Introduction to Kernel Methods

Ron Parr CPS 271

Outline

- · Motivation for kernel methods
- The dual view
- What makes a kernel?
- The Gaussian Process view
- Review of why we care
- Classification

Parametric Methods

- Methods discussed so far are parametric
- Learning mechanism/representation is characterized by a fixed set of parameters
 - Regression coefficients
 - Neural network weights
 - Linear discriminant means and covariances
- Connection between data and output is complicated and circuitous:
 - First we pick an algorithm, then our features,
 - Algorithm tunes parameters to data
 - Data are discarded
- Final result is a function from data to predictions
- Is there a more direct route?

Non-Parametric Methods

- Traditionally the simplest methods •
- Not discussed heavily in the book
- K-Nearest neighbor
- Classify according to nearest (in feature space) neighbor For k>1, vote
- Locally weighted regression
- Use query dependent regression weights (rerun regression for every query)
 Weights decay with distance from query point
- Issues
- Defining closeness (both scale and dimensions)
- Efficiency
- Tendency to overfit w/o careful tweaking

But how different are these, really?

- · Both are functions from data to predictions
- Non-parametric methods
 - Seem more constrained
 - Store less information
- Is there a unifying view?

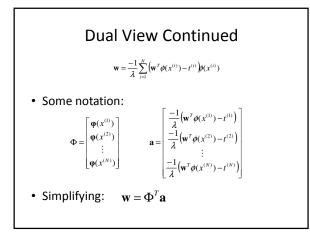
Dual View of Regression

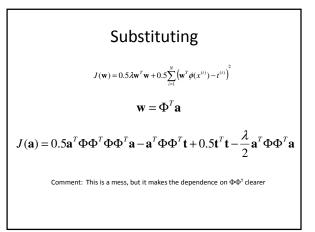
• Recall (regularized) regression minimizes:

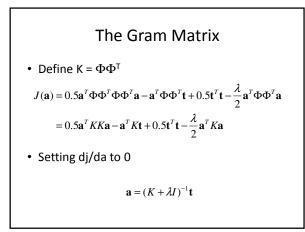
$$J(\mathbf{w}) = 0.5\lambda \mathbf{w}^T \mathbf{w} + 0.5\sum_{i=1}^N \left(\mathbf{w}^T \phi(x^{(i)}) - t^{(i)} \right)^T$$

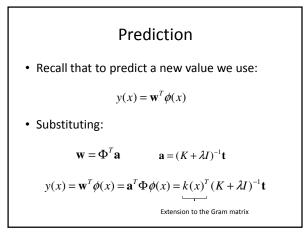
• Setting dJ/dw = 0:

$$\mathbf{v} = \frac{-1}{\lambda} \sum_{i=1}^{N} \left(\mathbf{w}^{T} \boldsymbol{\phi}(x^{(i)}) - t^{(i)} \right) \boldsymbol{\phi}(x^{(i)})$$









Why this matters?

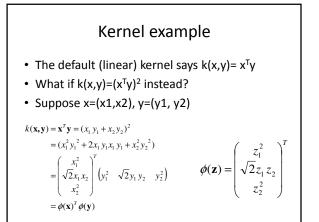
- We have expressed regression entirely in terms of the gram matrix, K, which we call the kernel
- But K is defined entirely of dot products between vectors in our training set
- Let's think about what K really means...

What does dot product mean?

- Elements of K are dot products between vectors, $\boldsymbol{x}^{T}\boldsymbol{y}$
- If x=y, x^Ty is squared magnitude
- If x has norm 1, then $x^T y$ is projection of y onto x
- In general $x^T y/(||x||||y||)$ is the cosine between x and y
- If x and y have the same magnitude, then x^Ty is maximized when x=y

What happens if we redefine dot product?

- If we change the meaning of x^Ty, then we change what it means to two vectors to be similar
- The big idea: Turn feature on its head
- Instead of invested effort in finding features, invest effort in finding kernels
- Algorithmically equivalent to redefining the K matrix



What just happened?

- We just changed the feature space from linear in x, to quadratic in x
- But we don't actually need to construct these features; we just need to redefine K
- Question: Does every choice of K make sense? (Does every K correspond to sensible basis?)

What makes a valid kernel?

- In general, we want our K matrix to be symmetric, positive semidefinite (not proved here)
- A *sufficient* (but not necessary) condition is for K to behave like a distance metric
 - Nonnegative
 - K(x,x)=0
 - Symmetric
 - Obeys triangle inequality
- Fancy kernels can be constructed by combining simple ones

Valid Kernel Combinations

Assumptions:

$$\begin{split} k(\mathbf{x},\mathbf{x}') &= ck_1(\mathbf{x},\mathbf{x}') \\ k(\mathbf{x},\mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x},\mathbf{x}')f(\mathbf{x}') \\ k(\mathbf{x},\mathbf{x}') &= q(k_1(\mathbf{x},\mathbf{x}')) \end{split}$$

 $k(\mathbf{x}, \mathbf{x}') = e^{k_1(\mathbf{x}, \mathbf{x}')}$ $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$ $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ $k(\mathbf{x}, \mathbf{x}') = k_3(\phi(\mathbf{x}), \phi(\mathbf{x}'))$ $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{A} \mathbf{x}'$

