CS 188: Artificial Intelligence
Spring 2010

Lecture 22: Nearest Neighbors, Kernels
4/18/2011

Pieter Abbeel – UC Berkeley
Slides adapted from Dan Klein

Announcements

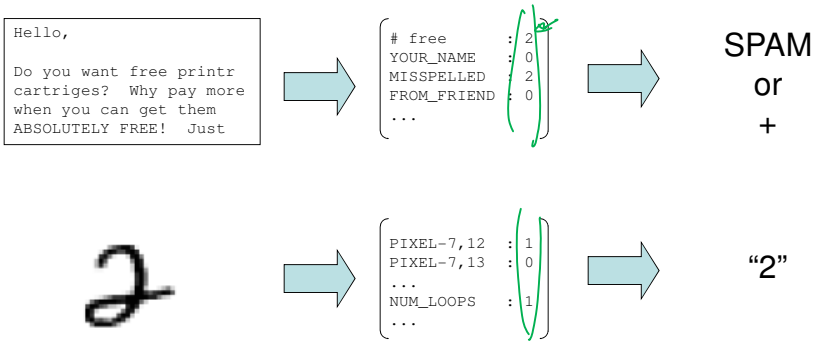
- On-going: contest (optional and FUN!)
- Remaining lectures:
  - Today: Machine Learning: Nearest Neighbors, Kernels
  - Wednesday: Machine Learning for Computer Vision
  - Next Monday: Case Studies in Speech/Language and Robotics
  - Next Wednesday:
    - Course Wrap-Up
    - Pointers to courses and Books for those who want to learn more AI
    - Contest!
  - RRR Week Monday and Wednesday: Review Sessions

Today

- Nearest neighbors
- Kernels
- Applications:
  - Extension to ranking / web-search
  - Pacman apprenticeship

Classification

x $\quad\quad$ $f(x)$ $\longrightarrow$ y

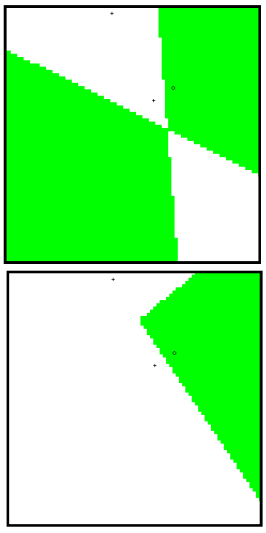


Classification overview

- Naïve Bayes:
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)
- Perceptron:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate
- MIRA:
  - Like perceptron, but adaptive scaling of size of update
- SVM:
  - Properties similar to perceptron
  - Convex optimization formulation
- Nearest-Neighbor:
  - Non-parametric: more expressive with more training data
- Kernels
  - Efficient way to make linear learning architectures into nonlinear ones

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]

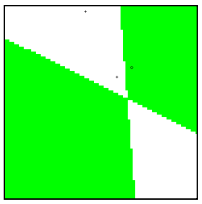


6

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

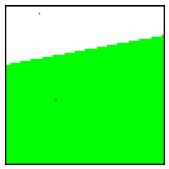
Parametric / Non-parametric

- 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

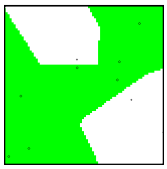


Truth

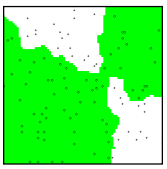
2 Examples



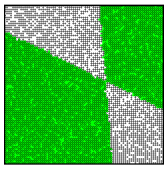
10 Examples



100 Examples



10000 Examples



7

Nearest-Neighbor Classification

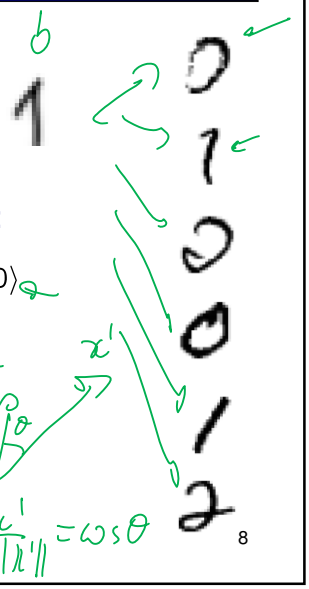
- Nearest neighbor for digits:
  - Take new image
  - Compare to all training images
  - Assign based on closest example
- Encoding: image is vector of intensities:

$$1 = \langle 0.0 \;\; 0.0 \;\; 0.3 \;\; 0.8 \;\; 0.7 \;\; 0.1 \ldots 0.0 \rangle$$

- 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?)



