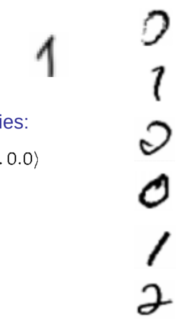
CS 188: Artificial Intelligence
Spring 2006

Lecture 14: Kernel Methods

3/2/2006

Dan Klein – UC Berkeley

Nearest-Neighbor Classification



- Nearest neighbor for digits:
  - Take new image
  - Compare to all training images
  - Assign based on closest example
- Encoding: image is vector of intensities:

 $= \langle 0.0\ 0.0\ 0.3\ 0.8\ 0.7\ 0.1 \ldots 0.0 \rangle$

- What's the similarity function?
  - Dot product of two images vectors?

$$\mathsf{sim}(x, y) = x \cdot y = \sum_i x_i y_i$$

  - min = 0 (when?), max = 1 (when?)

Today

- Kernels (Similarity Functions)
- Kernelized Perceptron
- Taste of Support Vector Machines

Basic Similarity

- Similarity based on feature dot products:

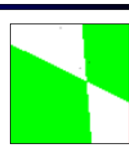
$$\mathsf{sim}(x, y) = f(x) \cdot f(y) = \sum_i f_i(x) f_i(y)$$

- If features are just the pixels:

$$\mathsf{sim}(x, y) = x \cdot y = \sum_i x_i y_i$$

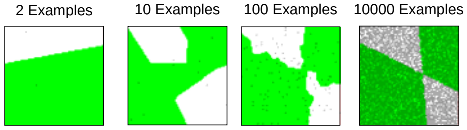
Recap: Nearest-Neighbor

- Nearest neighbor:
  - Classify test example based on closest training example
  - Requires a similarity function (kernel)
  - Eager learning: extract classifier from data
  - Lazy learning: keep data around and predict from it at test time



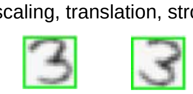
Truth

2 Examples | 10 Examples | 100 Examples | 10000 Examples



Invariant Metrics

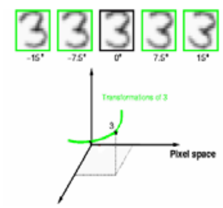
- Better distances use knowledge about vision
- Invariant metrics:
  - Similarities are invariant under certain transformations
  - Rotation, scaling, translation, stroke-thickness…
  - E.g:



    - 16 x 16 = 256 pixels; a point in 256-dim space
    - Small similarity in R^{256} (why?)
  - How to incorporate invariance into similarities?

This and next few slides adapted from Xiao Hu, UIUC

Rotation Invariant Metrics



- Each example is now a curve in R^{256}
- Rotation invariant similarity:

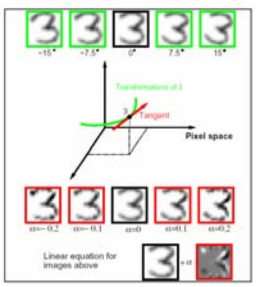
  s'=max s(r(), r())

- E.g. highest similarity between images' rotation lines

A Tale of Two Approaches…

- Nearest neighbor-like approaches
  - Can use fancy kernels (similarity functions)
  - Don't actually get to do explicit learning
- Perceptron-like approaches
  - Explicit training to reduce empirical error
  - Can't use fancy kernels (why not?)
  - Or can you? Let's find out!

Tangent Families



- Problems with s':
  - Hard to compute
  - Allows large transformations (6 → 9)
- Tangent distance:
  - 1st order approximation at original points.
    - Easy to compute
    - Models small rotations

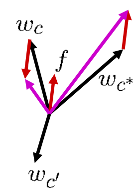
The Perceptron, Again

- Start with zero weights
- Pick up training instances one by one
- Try to classify

$$c = \arg\max_c \; w_c \cdot f(x)$$
$$= \arg\max_c \; \textstyle\sum_i w_{c,i} \cdot f_i(x)$$

- If correct, no change!
- If wrong: lower score of wrong answer, raise score of right answer

$$w_c = w_c - f(x)$$
$$w_{c^*} = w_{c^*} + f(x)$$



Template Deformation

- Deformable templates:
  - An "ideal" version of each category
  - Best-fit to image using min variance
  - Cost for high distortion of template
  - Cost for image points being far from distorted template
- Used in many commercial digit recognizers

Examples from [Hastie 94]

Perceptron Weights

- What is the final value of a weight w_c?
  - Can it be any real vector?
  - No! It's built by adding up inputs.

$$w_c = 0 + f(x_1) - f(x_5) + \ldots$$

$$w_c = \sum_i \alpha_{i,c} f(x_i)$$

- Can reconstruct weight vectors (the primal representation) from update counts (the dual representation)

$$\alpha_c = \langle \alpha_{1,c} \; \alpha_{2,c} \; \cdots \; \alpha_{n,c} \rangle$$

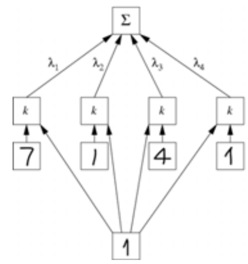
Dual Perceptron

- How to classify a new example x?

$$\begin{aligned}\text{score}(c,x) &= w_c \cdot f(x)\\ &= \left(\sum_i \alpha_{i,c} f(x_i)\right) \cdot f(x)\\ &= \sum_i \alpha_{i,c} \left(f(x_i) \cdot f(x)\right)\\ &= \sum_i \alpha_{i,c}\, K(x_i, x)\end{aligned}$$

- If someone tells us the value of K for each pair of examples, never need to build the weight vectors!

