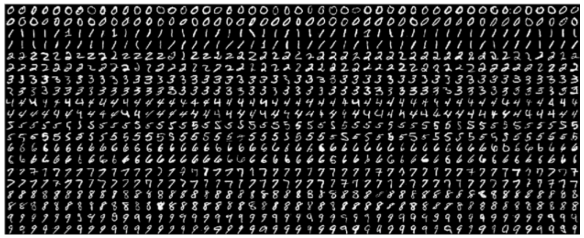
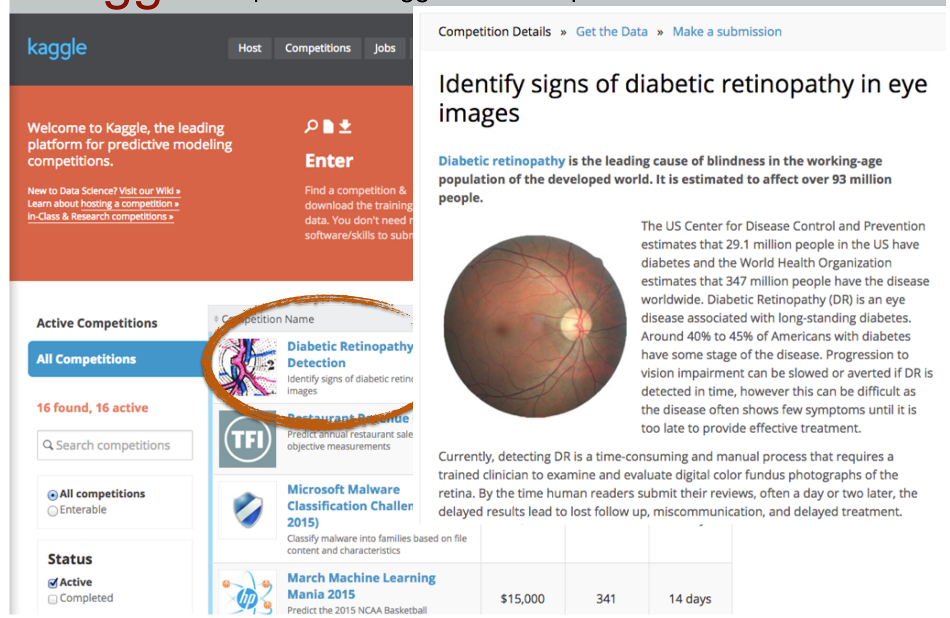
Administrivia

- Mini-project 2 due April 7, in class
  - implement multi-class reductions, naive bayes, kernel perceptron, multi-class logistic regression and two layer neural networks
  - training set: 
- Project proposals due April 2, in class
  - one page describing the project topic, goals, etc
  - list your team members (2+)
  - project presentations: April 23 and 27
  - final report: May 3

Kaggle https://www.kaggle.com/competitions



Kernel Methods

Subhransu Maji

CMPSCI 689: Machine Learning

24 March 2015

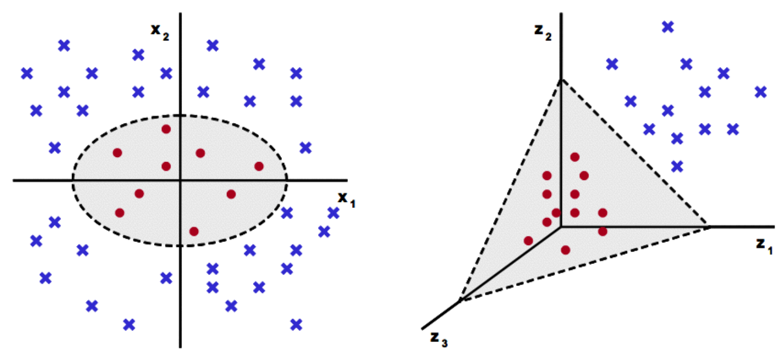
26 March 2015

Feature mapping

- Learn non-linear classifiers by mapping features

$$\varphi : R^2 \to R^3$$

$$(x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$$



Can we learn the XOR function with this mapping?

