1, 2015

Feature Extraction

- Focus on effectively representing $x \in \mathcal{X}$ as a vector $\phi(x) \in \mathbf{R}^d$.
- e.g. Bag of words:

[VentureBeat] As Android's reach expands, Google attracts fewer pioneer partners. The official theme of Google IO last week was Design, Develop and Distribute — but the unofficial one was Android Everywhere, as the mobile OS mounted new and renewed assaults on families of consumer devices.

Android: 2
Google: 2
IO: 1
Design: 1
Develop: 1
Distribute: 1
mobile: 1

Kernel Methods

- Primary focus is on comparing two inputs $w, x \in \mathcal{X}$.

Definition

A **kernel** is a function that takes a pair of inputs $w, x \in \mathcal{X}$ and returns a real value. That is, $k : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$.

- Can interpret $k(w,x)$ as a **similarity score**, but this is not precise.
- We will deal with symmetric kernels: $k(w,x) = k(x,w)$.

Comparing Documents

[VentureBeat] As Android's reach expands, Google attracts fewer pioneer partners. The official theme of Google IO last week was Design, Develop and Distribute — but the unofficial one was Android Everywhere, as the mobile OS mounted new and renewed assaults on families of consumer devices.

[Twitter Tweet] Google to take back platform control with Android Wear, Android TV, and Android Auto. This is good from a UI unity perspective, but it could potentially impact Android's openness. What are your thoughts on OEM customizations and their impact on Android as a platform?

Comparing Documents: Bag of Words

[VentureBeat] As Android's reach expands, Google attracts fewer pioneer partners. The official theme of Google IO last week was Design, Develop and Distribute — but the unofficial one was Android Everywhere, as the mobile OS mounted new and renewed assaults on families of consumer devices.

Android: 2
Google: 2
IO: 1
Design: 1
Develop: 1
Distribute: 1
mobile: 1

[Twitter Tweet] Google to take back platform control with Android Wear, Android TV, and Android Auto. This is good from a UI unity perspective, but it could potentially impact Android's openness. What are your thoughts on OEM customizations and their impact on Android as a platform?

Android: 5
Google: 1
UI: 1
OEM: 1
platform: 1
Wear: 1
TV: 1

Comparing Documents: Bag of Words

[VentureBeat] As Android's reach expands, Google attracts fewer pioneer partners. The official theme of Google IO last week was Design, Develop and Distribute — but the unofficial one was Android Everywhere, as the mobile OS mounted new and renewed assaults on families of consumer devices.

[Twitter Tweet] Google to take back platform control with Android Wear, Android TV, and Android Auto. This is good from a UI unity perspective, but it could potentially impact Android's openness. What are your thoughts on OEM customizations and their impact on Android as a platform?

Android: 2
Google: 2
IO: 1
Design: 1
Develop: 1
Distribute: 1
mobile: 1

Android: 5
Google: 1
UI: 1
OEM: 1
platform: 1
Wear: 1
TV: 1

Comparing Documents: Cosine Similarity

Android: 2
Google: 2
IO: 1
Design: 1
Develop: 1
Distribute: 1
mobile: 1

Android: 5
Google: 1
UI: 1
OEM: 1
platform: 1
Wear: 1
TV: 1

1. Normalize each feature vector to have $\|x\|_2 = 1$.

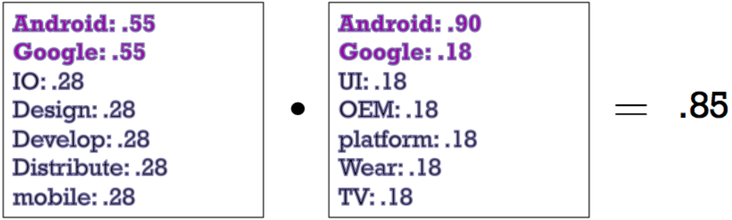
Comparing Documents

Android: .55
Google: .55
IO: .28
Design: .28
Develop: .28
Distribute: .28
mobile: .28

Android: .90
Google: .18
UI: .18
OEM: .18
platform: .18
Wear: .18
TV: .18

1. Normalize each feature vector to have $\|x\|_2 = 1$.
2. Take inner product

Comparing Documents: Cosine Similarity



1. Normalize each feature vector to have $\|x\|_2 = 1$.
2. Take inner product
3. Then define

$$k(\text{VentureBeat}, \text{Twitter Tweet}) = 0.85$$

Cosine Similarity Kernel

- Why the name? Recall

$$\langle w, x \rangle = \|w\| \|x\| \cos\theta,$$

where θ is the angle between $w, x \in \mathbf{R}^d$.

