# CS 343H: Honors AI

Lecture 23: Kernels and clustering

4/15/2014

Kristen Grauman

UT Austin

Slides courtesy of Dan Klein, except where otherwise noted

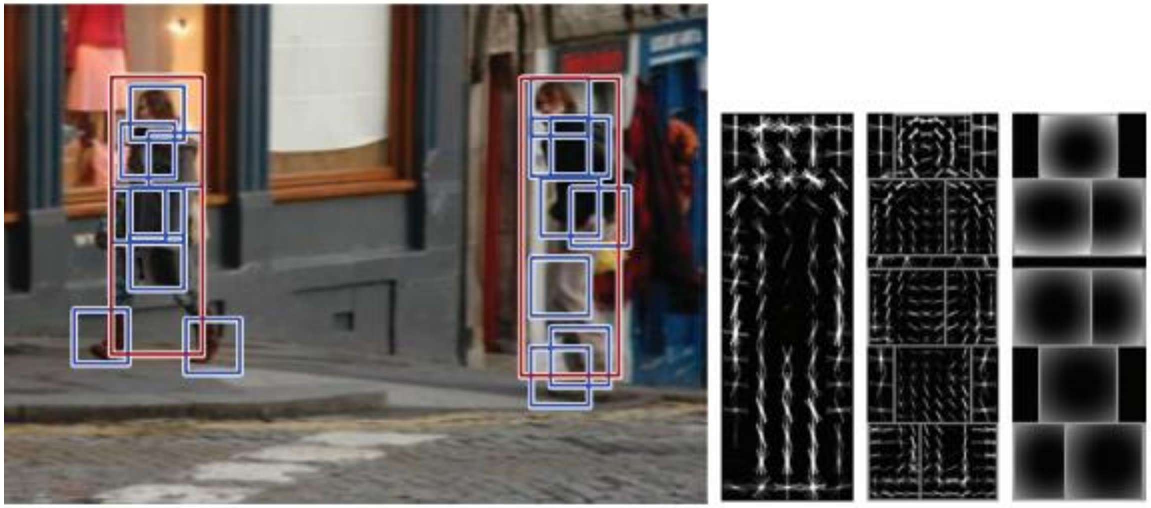
# Announcements

- Office hours
  - Kim’s office hours this week:
    - Mon 11-12 and Thurs 12:30-1:30 pm
  - No office hours Tues – contact me

- Class on Thursday 4/17 meets in GDC 2.216 (Auditorium)
  - See class page for associated reading assignment

# Thursday 4/17, 11 am

- Prof. Deva Ramanan, UC Irvine
- “Statistical analysis by synthesis: visual recognition through reconstruction”

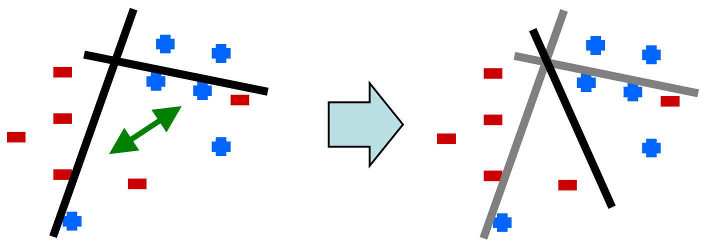


# Today

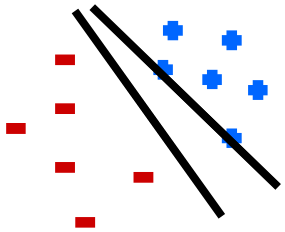
- Perceptron wrap-up
- Kernels and clustering

# Recall: Problems with the Perceptron

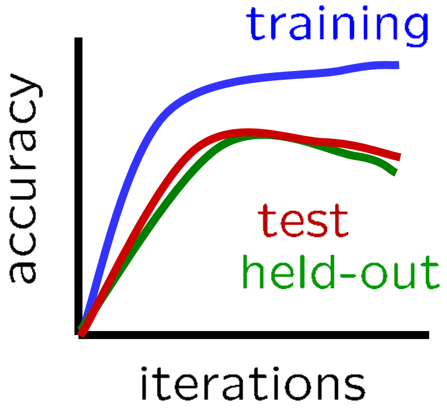
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)



- Mediocre generalization: finds a "barely" separating solution



- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting



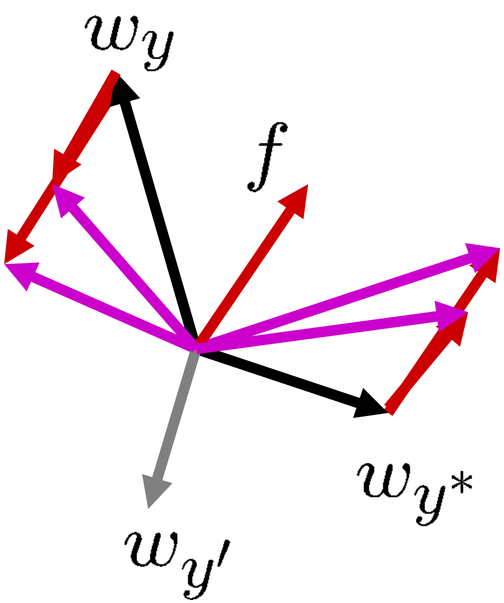
# Fixing the Perceptron

- Idea: adjust 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

$$\min_{w} \frac{1}{2}\sum_{y} ||w_y - w'_y||^2$$

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

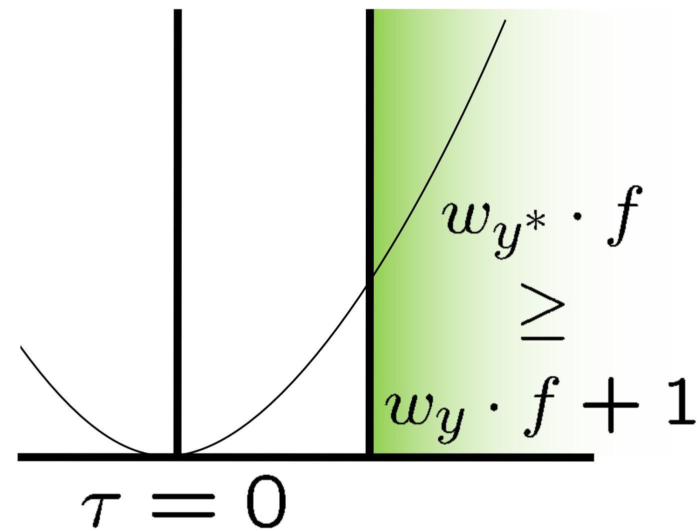
$$\min_{\tau} ||\tau f||^2$$

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

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

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

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

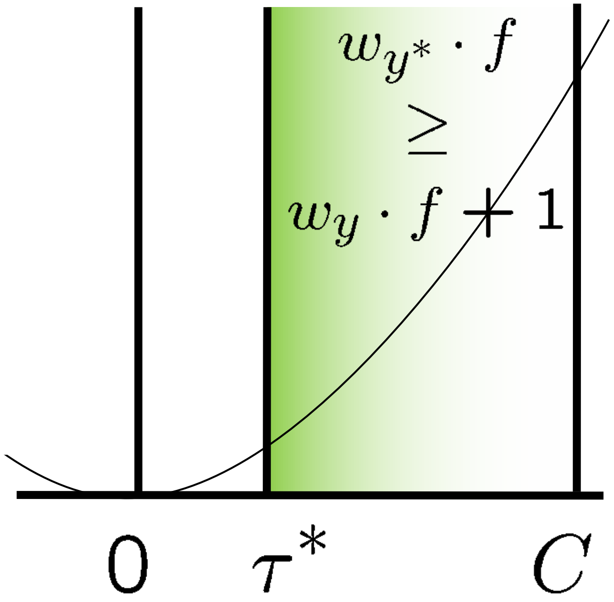



min not τ=0, or would not have made an error, so min will be where equality holds

