Kernel Methods

David Rosenberg

New York University

January 14, 2018

Kernels: High-Level View

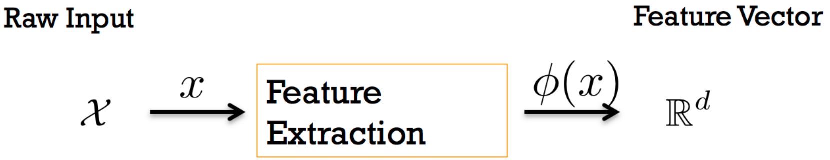
The Input Space $\mathcal{X}$

- Our general learning theory setup: no assumptions about $\mathcal{X}$
- But $\mathcal{X} = \mathbf{R}^d$ for the specific methods we've developed:
  - Ridge regression
  - Lasso regression
  - Linear SVM

Feature Extraction

Definition

Mapping an input from $\mathcal{X}$ to a vector in $\mathbf{R}^d$ is called **feature extraction** or **featurization**.



- e.g. Quadratic feature map: $\mathcal{X} = \mathbf{R}^d$

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

High-Dimensional Features Good but Expensive

- To get **expressive** hypothesis spaces using linear models,
  - need high-dimensional feature spaces
- But more costly in terms of computation and memory.

Some Methods Can Be "Kernelized"

Definition

A method is **kernelized** if inputs only appear inside inner products: $\langle \phi(x), \phi(y) \rangle$ for $x, y \in \mathcal{X}$.

- The function

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

is called the **kernel** function.

Kernel Evaluation Can Be Fast

Example

Quadratic feature map

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

has dimension $O(d^2)$, but

$$k(w,x) = \langle \phi(w), \phi(x) \rangle = \langle w,x \rangle + \langle w,x \rangle^2$$

- Naively explicit computation of $k(w,x)$: $O(d^2)$
- Implicit computation of $k(w,x)$: $O(d)$

Recap

1. Given a kernelized ML algorithm.
2. Can swap out the inner product for a new kernel function.
3. New kernel may correspond to a high dimensional feature space.
4. Once kernel matrix is computed, computational cost depends on number of data points, rather than the dimension of feature space.

Introduction

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: $k(w, x) = k(x, w)$.

Kernel Examples

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.

[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: 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: 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

Android: .55		**Android: .90**		
Google: .55		**Google: .18**		
IO: .28		UI: .18		
Design: .28	$\bullet$	OEM: .18	$=$	.85
Develop: .28		platform: .18		
Distribute: .28		Wear: .18		
mobile: .28		TV: .18		

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.

Kernelizing the SVM Dual

Linear SVM

- The SVM prediction function is the solution to

$$\min_{w\in\mathbf{R}^d,b\in\mathbf{R}} \frac{1}{2}\|w\|^2+\frac{c}{n}\sum_{i=1}^{n}\left(1-y_i\left[w^Tx_i+b\right]\right)_+.$$

- Found it's equivalent to solve the dual problem to get α^*:i

$$\begin{array}{ll}\sup_{\alpha} & \sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j x_j^T x_i\\ \text{s.t.} & \sum_{i=1}^{n}\alpha_i y_i=0\\ & \alpha_i\in\left[0,\frac{c}{n}\right]\ i=1,\ldots,n.\end{array}$$

- Notice: x's only show up as inner products with other x's.

Kernelization

Definition

We say a machine learning method is **kernelized** if all references to inputs $x \in \mathcal{X}$ are through an inner product between pairs of points $\langle x, y \rangle$ for $x, y \in \mathbf{R}^d$.

So far, we've only partially kernelized SVM

We've shown that the training portion is kernelized. Later we'll show the prediction portion is also kernelized.

SVM Dual Problem

- x's only show up in pairs of inner products: $x_j^T x_i = \langle x_j, x_i \rangle$:

$$
\begin{aligned}
\sup_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_j, x_i \rangle \\
\text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\
& \alpha_i \in \left[0, \frac{c}{n}\right] \; i = 1, \ldots, n.
\end{aligned}
$$

- Then primal optimal solution is given as:

$$
w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i
$$

and for any $\alpha_i \in \left(0, \frac{c}{n}\right)$,

$$
b^* = y_i - x_i^T w^*.
$$

SVM: Kernelizing b

- We found that for any j with $\alpha_j \in \left(0, \frac{c}{n}\right)$:

$$\begin{aligned} b^* &= y_j - x_j^T w^* \\ &= y_j - x_j^T \left(\sum_{i=1}^n \alpha_i^* y_i x_i \right). \\ &= y_j - \sum_{i=1}^n \alpha_i^* y_i \langle x_j, x_i \rangle . \end{aligned}$$

- What about kernelizing w^*?

$$w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$$

- Not obvious...
- But we really only care about kernelizing the predictions $f^*(x)$.