Quadratic feature map

- Let, $\mathbf{x} = [x_1, x_2, \ldots, x_D]$
- Then the quadratic feature map is defined as:

$$
\begin{aligned}
\phi(\mathbf{x}) = & [1, \sqrt{2}x_1, \sqrt{2}x_2, \ldots, \sqrt{2}x_D, \\
& x_1^2, x_1x_2, x_1x_3 \ldots, x_1x_D, \\
& x_2x_1, x_2^2, x_2x_3 \ldots, x_2x_D, \\
& \ldots, \\
& x_Dx_1, x_Dx_2, x_Dx_3 \ldots, x_D^2]
\end{aligned}
$$

- Contains all single and pairwise terms
- There are repetitions, e.g., x_1x_2 and x_2x_1, but hopefully the learning algorithm can handle redundant features

Drawbacks of feature mapping

- Computational
  - Suppose training time is linear in feature dimension, quadratic feature map squares the training time
- Memory
  - Quadratic feature map squares the memory required to store the training data
- Statistical
  - Quadratic feature mapping squares the number of parameters
  - For now lets assume that regularization will deal with overfitting

Quadratic kernel

- The dot product between feature maps of **x** and **z** is:

$$
\begin{aligned}
\phi(\mathbf{x})^T\phi(\mathbf{z}) &= 1 + 2x_1z_1 + 2x_2z_2, \ldots, 2x_Dz_D + x_1^2z_1^2 + x_1x_2z_1z_2 + \ldots + x_1x_Dz_1z_D + \ldots \\
& \ldots + x_Dx_1z_Dz_1 + x_Dx_2z_Dz_2 + \ldots + x_D^2z_D^2 \\
&= 1 + 2\left(\sum_i x_iz_i\right) + \sum_{i,j} x_ix_jz_iz_j \\
&= 1 + 2\left(\mathbf{x}^T\mathbf{z}\right) + \left(\mathbf{x}^T\mathbf{z}\right)^2 \\
&= \left(1 + \mathbf{x}^T\mathbf{z}\right)^2 \\
&= K(\mathbf{x}, \mathbf{z}) \longleftarrow \text{quadratic kernel}
\end{aligned}
$$

- Thus, we can compute $\phi(\mathbf{x})^T\phi(\mathbf{z})$ in almost the same time as needed to compute $\mathbf{x}^T\mathbf{z}$ (one extra addition and multiplication)
- We will rewrite various algorithms using only dot products (or kernel evaluations), and not explicit features

Perceptron revisited

Input: training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
feature map ϕ

Perceptron training algorithm

- Initialize $\mathbf{w} \leftarrow [0, \ldots, 0]$
- for iter = 1,…,T
  - for i = 1,..,n
    - predict according to the current model

$$
\hat{y}_i = \begin{cases} +1 & \text{if } \mathbf{w}^T\phi(\mathbf{x}_i) > 0 \\ -1 & \text{otherwise} \end{cases}
$$

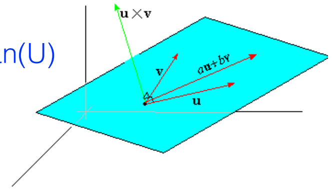
    - if $y_i = \hat{y}_i$, no change
    - else, $\mathbf{w} \leftarrow \mathbf{w} + y_i\phi(\mathbf{x}_i)$

dependence on ϕ

Obtained by replacing **x** by ϕ(**x**)

Properties of the weight vector

- Linear algebra recap:
  - Let U be set of vectors in R^D, i.e., U = $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_D\}$ and $\mathbf{u}_i \in R^D$
  - Span(U) is the set of all vectors that can be represented as $\sum_i a_i \mathbf{u}_i$, such that $a_i \in R$
  - Null(U) is everything that is left i.e., R^D \ Span(U)



Perceptron representer theorem: During the run of the perceptron training algorithm, the weight vector **w** is always in the span of $\phi(\mathbf{x}_1)$, $\phi(\mathbf{x}_1)$, …, $\phi(\mathbf{x}_D)$

$$\mathbf{w} = \textstyle\sum_i \alpha_i \phi(\mathbf{x}_i) \qquad \text{updates} \quad \alpha_i \leftarrow \alpha_i + y_i$$

$$\mathbf{w}^T \phi(\mathbf{z}) = \left(\textstyle\sum_i \alpha_i \phi(\mathbf{x}_i)\right)^T \phi(\mathbf{z}) = \textstyle\sum_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{z})$$

Kernelized perceptron

Input: training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
feature map ϕ