# Maximum Step Size

- In practice, it's also bad to make updates that are too large
  - Example may be labeled incorrectly
  - You may not have enough features
  - Solution: cap the maximum possible value of $\tau$ with some constant C

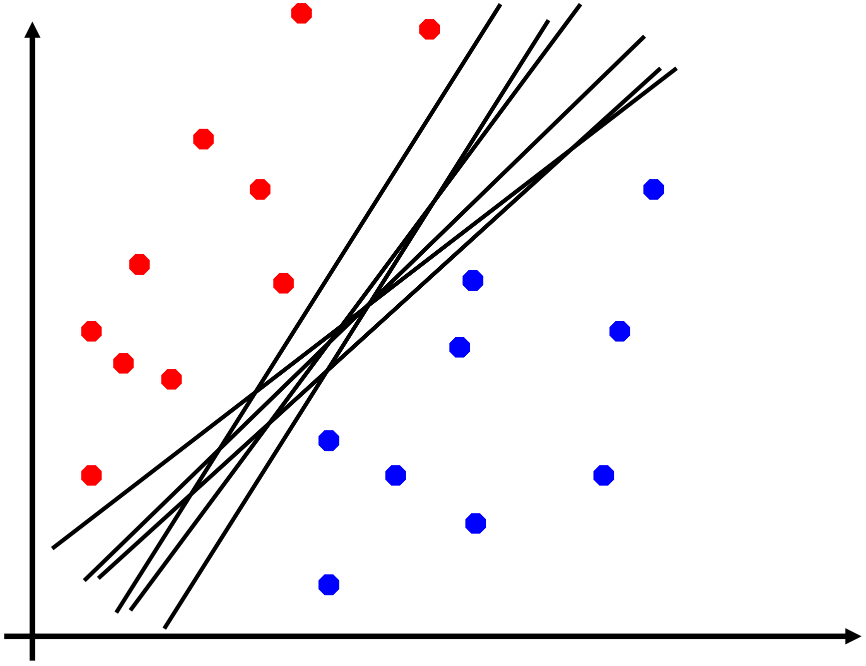
$$\tau^* = \min\left(\frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C\right)$$




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

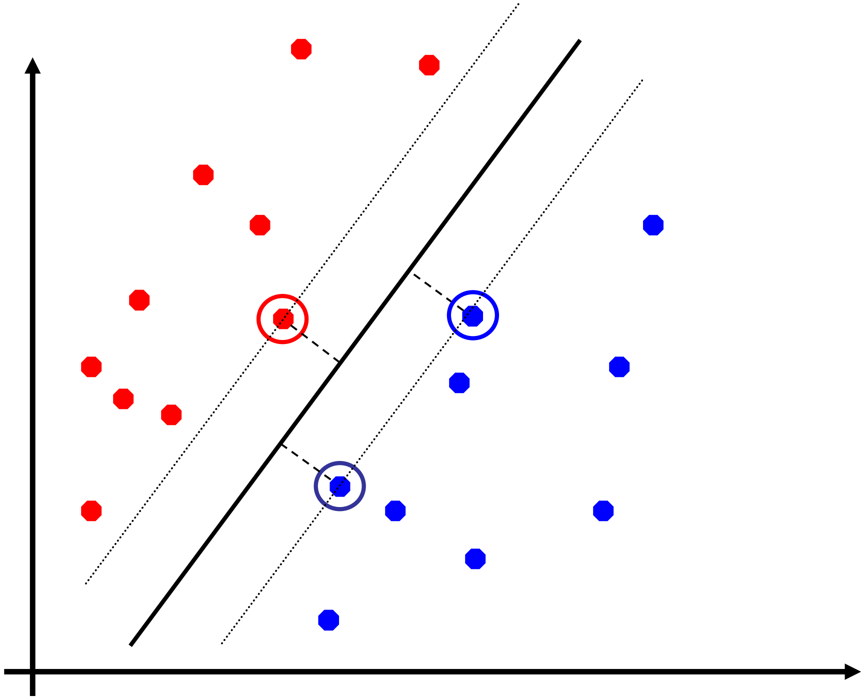
# Linear Separators

- Which of these linear separators is optimal?



# Support Vector Machines

- Maximizing the margin: good according to intuition, theory, practice
- 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

$$\min_w \frac{1}{2}||w||^2$$

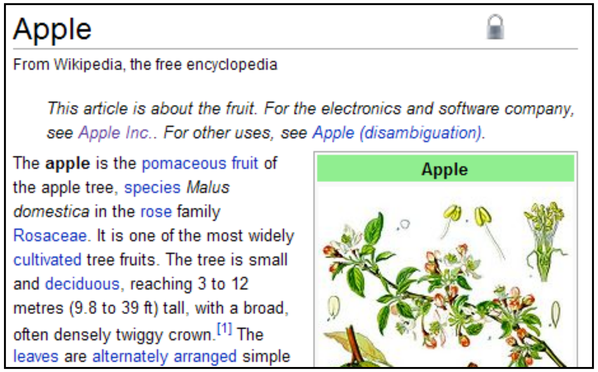
$$\forall i, y \; w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

# Extension: Web Search

$x$ = "Apple Computers"

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

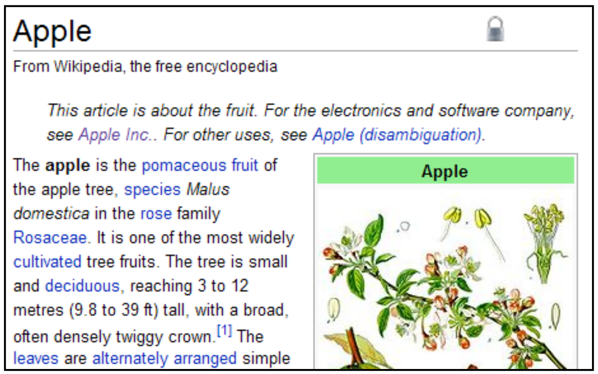

- Web search: not exactly classification, but rather ranking



# Feature-Based Ranking

$x$ = “Apple Computers”

$f(x,$


Apple

From Wikipedia, the free encyclopedia

*This article is about the fruit. For the electronics and software company, see Apple Inc.. For other uses, see Apple (disambiguation).*

The **apple** is the pomaceous fruit of the apple tree, species *Malus domestica* in the rose family Rosaceae. It is one of the most widely cultivated tree fruits. The tree is small and deciduous, reaching 3 to 12 metres (9.8 to 39 ft) tall, with a broad, often densely twiggy crown.[1] The leaves are alternately arranged simple

**Apple**

$) = [0.3\ 5\ 0\ 0\ \ldots]$

---

$f(x,$


Apple Inc.

