CS 188: Artificial Intelligence
Fall 2008

Lecture 26: Perceptron & Kernels
4/23/2009

John DeNero – UC Berkeley
Slides from Dan Klein

Announcements

- Written assignment 4 due next Thursday
  - Printed copies available after class
- Lecture next Tuesday: Dan Klein
  - Natural language processing
  - Great teacher who made most of my slides
- Lecture next Thursday: Pieter Abbeel
  - Robotics and control
  - Builds helicopters that fly on their own

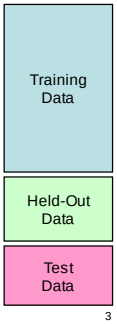
[Email from Hal]

Mistake-Driven Classification

- The multi-class perceptron has a weight vector w_y for each label value y
- The label is predicted by:

$$y = \arg\max_y \; w_y \cdot f(x)$$

- Predicts the label with highest activation
- Weights are updated whenever the classifier makes a mistake on a training datum (whenever predicted y is not y*)

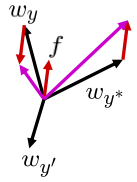


The Perceptron Update Rule

- Start with zero weights
- Pick up training instances one by one
- Classify with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$
$$= \arg\max_y \; \sum_i w_{y,i} \cdot f_i(x)$$

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

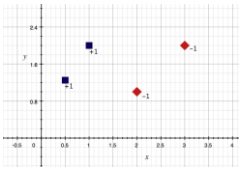


$$w_y = w_y - f(x)$$
$$w_{y^*} = w_{y^*} + f(x)$$

Only the weights of non-zero features change!

Perceptron Example

Features	Label
(0.5,1.25)	+1
(1,2)	+1
(2,1)	-1
(3,2)	-1



Obvious problems with the perceptron

- Sometimes updates too much
  - Good weights can be corrupted by a single outlier datum
- Sometimes updates too little
  - Even after an update, the prediction can still be incorrect
- Assumes separable data
  - Real data is never separable

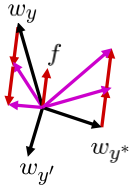
Fixing the Perceptron

- Idea: adjust the size of the weight update to mitigate these effects
- MIRA*: choose an update size that fixes the current mistake…
- … but, minimizes the change to w

$$\min_w \; \frac{1}{2} \sum_y ||w_y - w'_y||^2$$

$$w_{y^*} \cdot f(x) \geq w_y \cdot f(x) + 1$$

- The +1 helps to generalize



Guessed y instead of y^* on example x with features $f(x)$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$

* Margin Infused Relaxed Algorithm

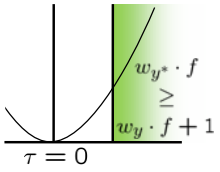
Minimum Correcting Update

$$\min_w \frac{1}{2}\sum_y ||w_y - w'_y||^2$$
$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$\min_\tau ||\tau f||^2$$
$$w_{y^*} \cdot f \geq w_y \cdot f + 1$$

$$(w'_{y^*} + \tau f) \cdot f = (w'_y - \tau f) \cdot f + 1$$
$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}$$

$$w_y = w'_y - \tau f(x)$$
$$w_{y^*} = w'_{y^*} + \tau f(x)$$



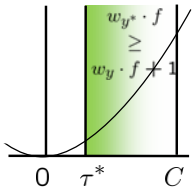
min not τ=0, or would not have made an error, so min will be where equality holds

MIRA

- In practice, it's bad to make updates that are too large
  - Example may be labeled incorrectly
  - Solution: cap the maximum possible value of τ with some constant C

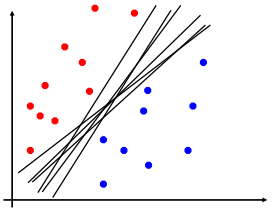
$$\tau^* = \min\left(\frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C\right)$$

  - Corresponds to an optimization that assumes non-separable data
  - Usually converges faster than perceptron
  - Usually performs better, especially on noisy data



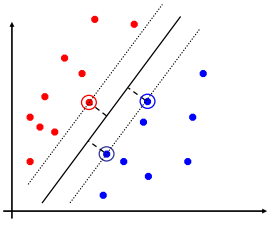
Linear Separators

- Which of these linear separators is optimal?