Kernelized perceptron training algorithm

- Initialize $\alpha \leftarrow [0, 0, \ldots, 0]$
- for iter = 1,…,T
  - for i = 1,..,n
    - predict according to the current model

$$\hat{y}_i = \begin{cases} +1 & \text{if } \sum_n \alpha_n \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_i) > 0 \\ -1 & \text{otherwise} \end{cases}$$

    - if $y_i = \hat{y}_i$, no change
    - else, $\alpha_i = \alpha_i + y_i$

$\phi(\mathbf{x})^T \phi(\mathbf{z}) = (1 + \mathbf{x}^T \mathbf{z})^p$ polynomial kernel of degree p

Support vector machines

- Kernels existed long before SVMs, but were popularized by them
- Does the representer theorem hold for SVMs?
- Recall that the objective function of an SVM is:

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \sum_n \max(0, 1 - y_n \mathbf{w}^T \mathbf{x}_n)$$

- Let, $\mathbf{w} = \mathbf{w}_{||} + \mathbf{w}_{\perp}$

only $\mathbf{w}_{||}$ affects classification

$$\begin{aligned} \mathbf{w}^T \mathbf{x}_i &= (\mathbf{w}_{||} + \mathbf{w}_{\perp})^T \mathbf{x}_i \\ &= \mathbf{w}_{||}^T \mathbf{x}_i + \mathbf{w}_{\perp}^T \mathbf{x}_i \\ &= \mathbf{w}_{||}^T \mathbf{x}_i \end{aligned}$$

norm decomposes

$$\begin{aligned} \mathbf{w}^T \mathbf{w} &= (\mathbf{w}_{||} + \mathbf{w}_{\perp})^T (\mathbf{w}_{||} + \mathbf{w}_{\perp}) \\ &= \mathbf{w}_{||}^T \mathbf{w}_{||} + \mathbf{w}_{\perp}^T \mathbf{w}_{\perp} \\ &\geq \mathbf{w}_{||}^T \mathbf{w}_{||} \end{aligned}$$

Hence, $\mathbf{w} \in \text{Span}(\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\})$

Kernel k-means

- Initialize k centers by picking k points randomly
- Repeat till convergence (or max iterations)
  - Assign each point to the nearest center (assignment step)

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \underline{\sum_{\mathbf{x} \in S_i}} ||\phi(\mathbf{x}) - \mu_i||^2$$

  - Estimate the mean of each group (update step)

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} ||\phi(\mathbf{x}) - \underline{\mu_i}||^2$$

- Representer theorem is easy here — $\mu_i \leftarrow \frac{1}{|S_i|} \sum_{\mathbf{x} \in S_i} \phi(\mathbf{x})$

- Exercise: show how to compute $||\phi(\mathbf{x}) - \mu_i||^2$ using dot products

What makes a kernel?

- A kernel is a mapping $\boldsymbol{K}\text{:}\ \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$
- Functions that can be written as dot products are valid kernels

$$K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

- Examples: polynomial kernel $K^d_{(\text{poly})}(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^T\mathbf{z})^d$

Alternate characterization of a kernel

- A function $\boldsymbol{K}\text{:}\ \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a kernel if $\boldsymbol{K}$ is positive semi-definite (psd)
- This property is also called as Mercer's condition
- This means that for all functions f that are squared integrable except the zero function, the following property holds:

$$\int\int f(\mathbf{x})K(\mathbf{x}, \mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} > 0 \qquad \int f(\mathbf{x})^2 d\mathbf{x} < \infty$$

Why is this characterization useful?

- We can prove some properties about kernels that are otherwise hard to prove
- Theorem: If K_1 and K_2 are kernels, then $K_1 + K_2$ is also a kernel
- Proof:

$$\begin{aligned}\int\int f(\mathbf{x})K(\mathbf{x}, \mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} &= \int\int f(\mathbf{x})\left(K_1(\mathbf{x}, \mathbf{z}) + K_2(\mathbf{x}, \mathbf{z})\right)f(\mathbf{z})d\mathbf{z}d\mathbf{x} \\ &= \int\int f(\mathbf{x})K_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} + \int\int f(\mathbf{x})K_2(\mathbf{x}, \mathbf{z})f(\mathbf{z})d\mathbf{z}d\mathbf{x} \\ &\geq 0 + 0\end{aligned}$$

- More generally if $K_1, K_2, \ldots, K_n$ are kernels then $\sum_i \alpha_i K_i$ with $\alpha_i \geq 0$, is a also a kernel
- Can build new kernels by linearly combining existing kernels

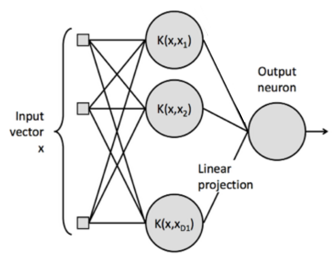
Why is this characterization useful?

