

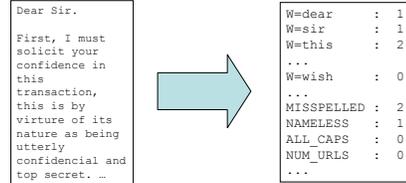
CS 188: Artificial Intelligence Fall 2007

Lecture 26: Kernels 11/29/2007

Dan Klein – UC Berkeley

Feature Extractors

- A feature extractor maps inputs to feature vectors



- Many classifiers take feature vectors as inputs
- Feature vectors usually very sparse, use sparse encodings (i.e. only represent non-zero keys)

The Perceptron Update Rule

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

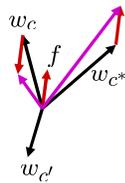
$$c = \arg \max_c w_c \cdot f(x)$$

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

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

$$w_c = w_c - f(x)$$

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



Nearest-Neighbor Classification

- Nearest neighbor for digits:
 - Take new image
 - Compare to all training images
 - Assign based on closest example

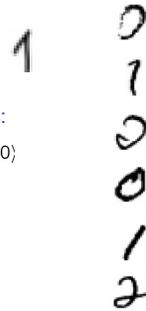
- Encoding: image is vector of intensities:

$$1 = (0.0 \ 0.0 \ 0.3 \ 0.8 \ 0.7 \ 0.1 \ \dots \ 0.0)$$

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

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

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



Basic Similarity

- Many similarities based on feature dot products:

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

- If features are just the pixels:

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

- 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...

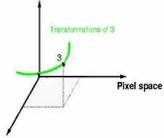
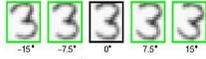
- E.g:



- 16 x 16 = 256 pixels; a point in 256-dim space
- Small similarity in \mathbb{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 \mathbb{R}^{256}
- Rotation invariant similarity:

$$s' = \max_s (r(\text{3}), r(\text{3}))$$
- E.g. highest similarity between images' rotation lines

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 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!

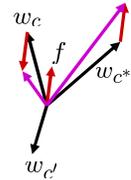
The Perceptron, Again

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

$$c = \arg \max_c w_c \cdot f(x)$$

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

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



$$w_c = w_c - f(x)$$

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

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.

$$w_c = \mathbf{0} + f(x_1) - f(x_5) + \dots$$

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

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

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

Dual Perceptron

- How to classify a new example x ?

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

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

Dual Perceptron

- Start with zero counts (alpha)
- 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)

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

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

Kernelized Perceptron

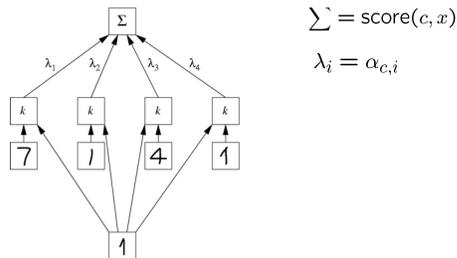
- If we had a black box (**kernel**) which told us the dot product of two examples x and y :
 - Could work entirely with the dual representation
 - No need to ever take dot products ("kernel trick")

$$\text{score}(c, x) = w_c \cdot f(x)$$

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

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

Kernelized Perceptron Structure



Kernels: Who Cares?

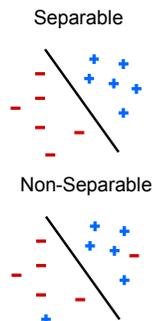
- So far: a very strange way of doing a very simple calculation
- "Kernel trick": we can substitute any* similarity function in place of the dot product
- Lets us learn new kinds of hypothesis

* 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).