Kernelized Perceptron Structure



$$\sum = \text{score}(c,x)$$

$$\lambda_i = \alpha_{c,i}$$

Dual Perceptron

- Start with zero counts
- Pick up training instances one by one
- Try to classify x_n,

$$c = \arg\max_c \sum_i \alpha_{i,c}\, K(x_i, x)$$

- If correct, no change!
- If wrong: lower count of wrong class (for this instance), raise score of right class (for this instance)

$$\alpha_{c,n} = \alpha_{c,n} - 1 \qquad w_c = w_c - f(x)$$

$$\alpha_{c^*,n} = \alpha_{c^*,n} + 1 \qquad w_{c^*} = w_{c^*} + f(x)$$

Kernels: Who Cares?

- So far: a very strange way of doing a very simple calculation
- "Kernel trick": we can substitute any* similarity function in place of the dot product
- Lets us learn new kinds of hypothesis

* Fine print: if your kernel doesn't satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels *sometimes* work (but not always).

Kernelized Perceptron

- What if we had a black box (kernel) which told us the dot product of two examples x and y?
  - Could work entirely with the dual representation
  - No need to ever take dot products ("kernel trick")

$$\begin{aligned}\text{score}(c,x) &= w_c \cdot f(x)\\ &= \sum_i \alpha_{i,c}\, K(x_i, x)\end{aligned}$$

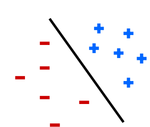
- Like nearest neighbor – work with black-box similarities
- Downside: slow if many examples get nonzero alpha

Properties of Perceptrons

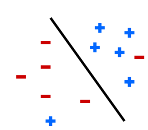
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

$$\text{mistakes} < \frac{1}{\delta^2}$$

Separable

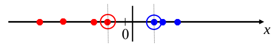


Non-Separable

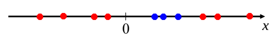


Non-Linear Separators

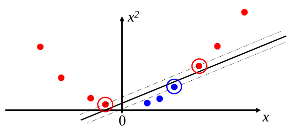
- Data that is linearly separable (with some noise) works out great:



- But what are we going to do if the dataset is just too hard?



- How about… mapping data to a higher-dimensional space:



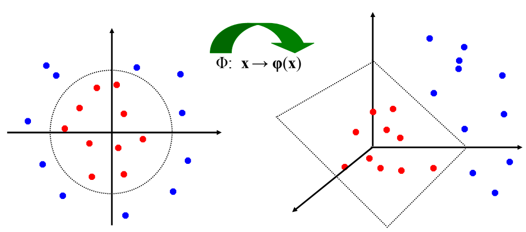
This and next few slides adapted from Ray Mooney, UT

Support Vector Machines

- Several (related) problems with perceptron
  - Can thrash around
  - No telling which separator you get
  - Once you make an update, can't retract it
- SVMs address these problems
  - Converge to globally optimal parameters
  - Good choice of separator (maximum margin)
  - Find sparse vectors (dual sparsity)

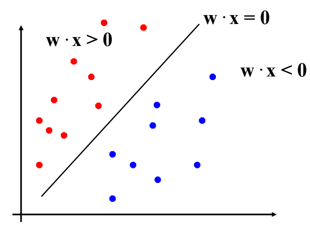
Non-Linear Separators

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



Linear Separators

- Binary classification can be viewed as the task of separating classes in feature space:



Some Kernels

- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back
- Linear kernel: $K(x, x') = x' \cdot x' = \sum_i x_i x'_i$
- Quadratic kernel: $K(x, x') = (x \cdot x' + 1)^2$

$$= \sum_{i,j} x_i x_j \, x'_i x'_j + 2 \sum_i x_i \, x'_i + 1$$

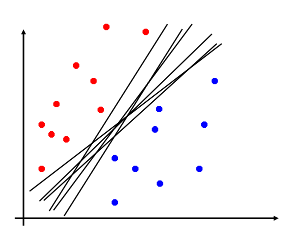
- RBF: infinite dimensional representation

$$K(x, x') = \exp(-||x - x'||^2)$$

- Discrete kernels: e.g. string kernels

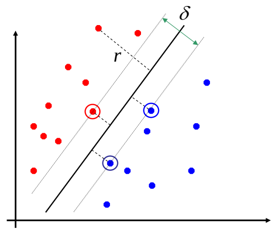
Linear Separators

- Which of the linear separators is optimal?



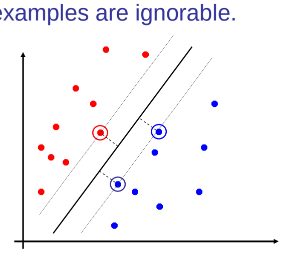
Classification Margin

- Distance from example $\mathbf{x}_i$ to the separator is r
- Examples closest to the hyperplane are support vectors.
- Margin δ of the separator is the distance between support vectors.



Maximum Margin Classification

- Maximizing the margin is good according to intuition and PAC theory.
- Implies that only support vectors matter; other training examples are ignorable.



Next Time

- Midterm: good luck!
- Speech Recognition and HMMs