- We can show that the Gaussian function is a kernel
  - Also called as radial basis function (RBF) kernel

$$K_{(\text{rbf})}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma||\mathbf{x} - \mathbf{z}||^2\right)$$

- Lets look at the classification function using a SVM with RBF kernel:

$$\begin{aligned}f(\mathbf{z}) &= \sum_i \alpha_i K_{(\text{rbf})}(\mathbf{x}_i, \mathbf{z}) \\ &= \sum_i \alpha_i \exp\left(-\gamma||\mathbf{x}_i - \mathbf{z}||^2\right)\end{aligned}$$



- This is similar to a two layer network with the RBF as the link function
- Gaussian kernels are examples of universal kernels — they can approximate any function in the limit as training data goes to infinity

Kernels in practice

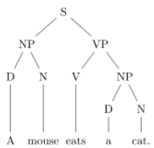
- Feature mapping via kernels often improves performance
- MNIST digits test error:
  - **8.4%** SVM linear
  - **1.4%** SVM RBF
  - **1.1%** SVM polynomial (d=4)

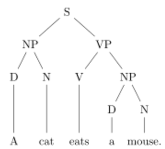
60,000 training examples

http://yann.lecun.com/exdb/mnist/

Kernels over general structures

- Kernels can be defined over any pair of inputs such as strings, trees and graphs!
- Kernel over trees:

K (

, 

) = number of common subtrees

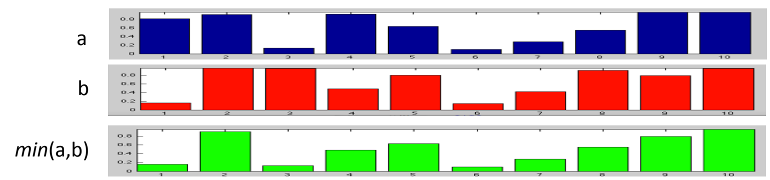
http://en.wikipedia.org/wiki/Tree_kernel

  - This can be computed efficiently using dynamic programming
  - Can be used with SVMs, perceptrons, k-means, etc
- For strings number of common substrings is a kernel
- Graph kernels that measure graph similarity (e.g. number of common subgraphs) have been used to predict toxicity of chemical structures

Kernels for computer vision

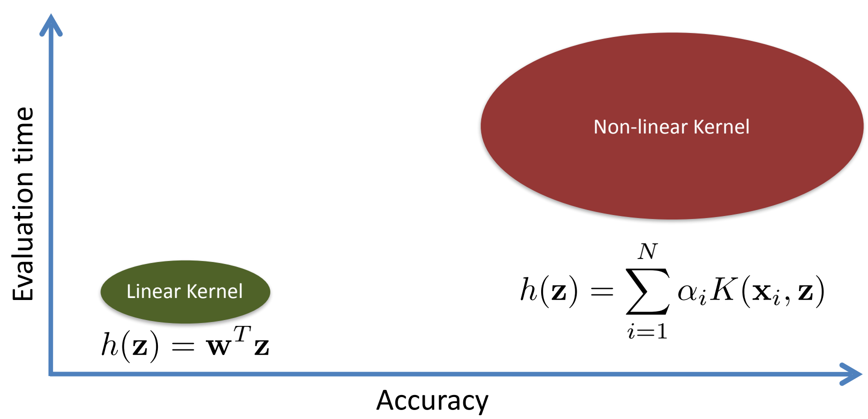
- Histogram intersection kernel between two histograms a and b

$$K_{\min}(a,b) = \sum_{i=1}^{n} \min(a_i, b_i) \quad a_i \geq 0 \quad b_i \geq 0$$



- Introduced by Swain and Ballard 1991 to compare color histograms

Kernel classifiers tradeoffs



Linear: O (feature dimension)
Non Linear: O (**N** X feature dimension)