- So

$$k(w, x) = \cos\theta = \left\langle \frac{w}{\|w\|}, \frac{x}{\|x\|} \right\rangle$$

Linear Kernel

- Input space $\mathcal{X} = \mathbf{R}^d$

$$k(w, x) = w^T x$$

- When we "kernelize" an algorithm, we write it in terms of the linear kernel.
- Then we can swap it out a replace it with a more sophisticated kernel

Quadratic Kernel in $\mathbf{R}^2$

- Input space $\mathcal{X} = \mathbf{R}^2$
- Feature map:

$$\phi : (x_1, x_2) \mapsto \left(x_1, x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2\right)$$

- Gives us ability to represent conic section boundaries.
- Define kernel as inner product in feature space:

$$\begin{aligned} k(w,x) &= \langle \phi(w), \phi(x) \rangle \\ &= w_1x_1 + w_2x_2 + w_1^2x_1^2 + w_2^2x_2^2 + 2w_1w_2x_1x_2 \\ &= w_1x_1 + w_2x_2 + (w_1x_1)^2 + (w_2x_2)^2 + 2(w_1x_1)(w_2x_2) \\ &= \langle w, x \rangle + \langle w, x \rangle^2 \end{aligned}$$

Based on Guillaume Obozinski's Statistical Machine Learning course at Louvain, Feb 2014.

Quadratic Kernel in $\mathbf{R}^d$

- Input space $\mathcal{X} = \mathbf{R}^d$
- Feature map:

$$\phi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, \sqrt{2}x_1x_2, \ldots, \sqrt{2}x_ix_j, \ldots \sqrt{2}x_{d-1}x_d)^T$$

- Number of terms $= d + d(d+1)/2 \approx d^2/2$.
- Still have

$$\begin{aligned} k(w,x) &= \langle \phi(w), \phi(x) \rangle \\ &= \langle x, y \rangle + \langle x, y \rangle^2 \end{aligned}$$

- Computation for inner product with explicit mapping: $O(d^2)$
- Computation for implicit kernel calculation: $O(d)$.

Based on Guillaume Obozinski's Statistical Machine Learning course at Louvain, Feb 2014.

Polynomial Kernel in $\mathbf{R}^d$

- Input space $\mathcal{X} = \mathbf{R}^d$
- Kernel function:

$$k(w,x) = (1 + \langle w,x \rangle)^M$$

- Corresponds to a feature map with all terms up to degree M.
- For any M, computing the kernel has same computational cost
- Cost of explicit inner product computation grows rapidly in M.

Radial Basis Function (RBF) Kernel

- Input space $\mathcal{X} = \mathbf{R}^d$

$$k(w,x) = \exp\left(-\frac{\|w-x\|^2}{2\sigma^2}\right),$$

  where σ^2 is known as the bandwidth parameter.

- Does it act like a similarity score?
- Why "radial"?
- Have we departed from our "inner product of feature vector" recipe?
  - Yes and no: corresponds to an infinte dimensional feature vector
- Probably the most common nonlinear kernel.

Feature Vectors from a Kernel

- So what can we do with a kernel?
- We can generate feature vectors:
- Idea: Characterize input x by its similarity to r fixed prototypes in $\mathcal{X}$.

Definition

A **kernelized feature vector** for an input $x \in \mathcal{X}$ with respect to a kernel k and prototype points $\mu_1, \ldots, \mu_r \in \mathcal{X}$ is given by

$$\Phi(x) = [k(x, \mu_1), \ldots, k(x, \mu_r)] \in \mathbf{R}^r.$$

Kernel Machines

Definition

A **kernel machine** is a linear model with kernelized feature vectors.

This corresponds to a prediction functions of the form

$$\begin{aligned} f(x) &= \alpha^T \Phi(x) \\ &= \sum_{i=1}^{r} \alpha_i k(x, \mu_i), \end{aligned}$$

for $\alpha \in \mathbf{R}^r$.