From Wikipedia, the free encyclopedia

(Redirected from Apple Computer)

**Apple Inc.**,

**Apple Inc.**

$) = [0.8\ 4\ 2\ 1\ \ldots]$

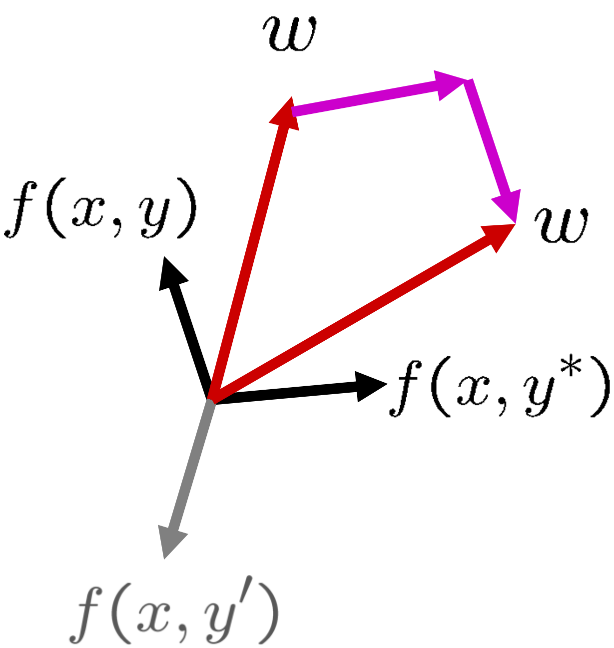
# Perceptron for Ranking

- 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):

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



# Classification: Comparison

- Naïve Bayes
  - Builds a model training data
  - Gives prediction probabilities
  - Strong assumptions about feature independence
  - One pass through data (counting)

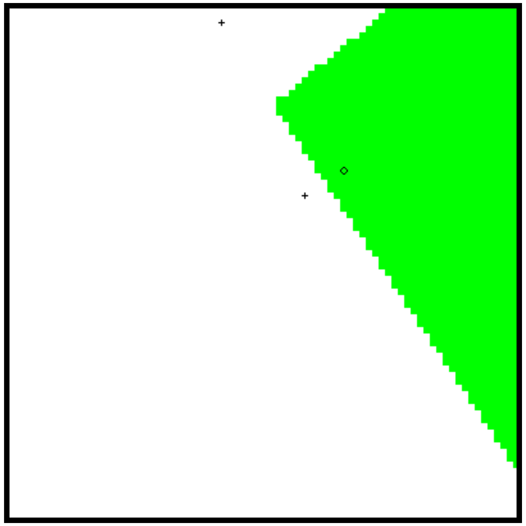
- Perceptrons / MIRA:
  - Makes less assumptions about data
  - Mistake-driven learning
  - Multiple passes through data (prediction)
  - Often more accurate

# Today

- Perceptron wrap-up
- **Kernels and clustering**

# Case-Based Reasoning: KNN

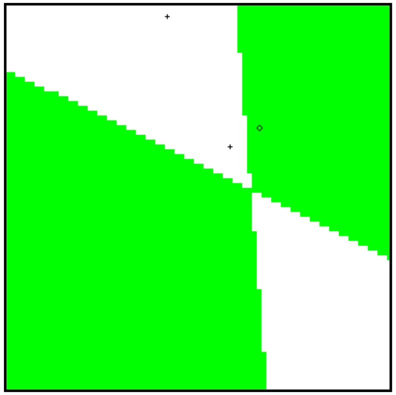
- Similarity for classification
  - Case-based reasoning
  - Predict an instance's label using similar instances
- Nearest-neighbor classification
  - 1-NN: copy the label of the most similar data point
  - K-NN: let the k nearest neighbors vote (have to devise a weighting scheme)
  - Key issue: how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



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

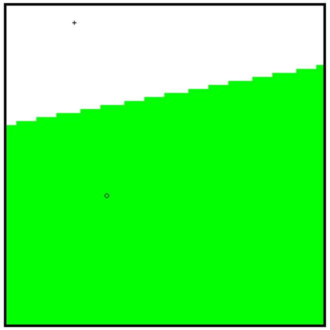
# Parametric / Non-parametric

- 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

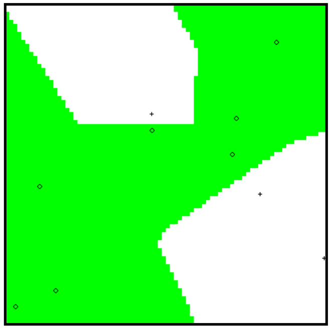


Truth

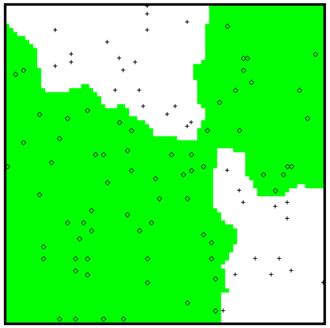
2 Examples



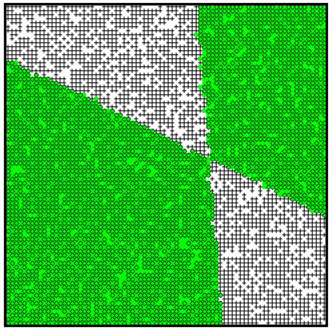
10 Examples



100 Examples

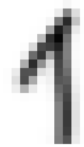


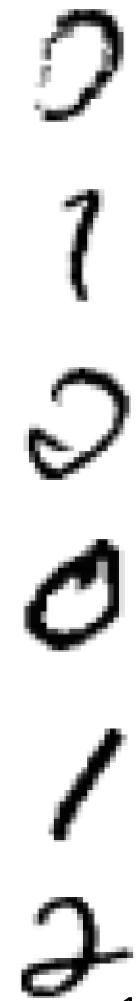
10000 Examples



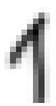
# Nearest-Neighbor Classification

- 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

# Basic Similarity

- Many similarities based on feature dot products:

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

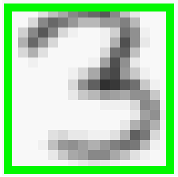
- If features are just the pixels:

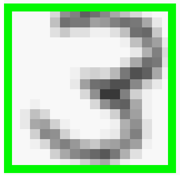
$$\text{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i$$

- 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:



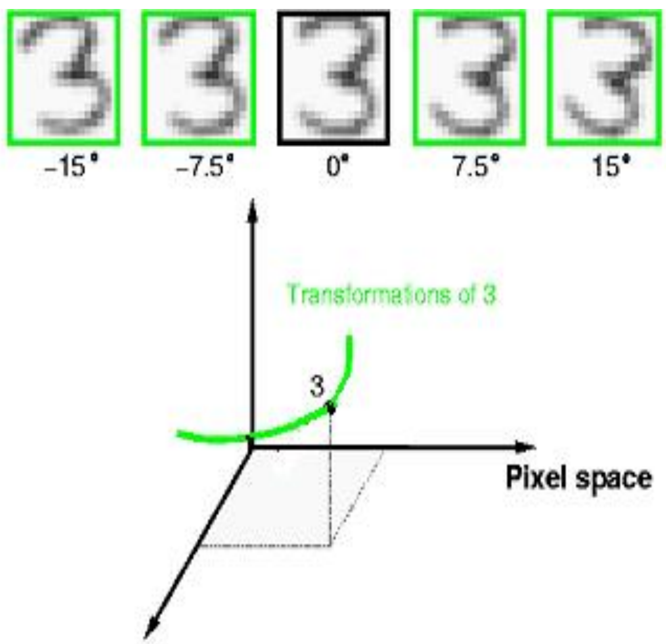


    - 16 x 16 = 256 pixels; a point in 256-dim space
    - Small similarity in $R^{256}$ (why?)
  - How to incorporate invariance into similarities?