SVM: Kernelizing Predictions $f^*(x)$

- For any j with $\alpha_j \in \left(0, \frac{c}{n}\right)$:

$$
\begin{aligned}
f^*(x) &= x^T w^* + b^* \\
&= x^T \left(\sum_{i=1}^{n} \alpha_i^* y_i x_i \right) + b^* \\
&= \sum_{i=1}^{n} \alpha_i^* y_i \langle x_i, x \rangle + \left(y_j - \sum_{i=1}^{n} \alpha_i^* y_i \langle x_j, x_i \rangle \right)
\end{aligned}
$$

- We now have a fully kernelized version of SVM.
- Can we kernelize the primal version of the SVM?

Kernelizing the SVM Primal Problem

Kernelizing the SVM Primal Problem

- Primal SVM

$$\min_{w\in\mathbf{R}^d, b\in\mathbf{R}} \frac{1}{2}\|w\|^2 + \frac{c}{n}\sum_{i=1}^{n}\left(1 - y_i\left[w^T x_i + b\right]\right)_+.$$

- From our study of the dual, found that

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i.$$

- So w^* is a linear combination of the input vectors.
- Restrict to optimization to w of the form

$$w = \sum_{i=1}^{n} \beta_i x_i.$$

Some Vectorization

- **Design matrix** $X \in \mathbf{R}^{n \times d}$ has input vectors as rows:

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

- The contraint on w looks like

$$w = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} | & \cdots & | \\ x_1 & \cdots & x_n \\ | & \cdots & | \end{pmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix} = X^T \beta.$$

- So replace all w with $X^T\beta$, with $\beta \in \mathbf{R}^n$ unrestricted.

The Kernel Matrix (or the Gram Matrix)

Definition

For a set of $\{x_1, \ldots, x_n\}$ and an inner product $\langle \cdot, \cdot \rangle$on the set, the **kernel matrix** or the **Gram matrix** is defined as

$$K = \left(\langle x_i, x_j \rangle \right)_{i,j} = \begin{pmatrix} \langle x_1, x_1 \rangle & \cdots & \langle x_1, x_n \rangle \\ \vdots & \ddots & \cdots \\ \langle x_n, x_1 \rangle & \cdots & \langle x_n, x_n \rangle \end{pmatrix}.$$

Then for the standard Euclidean inner product $\langle x_i, x_j \rangle = x_i^T x_j$, we have

$$K = XX^T$$

Some Vectorization

- Regularization Term:

$$\|w\|^2 = w^T w = \beta^T X X^T \beta = \beta^T K \beta$$

- Prediction on training point x_i:

$$\begin{aligned} f(x_i) &= b + x_i^T w \\ &= b + x_i^T \left(\sum_{j=1}^{n} \beta_j x_j \right) \\ &= b + \sum_{j=1}^{n} \beta_j K_{ij} \end{aligned}$$

Kernelized Primal SVM

- Putting it together, kernelized primal SVM is

$$\min_{\beta\in\mathbf{R}^n, b\in\mathbf{R}} \frac{1}{2}\beta^T K\beta + \frac{c}{n}\sum_{i=1}^{n}\left(1-y_i\left[b+\sum_{j=1}^{n}\beta_j K_{ij}\right]\right)_+ .$$

- We can write this as a differentiable, constrained optimization problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\beta^T K\beta + \frac{c}{n}\mathbf{1}^T\xi \\ \text{subject to} & \xi \succeq 0 \\ & \xi \succeq (\mathbf{1} - Y\,[b + K\beta]), \end{array}$$

where $Y = \text{diag}(y_1, \ldots, y_n)$, $\mathbf{1}$ is a column vector of 1's, and $\succeq$ represent element-wise vector inequality.