Support Vector Machines

- Maximizing the margin: good according to intuition and theory.
- Only support vectors matter; other training examples are ignorable.
- Support vector machines (SVMs) find the separator with max margin
- Basically, SVMs are MIRA where you optimize over all examples at once



MIRA

$$\min_w \frac{1}{2}||w - w'||^2$$
$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

SVM

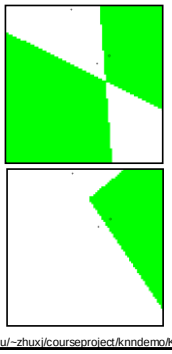
$$\min_w \frac{1}{2}||w||^2$$
$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Summary

- Naïve Bayes
  - Build classifiers using model of training data
  - Smoothing estimates is important in real systems
  - One pass through data
- Perceptrons / MIRA:
  - Make less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data

Case-Based Reasoning

- Similarity for classification
  - Case-based reasoning
  - Predict an instance's label using similar instances
- Nearest-neighbor classification
  - 1-NN: copy the label of the most similar data point
  - K-NN: let the k nearest neighbors vote (have to devise a weighting scheme)
  - Key issue: how to define similarity
  - Trade-off:
    - Small k gives relevant neighbors
    - Large k gives smoother functions
    - Sound familiar?
- [Demo]



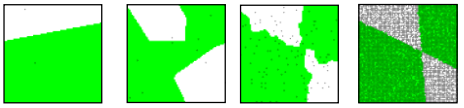
http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html

Parametric / Non-parametric

- Parametric models:
  - Fixed set of parameters
  - More data means better settings
- Non-parametric models:
  - Complexity of the classifier increases with data
  - Better in the limit, often worse in the non-limit
- (K)NN is non-parametric

Truth

2 Examples | 10 Examples | 100 Examples | 10000 Examples



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?

$$\text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i$$

  - Usually normalize vectors so ||x|| = 1
  - min = 0 (when?), max = 1 (when?)

Basic Similarity

- Many similarities based on feature dot products:

$$\text{sim}(x, x') = f(x) \cdot f(x') = \sum_i f_i(x) f_i(x')$$

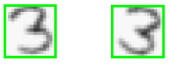
- If features are just the pixels:

$$\text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i$$

- Note: not all similarities are of this form

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

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]

A Tale of Two Approaches…

- Nearest neighbor-like approaches
  - Can use fancy similarity functions
  - Don't actually get to do explicit learning
- Perceptron-like approaches
  - Explicit training to reduce empirical error
  - Can't use fancy similarity, only linear
  - Or can they? Let's find out!

Perceptron Weights

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

$$w_y = 0 + f(x_1) - f(x_5) + \dots$$

$$w_y = \sum_i \alpha_{i,y} f(x_i)$$

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

$$\alpha_y = \langle \alpha_{1,y} \ \alpha_{2,y} \ \dots \ \alpha_{n,y} \rangle$$

Dual Perceptron

- How to classify a new example x?

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

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

Dual Perceptron

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

$$y = \arg\max_y \sum_i \alpha_{i,y} K(x_i, x)$$

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

$$\alpha_{y,n} = \alpha_{y,n} - 1 \qquad w_y = w_y - f(x)$$

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

Kernelized Perceptron

- 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")

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

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

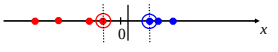
Kernels: Who Cares?

- 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

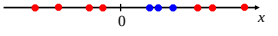
* Fine print: if your kernel doesn't satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels *sometimes* work (but not always).

Non-Linear Separators

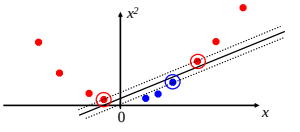
- Data that is linearly separable (with some noise) works out great:



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



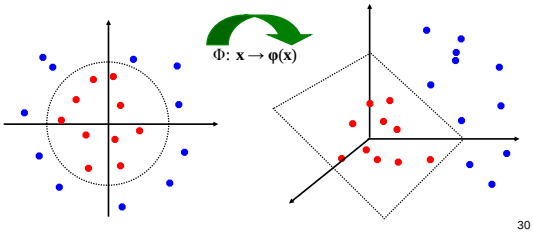
- How about… mapping data to a higher-dimensional space:



This and next few slides adapted from Ray Mooney, UT

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:



30

Why Kernels?

- Can't you just add these features on your own (e.g. add all pairs of features instead of using the quadratic kernel)?
  - Yes, in principle, just compute them
  - No need to modify any algorithms
  - But, number of features can get large (or infinite)
  - Some kernels not as usefully thought of in their expanded representation, e.g. RBF or data-defined kernels [Henderson and Titov05]
- Kernels let us compute with these features implicitly
  - Example: implicit dot product in quadratic kernel takes much less space and time per dot product
  - Of course, there's the cost for using the pure dual algorithms: you need to compute the similarity to every training datum