This and next few slides adapted from Xiao Hu, UIUC

# Rotation Invariant Metrics



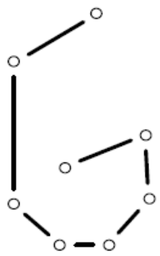

- Each example is now a curve in $R^{256}$
- Rotation invariant similarity:

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

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

# 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]

# Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA

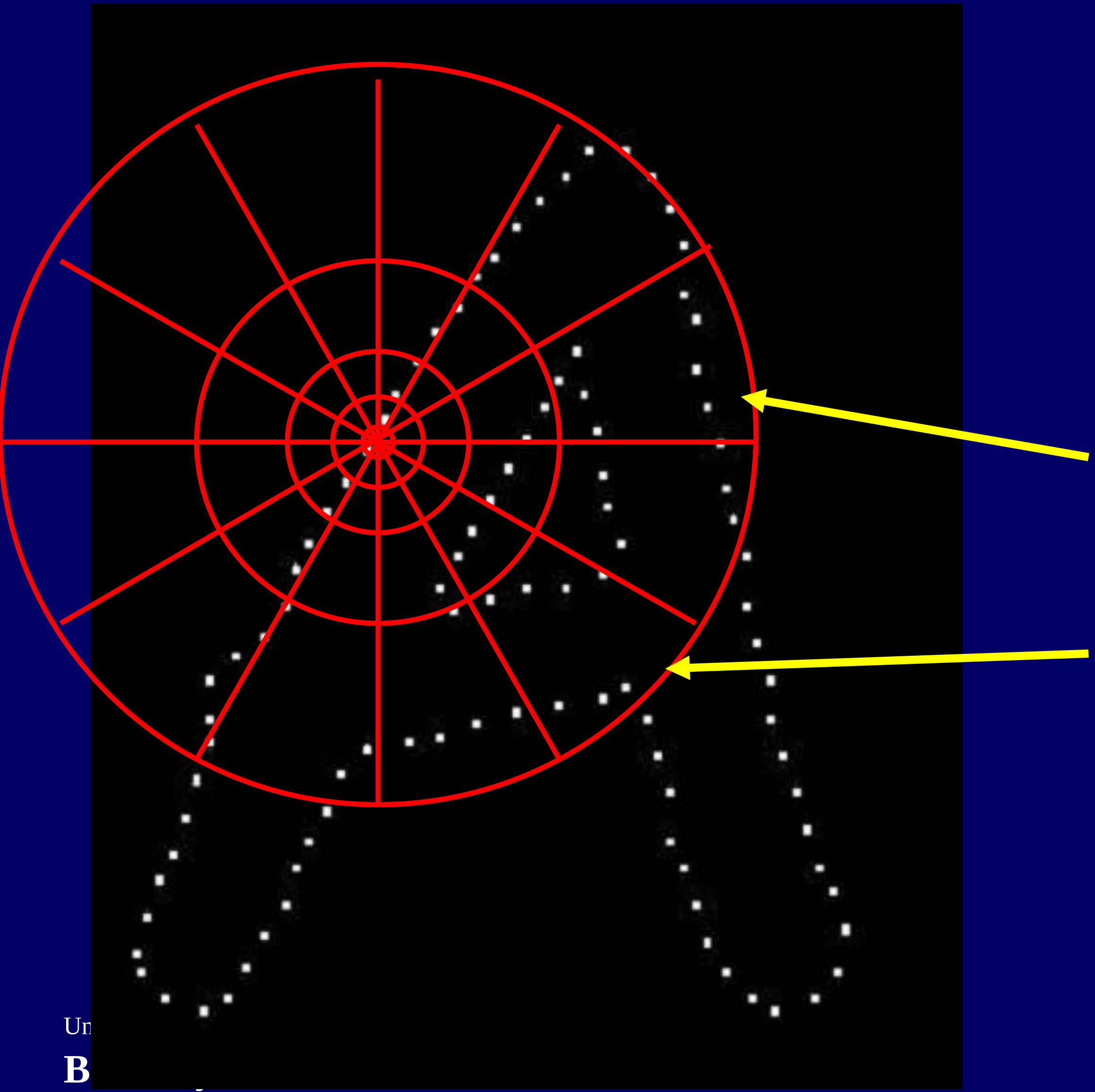
Greg Mori and Jitendra Malik
CVPR 2003

# EZ-Gimpy

- Word-based CAPTCHA
  - Task is to read a single word obscured in clutter
- Currently in use at Yahoo! and Ticketmaster
  - Filters out ‘bots’ from obtaining free email accounts, buying blocks of tickets



# Shape contexts (Belongie et al. 2001)



Count the number of points inside each bin, e.g.:

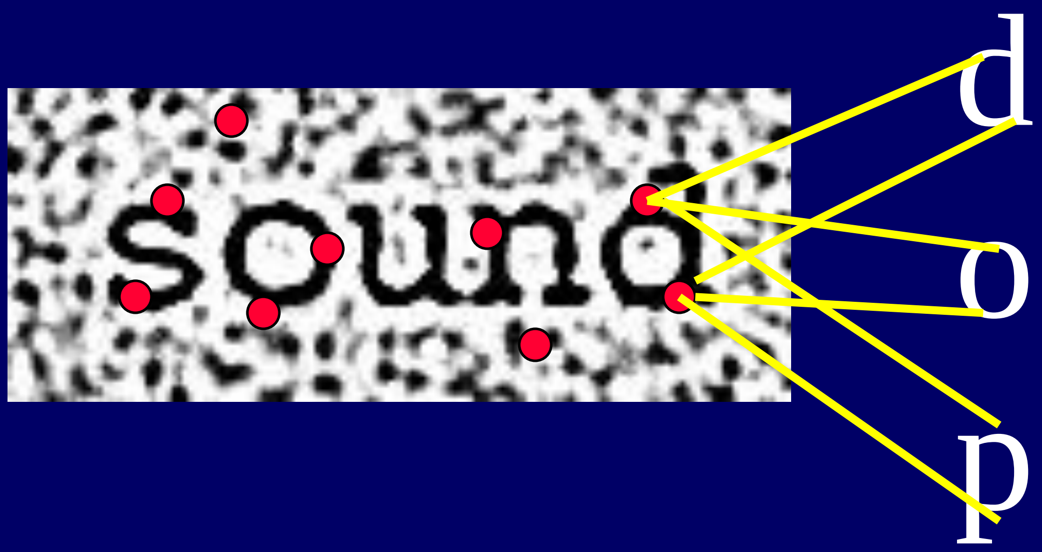
Count = 8

…

Count = 7

☞ Compact representation of distribution of points relative to each point

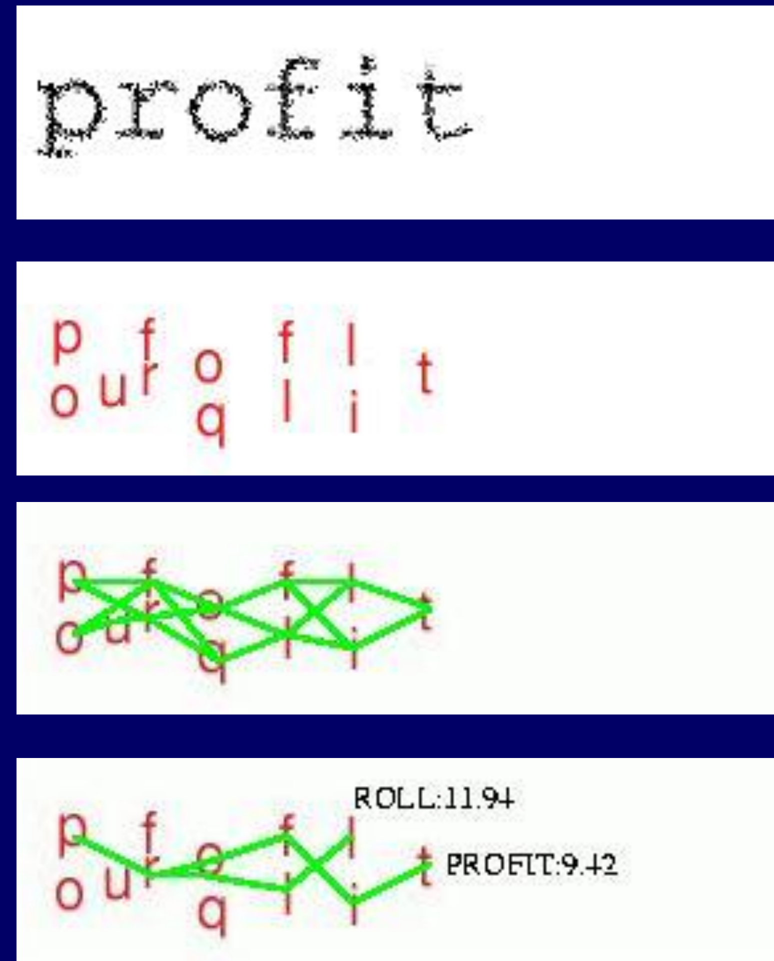
# Fast Pruning: Representative Shape Contexts




- Pick k points in the image at random
  - Compare to all shape contexts for all known letters
  - Vote for closely matching letters
- Keep all letters with scores under threshold

# Algorithm A

- Look for letters
  - Representative Shape Contexts
- Find pairs of letters that are “consistent”
  - Letters nearby in space
- Search for valid words
- Give scores to the words



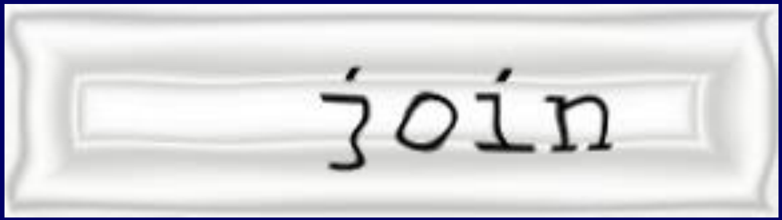
# EZ-Gimpy Results with Algorithm A

- 158 of 191 images correctly identified: 83%
  - Running time: ~10 sec. per image (MATLAB, 1 Ghz P3)


horse


spade


smile




join

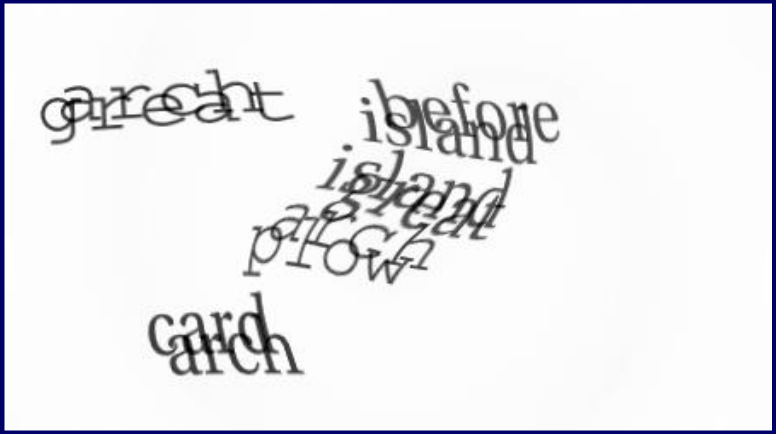

canvas


here

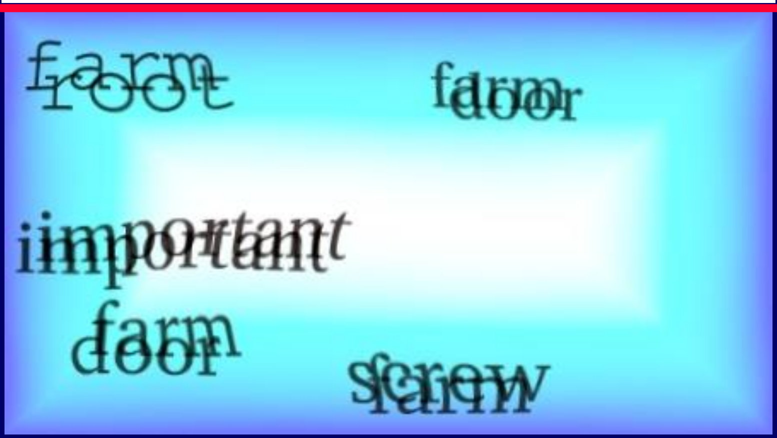
# Results with Algorithm B

dry clear medical

| # Correct words | % tests (of 24) |
|---|---|
| 1 or more | 92% |
| 2 or more | 75% |
| 3 | 33% |
| EZ-Gimpy | 92% |



card arch plate



door farm important

# 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.

$$w_y = w_y - f(x)$$

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

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

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

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

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

# Dual Perceptron

- How to classify a new example x?

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

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

# 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)

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

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

# 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

# Kernels: Who Cares?

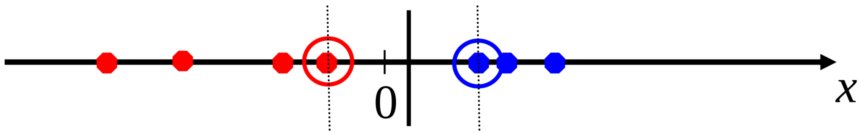
- So far: a very strange way of doing a very simple calculation
- "Kernel trick": we can substitute any* similarity function in place of the dot product
- Lets us learn new kinds of hypothesis

$$K(x_i,x_j) = f(x_i)^T f(x_j)$$

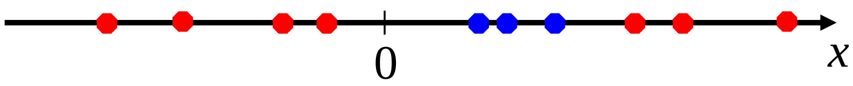
* Fine print: if your kernel doesn't satisfy certain technical requirements, lots of proofs break. E.g. convergence, mistake bounds. In practice, illegal kernels *sometimes* work (but not always).