Properties of Perceptrons

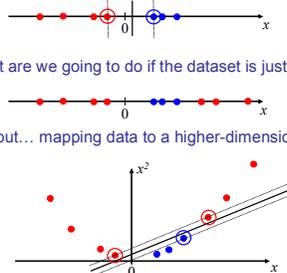
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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



Non-Linear Separators

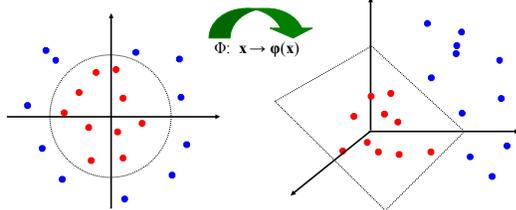
- Data that is linearly separable (with some noise) works out great:
- But what are we going to do if the dataset is just too hard?
- How about... mapping data to a higher-dimensional space:



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:



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

Recap: Classification

- Classification systems:
 - Supervised learning**
 - Make a rational **prediction** given evidence
 - We've seen several methods for this
 - Useful when you have **labeled data** (or can get it)



Clustering

- Clustering systems:
 - Unsupervised learning**
 - Detect patterns** in unlabeled data
 - E.g. group emails or search results
 - E.g. find categories of customers
 - E.g. detect anomalous program executions
 - Useful when don't know what you're looking for
 - Requires data, but no labels
 - Often get gibberish



Clustering

- Basic idea: group together similar instances
- Example: 2D point patterns

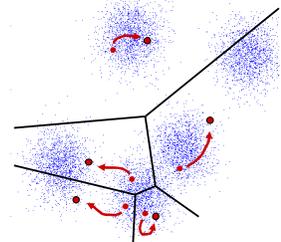


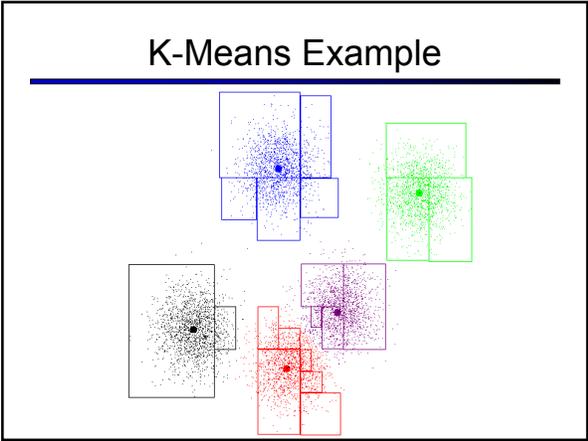
- What could "similar" mean?
 - One option: small (squared) Euclidean distance

$$\text{dist}(x, y) = (x - y)^T (x - y) = \sum_i (x_i - y_i)^2$$

K-Means

- An iterative clustering algorithm
 - Pick K random points as cluster centers (means)
 - Alternate:
 - Assign data instances to closest mean
 - Assign each mean to the average of its assigned points
 - Stop when no points' assignments change





K-Means as Optimization

- Consider the total distance to the means:

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

points assignments means
- Each iteration reduces phi
- Two stages each iteration:
 - Update assignments: fix means c, change assignments a
 - Update means: fix assignments a, change means c

Phase I: Update Assignments

- For each point, re-assign to closest mean:

$$a_i = \underset{k}{\text{argmin}} \text{dist}(x_i, c_k)$$
- Can only decrease total distance phi!

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

Phase II: Update Means

- Move each mean to the average of its assigned points:

$$c_k = \frac{1}{|\{i : a_i = k\}|} \sum_{i: a_i=k} x_i$$
- Also can only decrease total distance... (Why?)
- Fun fact: the point y with minimum squared Euclidean distance to a set of points {x} is their mean

Initialization

- K-means is non-deterministic
 - Requires initial means
 - It does matter what you pick!
 - What can go wrong?
 - Various schemes for preventing this kind of thing: variance-based split / merge, initialization heuristics

K-Means Getting Stuck

- A local optimum:

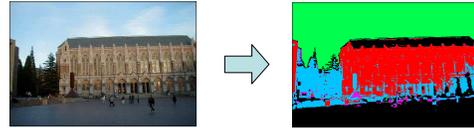
Why doesn't this work out like the earlier example, with the purple taking over half the blue?

K-Means Questions

- Will K-means converge?
 - To a global optimum?
- Will it always find the true patterns in the data?
 - If the patterns are very very clear?
- Will it find something interesting?
- Do people ever use it?
- How many clusters to pick?

Clustering for Segmentation

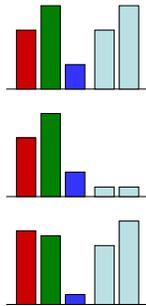
- Quick taste of a simple vision algorithm
- Idea: break images into manageable regions for visual processing (object recognition, activity detection, etc.)