An Interpretation

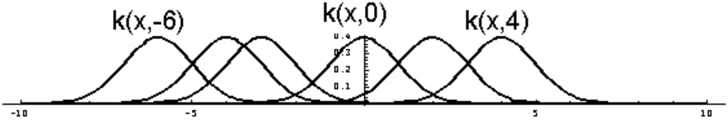
For each μ_i, we get a function on $\mathcal{X}$:

$$x \mapsto k(x, \mu_i)$$

$f(x)$ is a linear combination of these functions.

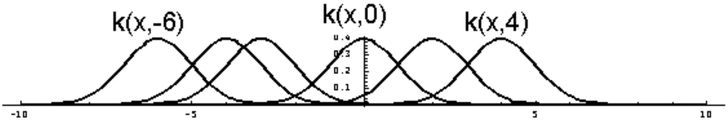
Kernel Machine Basis Functions

- Input space $\mathcal{X} = \mathbf{R}$
- RBF kernel $k(w,x) = \exp\left(-(w-x)^2\right)$.
- Prototypes at $\{-6, -4, -3, 0, 2, 4\}$.
- Corresponding basis functions:



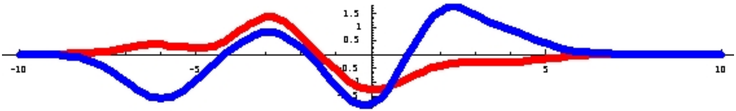
Kernel Machine Prediction Functions

- Basis functions



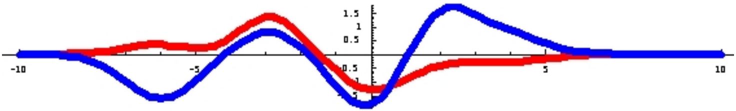
- Predictions of the form

$$f(x) = \sum_{i=1}^{r} \alpha_i k(x, \mu_i)$$



RBF Network

- An **RBF network** is a linear model with an RBF kernel.
  - First described in 1988 by Broomhead and Lowe (neural network literature)



- Characteristics:
  - Nonlinear
  - Smoothness depends on RBF kernel bandwidth

How to Choose Prototypes

- Uniform grid on space?
  - only feasible in low dimensions
  - where to focus the grid?
- Cluster centers of training data?
  - Possible, but clustering is difficult in high dimensions
- **Use all (or a subset of) the training points**
  - Most common approach for kernel methods

All Training Points as Prototypes

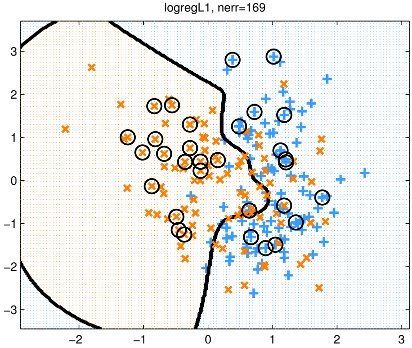
- Consider training inputs $x_1, \ldots, x_n \in \mathcal{X}$
- Then

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i).$$

- Requires all training examples for prediction?
- Not quite: Only need x_i for $\alpha_i \neq 0$.
- Want α_i's to be sparse.
  - Train with ℓ_1 regularization: ℓ_1-**regularized vector machine**
  - [Will show SVM also gives sparse functions of this form.]

ℓ_1-Regularized Vector Machine

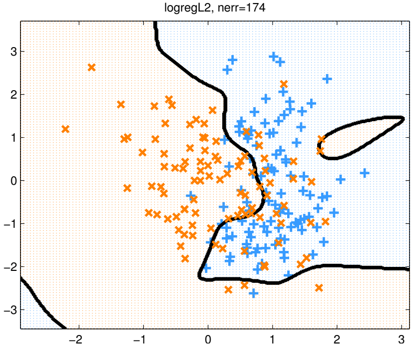
- RBF Kernel with bandwidth $\sigma = 0.3$.
- Linear hypothesis space: $\mathcal{F} = \{f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \mid \alpha \in \mathbf{R}^n\}$.
- Logistic loss function: $\ell(y, \hat{y}) = \log\left(1 + e^{-y\hat{y}}\right)$
- ℓ_1-regularization, $n = 200$ training points



KPM Figure 14.4b

ℓ_2-Regularized Vector Machine

- RBF Kernel with bandwidth $\sigma = 0.3$.
- Linear hypothesis space: $\mathcal{F} = \{f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \mid \alpha \in \mathbf{R}^n\}$.
- Logistic loss function: $\ell(y, \hat{y}) = \log\left(1 + e^{-y\hat{y}}\right)$
- ℓ_2-regularization, $n = 200$ training points