# Non-Linear Separators

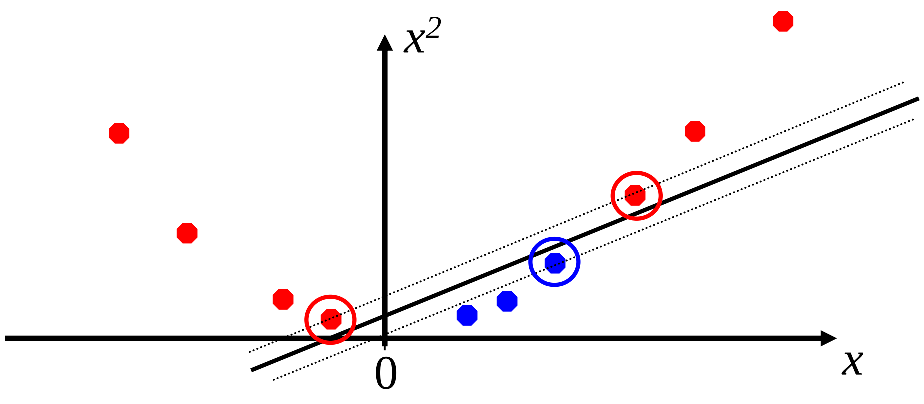
- Data that is linearly separable (with some noise) works out great:




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

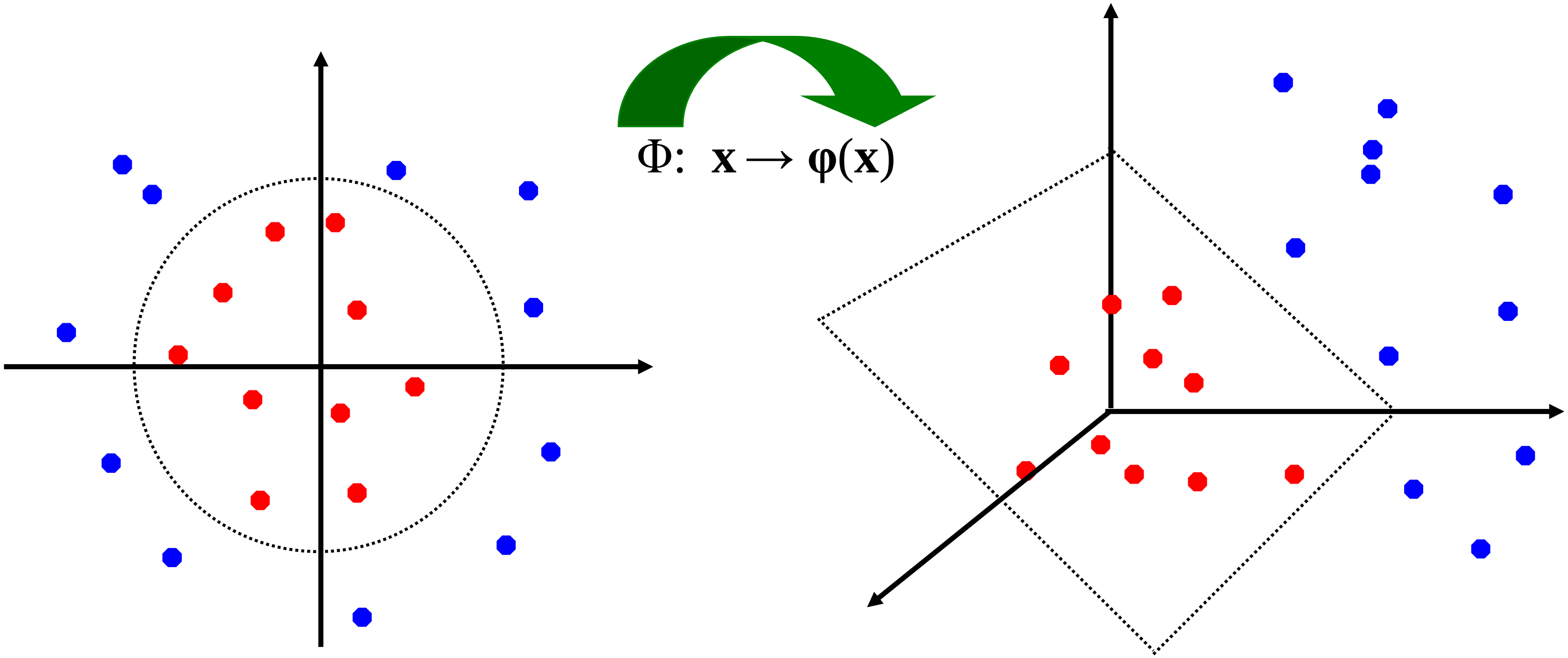



- How about… mapping data to a higher-dimensional space:



# Non-Linear Separators

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



# Example

2-dimensional vectors $\mathrm{x}=[x_1 \;\; x_2]$;

let $K(\mathrm{x_i},\mathrm{x_j})=(1 + \mathrm{x_i}^{\mathrm{T}}\mathrm{x_j})^2$

Need to show that $K(\mathrm{x_i},\mathrm{x_j})= \varphi(\mathrm{x_i})^{\mathrm{T}}\varphi(\mathrm{x_j})$:

$$K(\mathrm{x_i},\mathrm{x_j})=(1 + \mathrm{x_i}^{\mathrm{T}}\mathrm{x_j})^2,$$

$$= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}$$

$$= [1 \;\; x_{i1}^2 \;\; \sqrt{2}\, x_{i1}x_{i2} \;\; x_{i2}^2 \;\; \sqrt{2}x_{i1} \;\; \sqrt{2}x_{i2}]^{\mathrm{T}}$$

$$[1 \;\; x_{j1}^2 \;\; \sqrt{2}\, x_{j1}x_{j2} \;\; x_{j2}^2 \;\; \sqrt{2}x_{j1} \;\; \sqrt{2}x_{j2}]$$

$$= \varphi(\mathrm{x_i})^{\mathrm{T}}\varphi(\mathrm{x_j}),$$

where $\varphi(\mathrm{x}) = [1 \;\; x_1^2 \;\; \sqrt{2}\, x_1x_2 \;\; x_2^2 \;\; \sqrt{2}x_1 \;\; \sqrt{2}x_2]$

from Andrew Moore's tutorial: http://www.autonlab.org/tutorials/svm.html

# Examples of kernel functions

- Linear: $$K(x_i, x_j) = x_i^T x_j$$

- Gaussian RBF: $$K(x_i,x_j) = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{2\sigma^2}\right)$$

- Histogram intersection:

$$K(x_i, x_j) = \sum_k \min(x_i(k), x_j(k))$$

# 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 kernels
- 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

# Recap: Classification

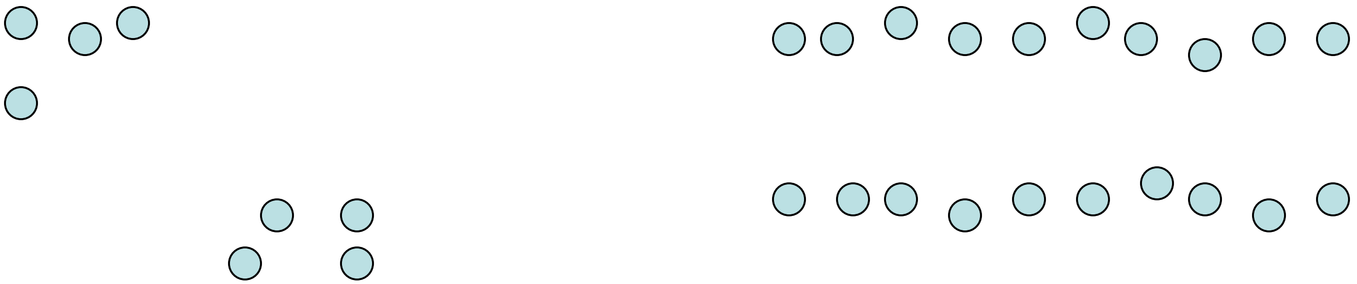
- Classification systems:
  - Supervised learning
  - Make a prediction given evidence
  - We've seen several methods for this
  - Useful when you have labeled data