Kernelized Primal SVM: Kernel Trick

- Kernelized primal SVM is

$$\min_{\beta\in\mathbf{R}^n, b\in\mathbf{R}} \frac{1}{2}\beta^T K\beta + \frac{c}{n}\sum_{i=1}^{n}\left(1-y_i\left[b+\sum_{j=1}^{n}\beta_j K_{ij}\right]\right)_+ .$$

- We derived this with $K = XX^T$, which corresponds to the linear kernel.
- Suppose we have another kernel defined in terms of a map ϕ, i.e.

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

  then we can just plug in the corresponding kernel matrix K_ϕ to the optimization problem above.
- What kernels can be written as an inner product of feature vectors?

Kernelizing Ridge Regression

Ridge Regression

- Recall the ridge regression objective:

$$J(w) = \|Xw - y\|^2 + \lambda \|w\|^2.$$

- Differentiating and setting equal to zero ,we get

$$\left(X^T X + \lambda I\right) w \quad = \quad X^T y$$

- On board to review?

Kernelizing Ridge Regression

- So we have, for $\lambda > 0$:

$$\begin{aligned}(X^TX+\lambda I)w &= X^Ty \\ \lambda w &= X^Ty - X^TXw \\ w &= \frac{1}{\lambda}X^T(y-Xw) \\ w &= X^T\alpha\end{aligned}$$

  for $\alpha = \lambda^{-1}(y-Xw) \in \mathbf{R}^n$.

- So w is "**in the span of the data**":

$$w = \begin{pmatrix} | & \cdots & | \\ x_1 & \cdots & x_n \\ | & \cdots & | \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \alpha_1 x_1 + \cdots \alpha_n x_n$$

Kernelizing Ridge Regression

- So plugging in $w = X^T\alpha$ to

$$\begin{aligned}
\alpha &= \lambda^{-1}(y - Xw) \\
\lambda\alpha &= y - XX^T\alpha \\
XX^T\alpha + \lambda\alpha &= y \\
\left(XX^T + \lambda I\right)\alpha &= y \\
\alpha &= (\lambda I + XX^T)^{-1}y
\end{aligned}$$

- So we have α. How to do prediction?

$$\begin{aligned}
Xw &= X\left(X^T\alpha\right) \\
&= \left(XX^T\right)(\lambda I + XX^T)^{-1}y
\end{aligned}$$

- To predict on new data, need the "cross-kernel" matrix, between new and old data.

Mercer's Theorem

Positive Semidefinite Matrices

Definition

A real, symmetric matrix $M \in \mathbf{R}^{n \times n}$ is **positive semidefinite (psd)** if for any $x \in \mathbf{R}^n$,

$$x^T M x \geqslant 0.$$

Theorem

The following conditions are each necessary and sufficient for M to be positive semidefinite:

- *M has a "square root", i.e. there exists R s.t. $M = R^T R$.*
- *All eigenvalues of M are greater than or equal to 0.*

Positive Semidefinite Function

Definition

A symmetric kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbf{R}$ is **positive semidefinite (psd)** if for any finite set $\{x_1, \ldots, x_n\} \in \mathcal{X}$, the kernel matrix on this set

$$K = \left(k(x_i, x_j)\right)_{i,j} = \begin{pmatrix} k(x_1, x_1) & \cdots & k(x_1, x_n) \\ \vdots & \ddots & \cdots \\ k(x_n, x_1) & \cdots & k(x_n, x_n) \end{pmatrix}$$

is a positive semidefinite matrix.

Mercer's Theorem

Theorem

A symmetric function $k(w,x)$ can be expressed as an inner product

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

for some ϕ if and only if $k(w,x)$ is ***positive semidefinite.***

- If we start with a psd kernel, can we generate more?

Additive Closure

- Suppose k_1 and k_2 are psd kernels with feature maps ϕ_1 and ϕ_2, respectively.
- Then
$$k_1(w,x)+k_2(w,x)$$
is a psd kernel.
- Proof: Concatenate the feature vectors to get
$$\phi(x)=(\phi_1(x),\phi_2(x)).$$
Then ϕ is a feature map for k_1+k_2.

Closure under Positive Scaling

- Suppose k is a psd kernel with feature maps ϕ.
- Then for any $\alpha > 0$,

$$\alpha k$$

  is a psd kernel.
- Proof: Note that

$$\phi(x) = \sqrt{\alpha}\phi(x)$$

  is a feature map for αk.

Scalar Function Gives a Kernel

- For any function $f(x)$,

$$k(w,x) = f(w)f(x)$$

  is a kernel.

- Proof: Let $f(x)$ be the feature mapping. (It maps into a 1-dimensional feature space.)