KPM Figure 14.4a

ℓ_2-Regularized Vector Machine for Regression

- Kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$ is symmetric (but nothing else).
- Hypothesis space (linear functions on kernelized feature vector)

$$\mathcal{F} = \left\{ f_\alpha(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \mid \alpha \in \mathbf{R}^n \right\}.$$

- Objective function (square loss with ℓ_2 regularization):

$$J(\alpha) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\alpha(x_i))^2 + \lambda \alpha^T \alpha,$$

where

$$f_\alpha(x_i) = \sum_{j=1}^{n} \alpha_j k(x_i, x_j).$$

- **Note: All dependence on x's is via the kernel function.**

The Kernel Matrix

- Note that

$$f(x_i) = \sum_{j=1}^{n} \alpha_j k(x_i, x_j)$$

only depends on the kernel function on all pairs of n training points.

Definition

The **kernel matrix** for a kernel k on a set $\{x_1, \ldots, x_n\}$ as

$$K = \left(k(x_i, x_j)\right)_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \cdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix} \in \mathbf{R}^{n \times n}.$$

Vectorizing the Vector Machine

Claim: $K\alpha$ gives the prediction vector $(f_\alpha(x_1),\ldots,f_\alpha(x_n))^T$:

$$
\begin{aligned}
K\alpha &= \begin{pmatrix} k(x_1,x_1) & \cdots & k(x_1,x_n) \\ \vdots & \ddots & \cdots \\ k(x_n,x_1) & \cdots & k(x_n,x_n) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \\
&= \begin{pmatrix} \alpha_1 k(x_1,x_1) + \cdots + \alpha_n k(x_1,x_n) \\ \vdots \\ \alpha_1 k(x_n,x_1) + \cdots + \alpha_n k(x_{1,}x_n) \end{pmatrix} \\
&= \begin{pmatrix} f_\alpha(x_1) \\ \vdots \\ f_\alpha(x_n) \end{pmatrix}.
\end{aligned}
$$

Vectorizing the Vector Machine

- The ith residual is $y_i - f_\alpha(x_i)$. We can vectorize as:

$$y - K\alpha \quad = \quad \begin{pmatrix} y_1 - f_\alpha(x_1) \\ \vdots \\ y_n - f_\alpha(x_n) \end{pmatrix}$$

- Sum of square residuals is

$$(y - K\alpha)^T (y - K\alpha)$$

- Objective function:

$$J(\alpha) = \frac{1}{n}\|y - K\alpha\|^2 + \lambda\alpha^T\alpha$$

Vectorizing the Vector Machine

- Consider $\mathcal{X} = \mathbf{R}^d$ and $k(w,x) = w^T x$ (linear kernel)
- Let $X \in \mathbf{R}^{n \times d}$ be the **design matrix**, which has each input vector as a row:

$$X = \begin{pmatrix} -x_1- \\ \vdots \\ -x_n- \end{pmatrix}.$$

- Then the kernel matrix is

$$K = XX^T = \begin{pmatrix} -x_1- \\ \vdots \\ -x_n- \end{pmatrix} \begin{pmatrix} | & \cdots & | \\ x_1 & \cdots & x_n \\ | & \cdots & | \end{pmatrix}$$

- And the objective function is

$$J(\alpha) = \frac{1}{n} \|y - XX^T\alpha\|^2 + \lambda \alpha^T \alpha$$

Features vs Kernels

Theorem

Suppose a kernel can be written as an inner product:

$$k(w,x) = \langle \phi(w), \phi(x) \rangle .$$

Then the kernel machine is a ***linear classifier*** *with feature map $\phi(x)$.*

- Mercer's Theorem characterizes kernels with these properties.

Features vs Kernels

Proof.

For prototype points $x_1, \ldots, x_r$,

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{r} \alpha_i k(x, x_i) \\
&= \sum_{i=1}^{r} \alpha_i \langle \phi(x), \phi(x_i) \rangle \\
&= \left\langle \sum_{i=1}^{r} \alpha_i \phi(x_i), \phi(x) \right\rangle \\
&= w^T \phi(x)
\end{aligned}
$$

where $w = \sum_{i=1}^{r} \alpha_i \phi(x_i)$. $\square$