8

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

9

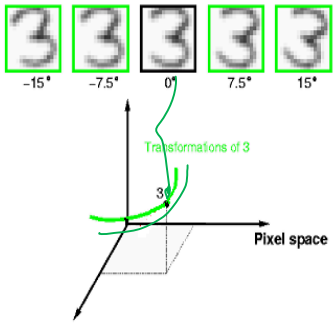
Invariant Metrics

- 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?)
  - Variety of invariant metrics in literature
- Viable alternative: transform training examples such that training set includes all variations

10

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

11

Classification overview

- Naïve Bayes
- Perceptron, MIRA
- SVM
- Nearest-Neighbor
- ***Kernels***

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!

16

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)

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

17

Dual Perceptron

- How to classify a new example x?

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

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

18

Dual Perceptron

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

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

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

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

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

dual primal

19

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")

$$\text{score}(y, x) = w_y \cdot f(x)$$

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

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

20

Kernelized MIRA

$$\alpha_{y,n} = \alpha_{y,n} - \tau$$

$$\alpha_{y^*,n} = \alpha_{y^*,n} + \tau$$

$$w_y = w'_y - \tau f(x)$$

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

primal

- Our formula for τ (see last lecture)

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

primal

$$\rightarrow \tau^* = \min\left(\frac{\sum_i \alpha_{i,y} K(x^{(i)}, x) - \sum_i \alpha_{i,y} K(x^{(i)}, x) + 1}{2K(x, x)}, C\right)$$

dual

$= w'_y \cdot f - w'_{y^*} \cdot f$

22

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

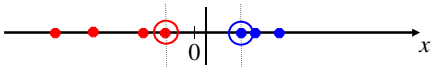
$K(x, x') = \phi(x) \cdot \phi(x')$

* 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).

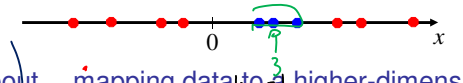
23

Non-Linear Separators

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



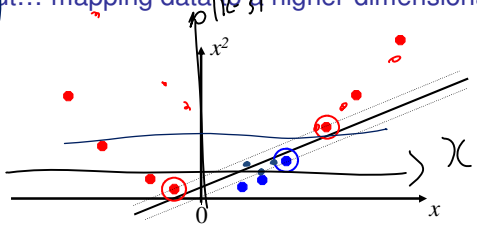
- But what are we going to do if the dataset is just too hard?

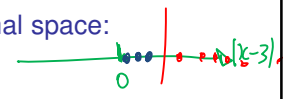


$f(x) = |x-3|$

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

$\phi(x) = \begin{pmatrix} x \\ |x-3| \end{pmatrix}$



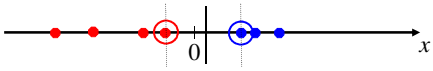


24

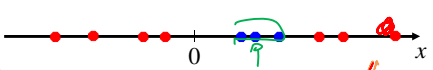
This and next few slides adapted from Ray Mooney, UT

Non-Linear Separators

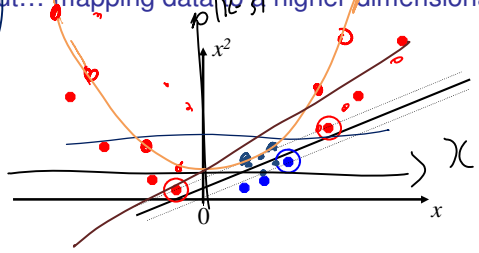
- Data that is linearly separable (with some noise) works out great:



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



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

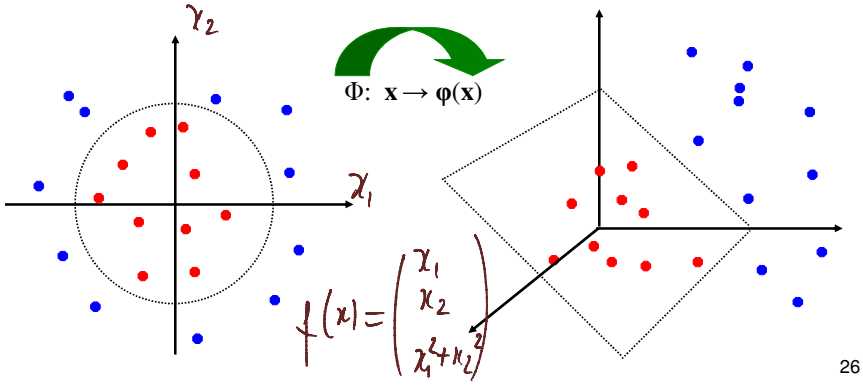


25

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:



26

Some Kernels

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

  $\phi(x) = x$
- Quadratic kernel: $K(x, x') = (x \cdot x' + 1)^2$

  $= \sum x_i x_j x'_i x'_j + 2 \sum x_i x'_i + 1$

For $x \in \Re^3$:

$\phi(x) = [x_1 x_1 \; x_1 x_2 \; x_1 x_3 \; x_2 x_1 \; x_2 x_2 \; x_2 x_3 \; x_3 x_1 \; x_3 x_2 \; x_3 x_3 \; \sqrt{2}x_1 \; \sqrt{2}x_2 \; \sqrt{2}x_3 \; 1]$

Some Kernels (2)

- Polynomial kernel: $K(x, x') = (x \cdot x' + 1)^d$

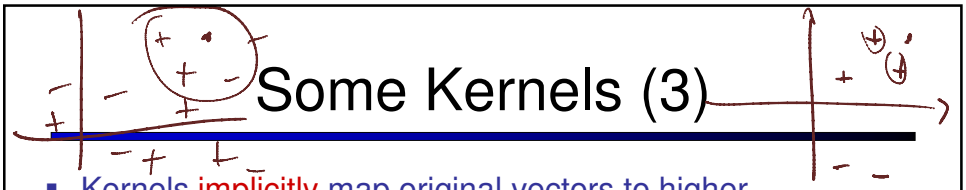
For $x \in \Re^3$:

$\phi(x) = [x_1^d \; x_2^d \; x_3^d \; \sqrt{d}x_1^{d-1}x_2 \; \sqrt{d}x_1^{d-1}x_3 \; \ldots \; \sqrt{d}x_1 \; \sqrt{d}x_2 \; \sqrt{d}x_3 \; 1]$

For $x \in \Re^n$ the d-order polynomial kernel's implicit feature space is $\binom{n+d}{d}$ dimensional.

By contrast, computing the kernel directly only requires $O(n)$ time.

Some Kernels (3)



- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back

- Radial Basis Function (or Gaussian) Kernel: infinite dimensional representation

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{\lambda^2}\right)$$

- Discrete kernels: e.g. string kernels
  - Features: all possible strings up to some length
  - To compute kernel: don't need to enumerate all substrings for each word, but only need to find strings appearing in both x and x'

29

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)

- Kernels let us compute with these features implicitly
  - Example: implicit dot product in polynomial, Gaussian and string 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

Recap: Classification

- Classification systems:
  - Supervised learning
  - Make a prediction given evidence
  - We've seen several methods for this
  - Useful when you have labeled data

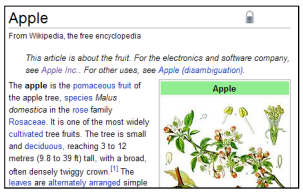
32

Extension: Web Search

- Information retrieval:
  - Given information needs, produce information
  - Includes, e.g. web search, question answering, and classic IR

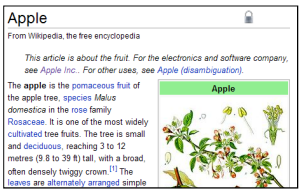
- Web search: not exactly classification, but rather ranking

x = "Apple Computers"



Feature-Based Ranking

x = "Apple Computers"

$w \cdot f($ $x,$  $) = [0.3\ 5\ 0\ 0\ \ldots]$

$w \cdot f($ $x,$  $) = [0.8\ 4\ 2\ 1\ \ldots]$

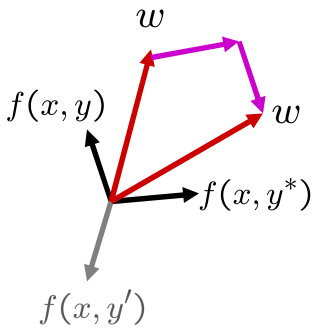
Perceptron for Ranking

- Inputs x
- Candidates y
- Many feature vectors: $f(x, y)$
- One weight vector: w
  - Prediction:

$$y = \arg\max_y \ w \cdot f(x, y)$$

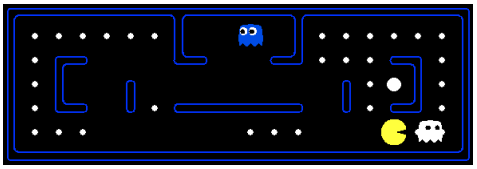
  - Update (if wrong):

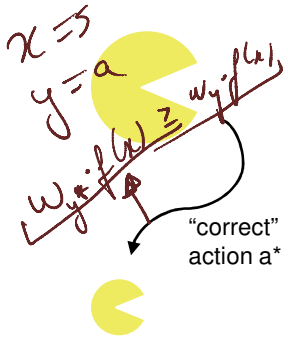
$$w = w + f(x, y^*) - f(x, y)$$



Pacman Apprenticeship!

- Examples are states s





- Candidates are pairs (s,a)
- "Correct" actions: those taken by expert
- Features defined over (s,a) pairs: f(s,a)
- Score of a q-state (s,a) given by:

$$w \cdot f(s,a)$$

$$\forall a \neq a^*, \; w \cdot f(a^*) > w \cdot f(a)$$

- How is this VERY different from reinforcement learning?