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

Closure under Hadamard Products

- Suppose k_1 and k_2 are psd kernels with feature maps ϕ_1 and ϕ_2, respectively.
- Then
$$k_1(w,x)k_2(w,x)$$
is a psd kernel.
- Proof: Take the outer product of the feature vectors:
$$\phi(x) = \phi_1(x)\left[\phi_2(x)\right]^T.$$
Note that $\phi(x)$ is a matrix.
- Continued...

Closure under Hadamard Products

- Then

$$
\begin{aligned}
\langle \phi(x), \phi(w) \rangle &= \sum_{i,j} \phi(x)\phi(w) \\
&= \sum_{i,j} \left[\phi_1(x) \left[\phi_2(x)\right]^T \right]_{ij} \left[\phi_1(w) \left[\phi_2(w)\right]^T \right]_{ij} \\
&= \sum_{i,j} \left[\phi_1(x)\right]_i \left[\phi_2(x)\right]_j \left[\phi_1(w)\right]_i \left[\phi_2(w)\right]_j \\
&= \left(\sum_i \left[\phi_1(x)\right]_i \left[\phi_1(w)\right]_i \right) \left(\sum_j \left[\phi_2(x)\right]_j \left[\phi_2(w)\right]_j \right) \\
&= k_1(w,x) k_2(w,x)
\end{aligned}
$$

Kernel Machines

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

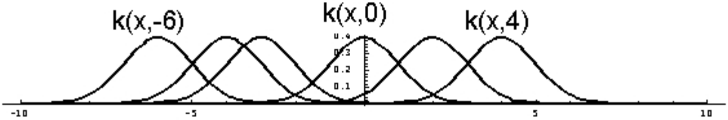
For each μ_i, we get a function on $\mathcal{X}$:

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

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

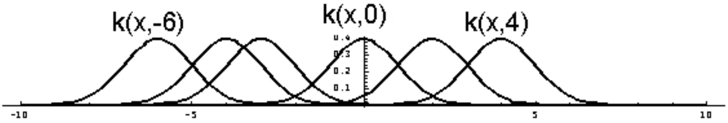
Kernel Machine Basis Functions

- 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:



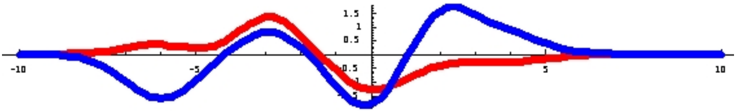
Kernel Machine Prediction Functions

- Basis functions



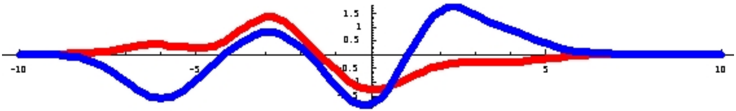
- Predictions of the form

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



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

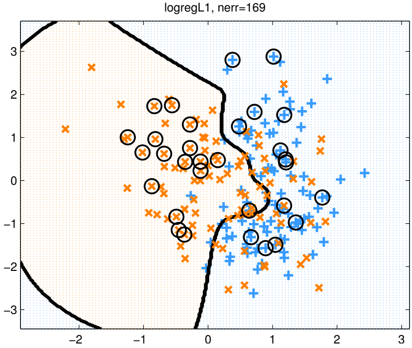
- Consider training inputs $x_1, \ldots, x_n \in \mathcal{X}$
- Then

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

- 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

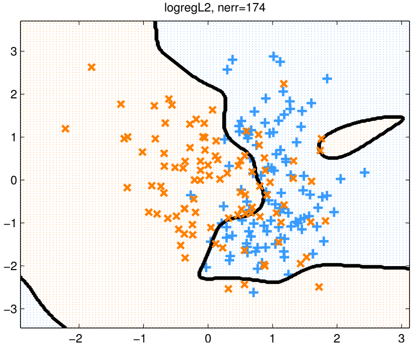
- RBF Kernel with bandwidth $\sigma = 0.3$.
- Linear hypothesis space: $\mathcal{F} = \{f(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i) \mid \alpha \in \mathbf{R}^n\}$.
- Logistic loss function: $\ell(y, \hat{y}) = \log\left(1 + e^{-y\hat{y}}\right)$
- ℓ_1-regularization, $n = 200$ training points



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

Example: Vector Machine for Ridge Regression

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:

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

- 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

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