Kernel classification function

$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

Kernel classification function

$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

Key insight: additive property

$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

$$= \sum_{j=1}^{D} \left(\sum_{i=1}^{N} \alpha_i \min(x_{ij}, z_j) \right)$$

$$= \sum_{j=1}^{D} h_j(z_j) \qquad h_j(z_j) = \sum_{i=1}^{N} \alpha_i \min(x_{ij}, z_j)$$

Kernel classification function

$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

Algorithm 1

$$h_j(z_j) = \sum_{i=1}^{N} \alpha_i \min(x_{ij}, z_j) \qquad O(N)$$

Kernel classification function

$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

[Maji et al. PAMI 13]

Algorithm 1

$$h_j(z_j) = \sum_{i=1}^{N} \alpha_i \min(x_{ij}, z_j) \qquad \cancel{O(N)}$$

$$= \sum_{i:x_{ij}<z_j}^{N} \alpha_i x_{ij} + \sum_{i:x_{ij}\geq z_j}^{N} \alpha_i z_j \qquad O(\log N)$$

Sort the support vector values in each coordinate, and pre-compute these sums for each rank.

To evaluate, find the position of z_j the sorted list of support vector x_{ij} .This can be done in O(log N) time using binary search.

Kernel classification function

$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

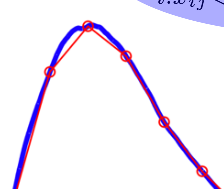
[Maji et al. PAMI 13]

Algorithm 2

$$h_j(z_j) = \sum_{i=1}^{N} \alpha_i \min(x_{ij}, z_j) \qquad \cancel{O(N)}$$

$$= \sum_{i:x_{ij}<z_j}^{N} \alpha_i x_{ij} + \sum_{i:x_{ij}\geq z_j}^{N} \alpha_i z_j \qquad \cancel{O(\log N)}$$

$$O(1)$$



For many problems h_i is smooth (blue plot). Hence, we can *approximate* it with fewer uniformly spaced segments (red plot). This saves time and space!

Kernel classification function

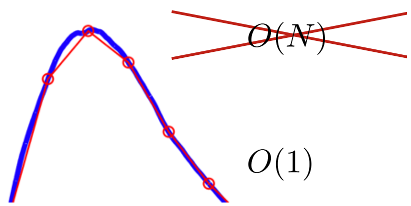
$$h(\mathbf{z}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}_i, \mathbf{z}) = \sum_{i=1}^{N} \alpha_i \left(\sum_{j=1}^{D} \min(x_{ij}, z_j) \right)$$

[Maji et al. PAMI 13]

Algorithm 2

$$K(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{D} k_i(x_i, z_i)$$

additive kernels



Intersection $k(a,b) = \min(a,b)$

Chi-squared $k(a,b) = \frac{2ab}{a+b}$

Jensen-Shannon $k(a,b) = a \log\left(\frac{a+b}{a}\right) + b \log\left(\frac{a+b}{b}\right)$

Linear and intersection kernel SVM

Using histograms of oriented gradients feature:

Dataset	Measure	Linear SVM	IK SVM	Speedup
INRIA pedestrians	Recall@ 2 FPPI	78.9	86.6	2594 X
DC pedestrians	Accuracy	72.2	89.0	2253 X
Caltech101, 15 examples	Accuracy	38.8	50.1	37 X
Caltech101, 30 examples	Accuracy	44.3	56.6	62 X
MNIST digits	Error	1.44	0.77	2500 X
UIUC cars (Single Scale)	Precision@ EER	89.8	98.5	65 X

On average **more accurate** than linear and **100-1000x** faster than standard kernel classifier. Similar idea can be applied to training as well. **Research question:** when can we approximate kernels efficiently?

Slides credit

- Some of the slides are based on CIML book by Hal Daume III
- Experiments on various datasets: “Efficient Classification for Additive Kernel SVMs”, S. Maji, A. C. Berg and J. Malik, PAMI, Jan 2013
- Some resources:
  - LIBSVM: kernel SVM classifier training and testing
    - http://www.csie.ntu.edu.tw/~cjlin/libsvm/
  - LIBLINEAR: fast linear classifier training
    - http://www.csie.ntu.edu.tw/~cjlin/liblinear/
  - LIBSPLINE: fast additive kernel training and testing
    - https://github.com/msubhransu/libspline