# 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

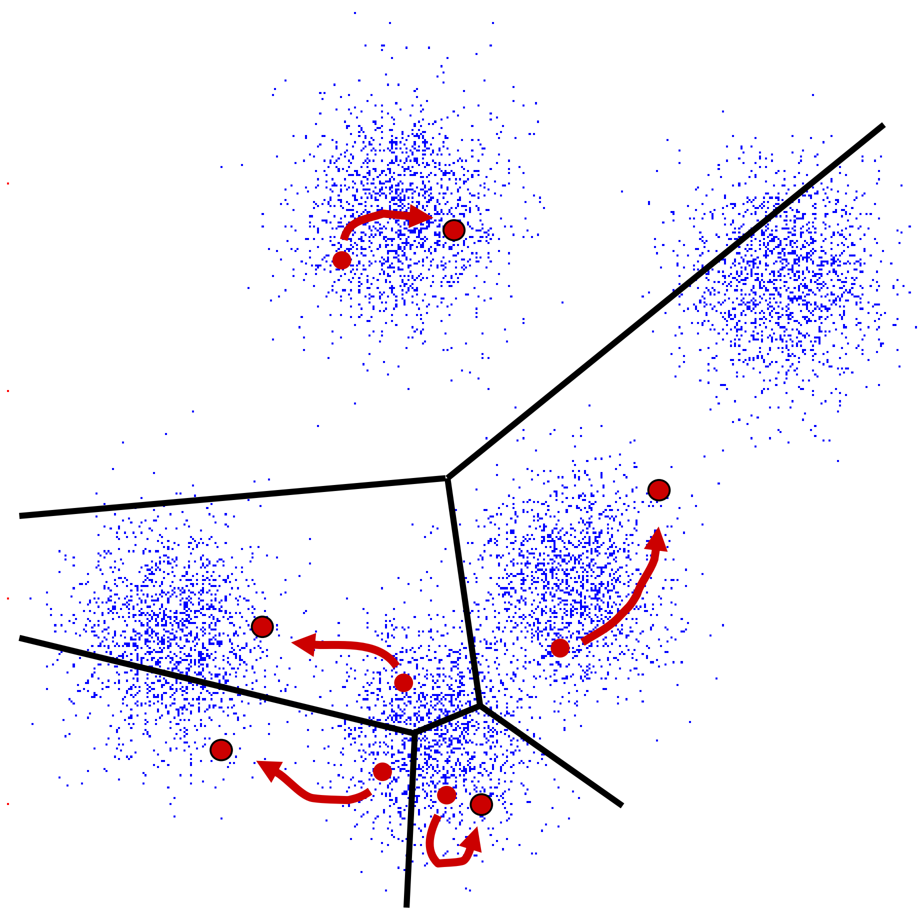


- What could "similar" mean?
  - One option: small (squared) Euclidean distance

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

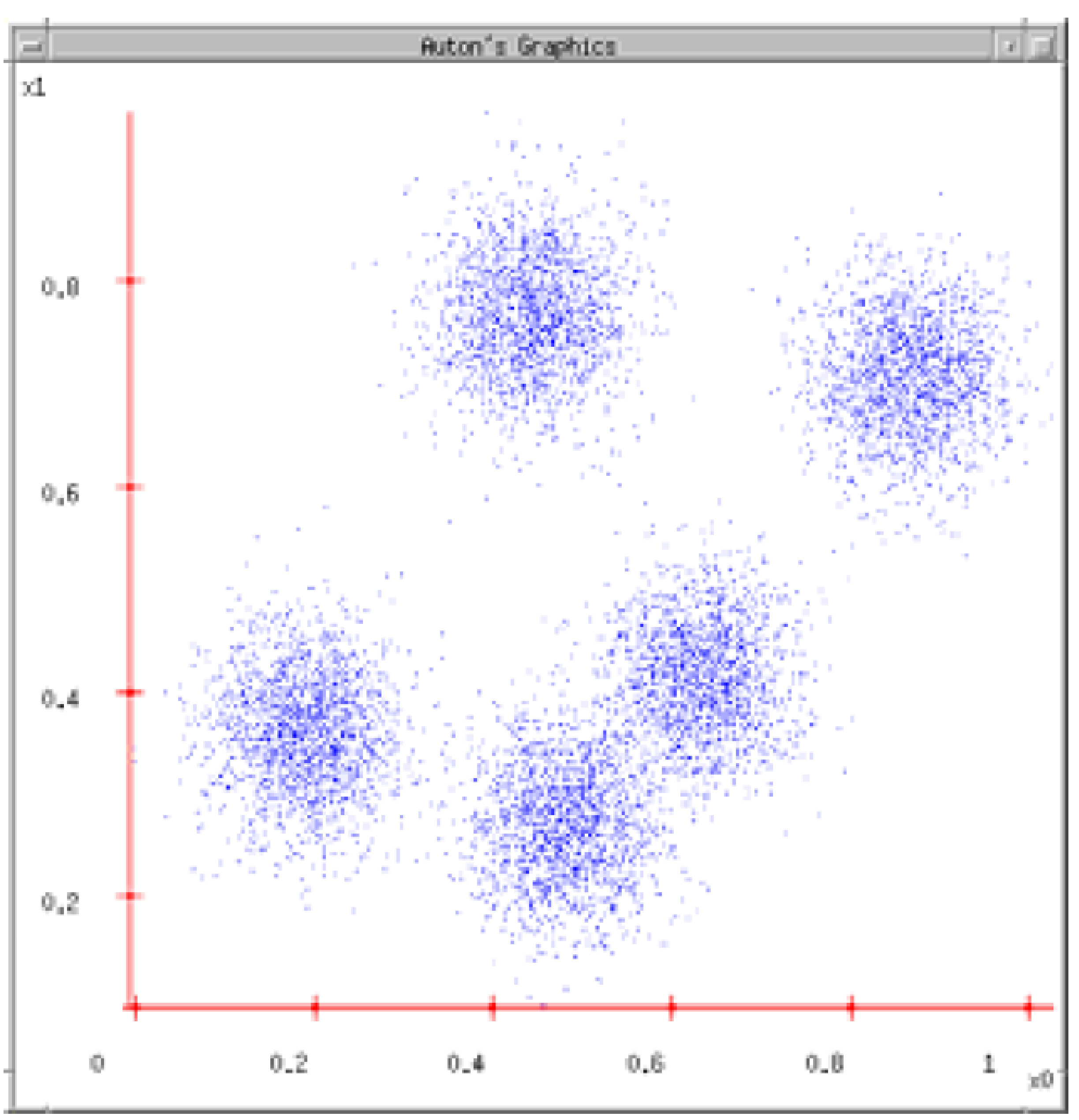