<http://www.cs.washington.edu/research/imagetatabase/demo/kmcluster/>

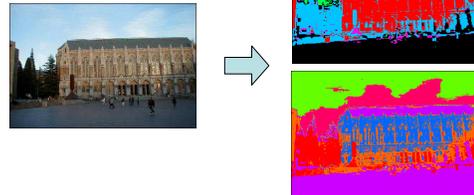
Representing Pixels

- Basic representation of pixels:
 - 3 dimensional color vector $\langle r, g, b \rangle$
 - Ranges: r, g, b in $[0, 1]$
 - What will happen if we cluster the pixels in an image using this representation?
- Improved representation for segmentation:
 - 5 dimensional vector $\langle r, g, b, x, y \rangle$
 - Ranges: x in $[0, M]$, y in $[0, N]$
 - Bigger M, N makes position more important
 - How does this change the similarities?
- Note: real vision systems use more sophisticated encodings which can capture intensity, texture, shape, and so on.



K-Means Segmentation

- Results depend on initialization!
 - Why?



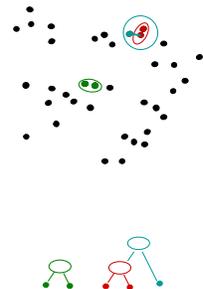
- Note: best systems use graph segmentation algorithms

Other Uses of K-Means

- Speech recognition: can use to quantize wave slices into a small number of types (SOTA: work with multivariate continuous features)
- Document clustering: detect similar documents on the basis of shared words (SOTA: use probabilistic models which operate on topics rather than words)

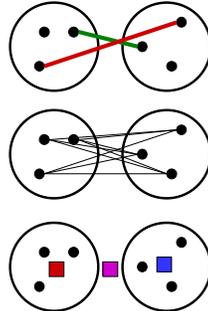
Agglomerative Clustering

- Agglomerative clustering:
 - First merge very similar instances
 - Incrementally build larger clusters out of smaller clusters
- Algorithm:
 - Maintain a set of clusters
 - Initially, each instance in its own cluster
 - Repeat:
 - Pick the two **closest** clusters
 - Merge them into a new cluster
 - Stop when there's only one cluster left
- Produces not one clustering, but a family of clusterings represented by a **dendrogram**



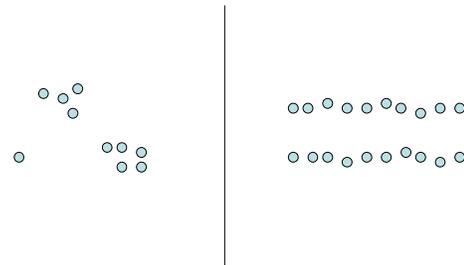
Agglomerative Clustering

- How should we define "closest" for clusters with multiple elements?
- Many options
 - Closest pair (single-link clustering)
 - Farthest pair (complete-link clustering)
 - Average of all pairs
 - Distance between centroids (broken)
 - Ward's method (my pick, like k-means)
- Different choices create different clustering behaviors



Agglomerative Clustering

- Complete Link (farthest) vs. Single Link (closest)



Back to Similarity

- K-means naturally operates in Euclidean space (why?)
- Agglomerative clustering didn't require any mention of averaging
 - Can use any function which takes two instances and returns a similarity
 - Kernelized clustering: if your similarity function has the right properties, can adapt k-means too
- Kinds of similarity functions:
 - Euclidian (dot product)
 - Weighted Euclidian
 - Edit distance between strings
 - Anything else?

Collaborative Filtering

- Ever wonder how online merchants decide what products to recommend to you?
- Simplest idea: recommend the most popular items to everyone
 - Not entirely crazy! (Why)
 - Can do better if you know something about the customer (e.g. what they've bought)
- Better idea: recommend items that similar customers bought
 - A popular technique: collaborative filtering
 - Define a similarity function over customers (how?)
 - Look at purchases made by people with high similarity
 - Trade-off: relevance of comparison set vs confidence in predictions
 - How can this go wrong?