 $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b')$ $k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a')k_b(\mathbf{x}_b, \mathbf{x}_b')$ c>0 f(.) is any function q(.) is polynomial w/>=0 coefficients ki is any valid kernel function x=(xa,xb) A is symmetric and PSD

Other Interesting Kernels

• Polynomial kernel:

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^M$$

• Gaussian Kernel:

$$k(\mathbf{x}, \mathbf{x}') = e^{\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}}$$

More Kernels

• Kernels over sets:

 $k(A, A') = 2^{|A \cap A'|}$

• Kernels from probabilities

 $k(\mathbf{x}, \mathbf{x}') = p(\mathbf{x})p(\mathbf{x}')$

Gaussian Process View

- Recall that regularized regression is equivalent to the ML solution to Bayesian regression with a mean 0 Gaussian prior on the weights
- In the dual view, we interpret our training data as inducing a joint distribution over the y targets

Gaussian Process Formulation $\mathbf{y} = \Phi \mathbf{w}$ w is a vector of Gaussian rvs, so y is a vector of Gaussian rvs $E(\mathbf{y}) = \Phi E(\mathbf{w}) = 0$ $\exp(\mathbf{w}) - E(\mathbf{w}\mathbf{w}^T) = \Phi F(\mathbf{w}\mathbf{w}^T) \Phi^T = \frac{1}{-}\Phi \Phi^T = K$

$$\operatorname{cov}(\mathbf{y}) = E(\mathbf{y}\mathbf{y}') = \Phi E(\mathbf{w}\mathbf{w}')\Phi' = -\Phi\Phi' = \alpha$$

- Notes:
 Prior variance on w = α
 - We hid this in K

Noise model for targets

• Recall that we assume our targets values t, are the result of an underlying y, corrupted with Gaussian noise

 $p(\mathbf{y}) = N(\mathbf{y} \mid 0, \mathbf{K})$ $p(t \mid y) = N(t \mid y, \beta^{\prec})$ $p(t \mid \mathbf{y}) = N(t \mid \mathbf{y}, \beta^{\prec} \mathbf{I})$ $p(t) = \int p(t \mid \mathbf{y}) p(\mathbf{y}) d\mathbf{y} = N(t \mid 0, C)$ $C = \mathbf{K} + \beta^{\prec} \mathbf{I}$

Extending to Prediction

• For prediction, we want

$$p(\mathbf{t}_{N+1}) = N(\mathbf{t}_{N+1} | 0, C_{N+1})$$

$$C_{N+1} = \begin{pmatrix} C_N & k \\ k^T & c \end{pmatrix}$$

• Where C_N is our previous C, k is the kernel beween old (training) and new (x_{N+1}) points, and c = $k(x_{N+1}, x_{N+1})+\beta^{-1}$

Now we turn the magic crank

• All we need to do now is to condition on previously observed target values to get

 $P(t_{\scriptscriptstyle N+1} \,|\, \mathbf{t})$

• Using the conditioning results from chapter 2:

 $\mu(x_{N+1}) = \mathbf{k}^T C_N^{-1} \mathbf{t}$ $\sigma^2(x_{N+1}) = c - \mathbf{k}^T C_N^{-1} \mathbf{k}$

Compare with Regularized Regression

• Regularized regression with kernels:

$$y(x) = \mathbf{w}^T \phi(x) = \mathbf{a}^T \Phi \phi(x) = k(x)^T (K + \lambda I)^{-1} \mathbf{t}$$

• Gaussian process regression:

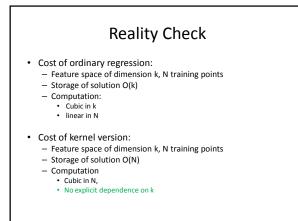
$$\mu(x_{N+1}) = \mathbf{k}^T C_N^{-1} \mathbf{t}$$
$$\sigma^2(x_{N+1}) = c - \mathbf{k}^T C_N^{-1} \mathbf{k}$$

Recall:

 $C = \mathbf{K} + \boldsymbol{\beta}^{-1}\mathbf{I}$

Advantages of the GP View

- Why bother with GP if regularized regression does the same thing?
- · GP also gives us a variance in prediction
- GP gives us a distribution over targets
- GP is more general, can incorporate other types of priors, including priors over priors



Why do we like Kernels?

- Let us experiment with feature spaces without paying the cost of constructing the features
- But what about overfitting? (Isn't k>N dangerous?)
- Yes! This is why we need regularization!
- This issue becomes particularly interesting in the context of support vector machines

Kernels for Classification

• Idea 1:

 Use logistic regression, replacing x^Tw with some kind of kernel regression

- Problem:

- No clean training algorithm
- Must use approximations
- Idea 2: Support Vector Machines (next chapter)

Relationship to Neural Networks

- Assume:
 - Linear output node
 - Large number of hidden nodes
 - Gaussian prior on weights
- Can show that in the limit of a large number of hidden nodes, the above neural network behaves like a Gaussian process

Some Concluding Thoughts

- Kernels and Gaussian processes have become very popular in recent years
- Why?

 - Coolness factor
 Ability to work in weird spaces implicitly
- Is this a good thing?
 - Not a substitute for standard linear + good features
 Some advantages over other non-parametric methods
 - Potential for elegant treatment of regularization
 GP provides probability distribution
 - Kernels useful for classification and other techniques beyond regression