# K-Means

- 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



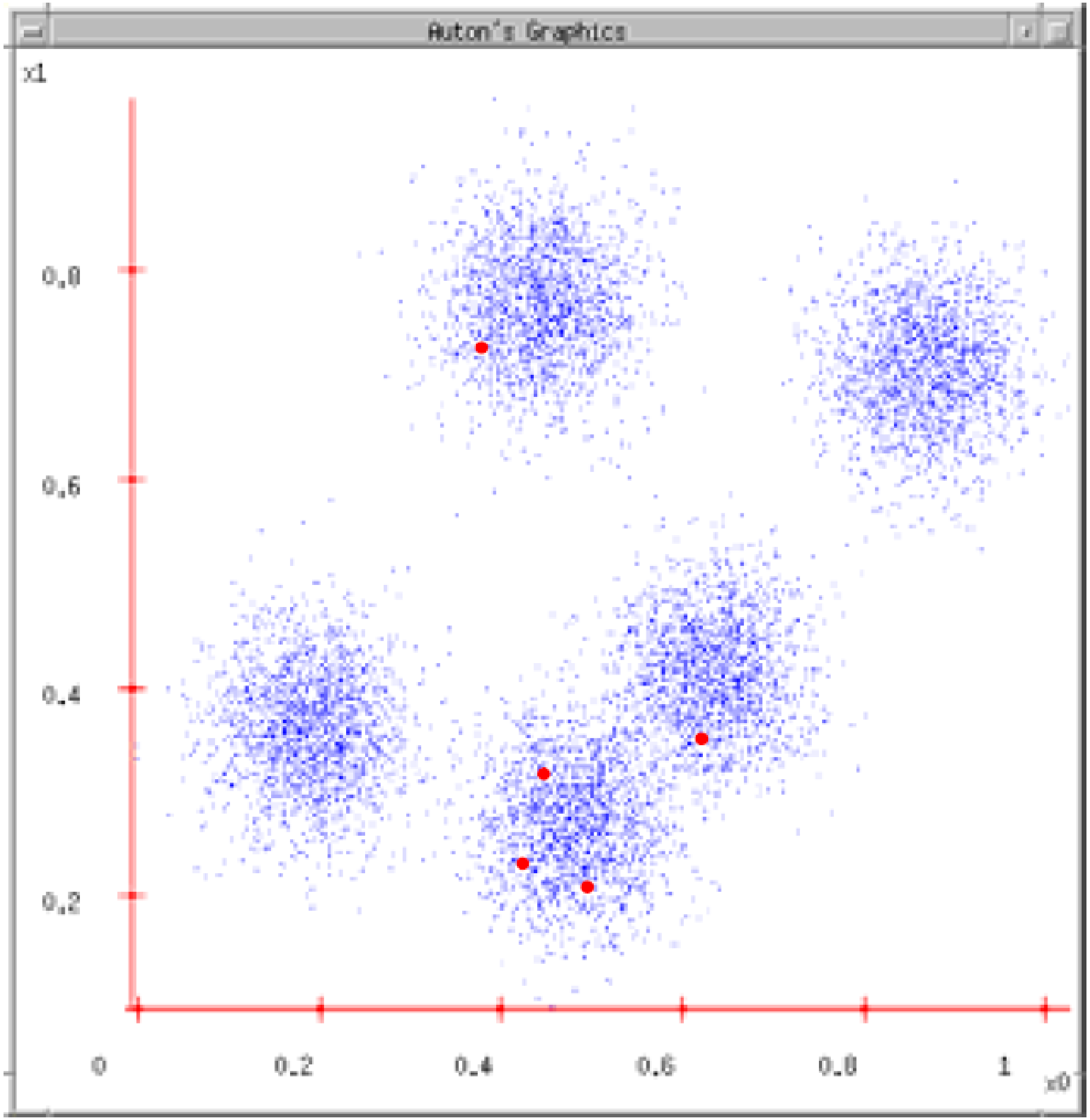
# K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*



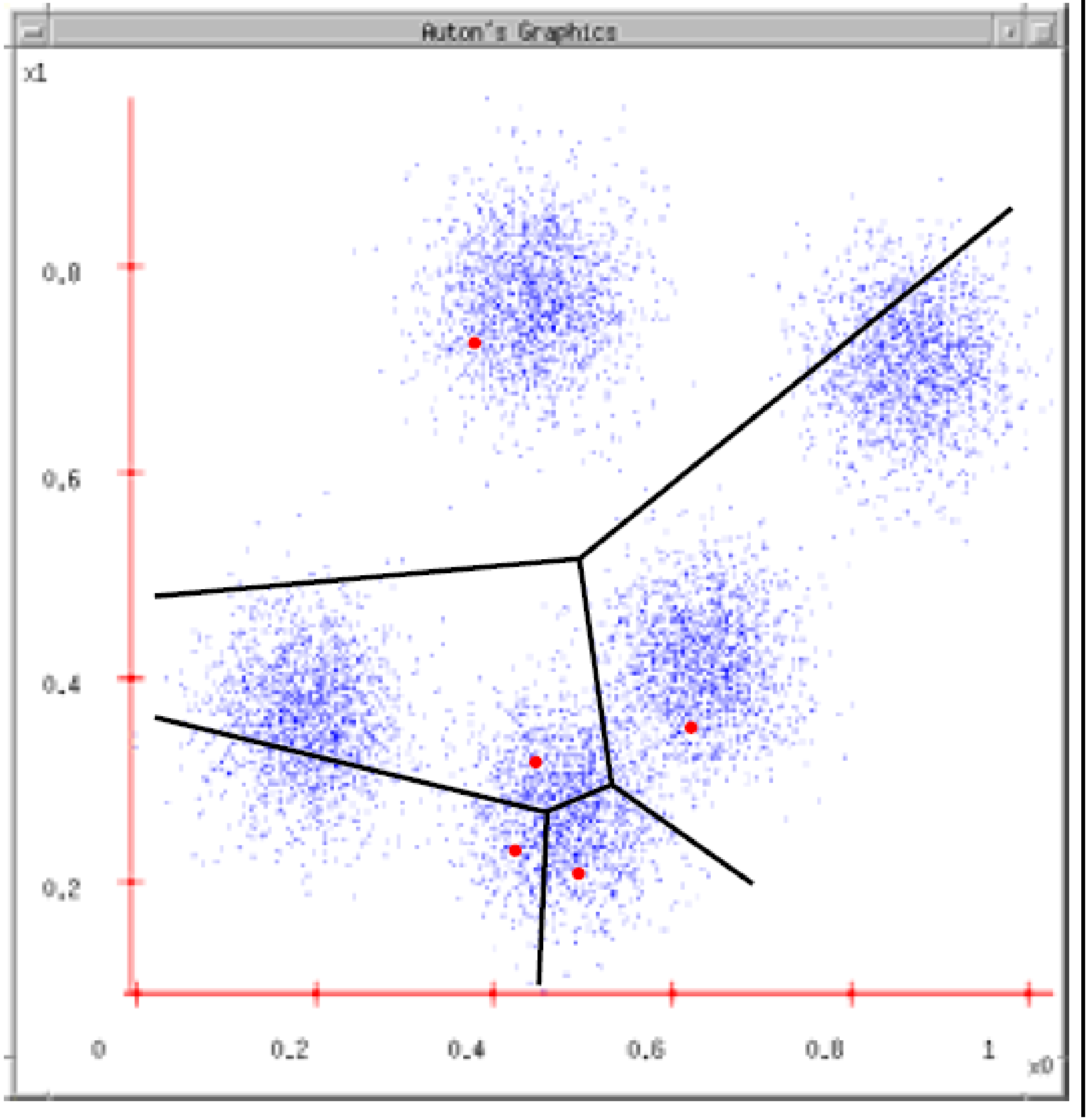
# K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



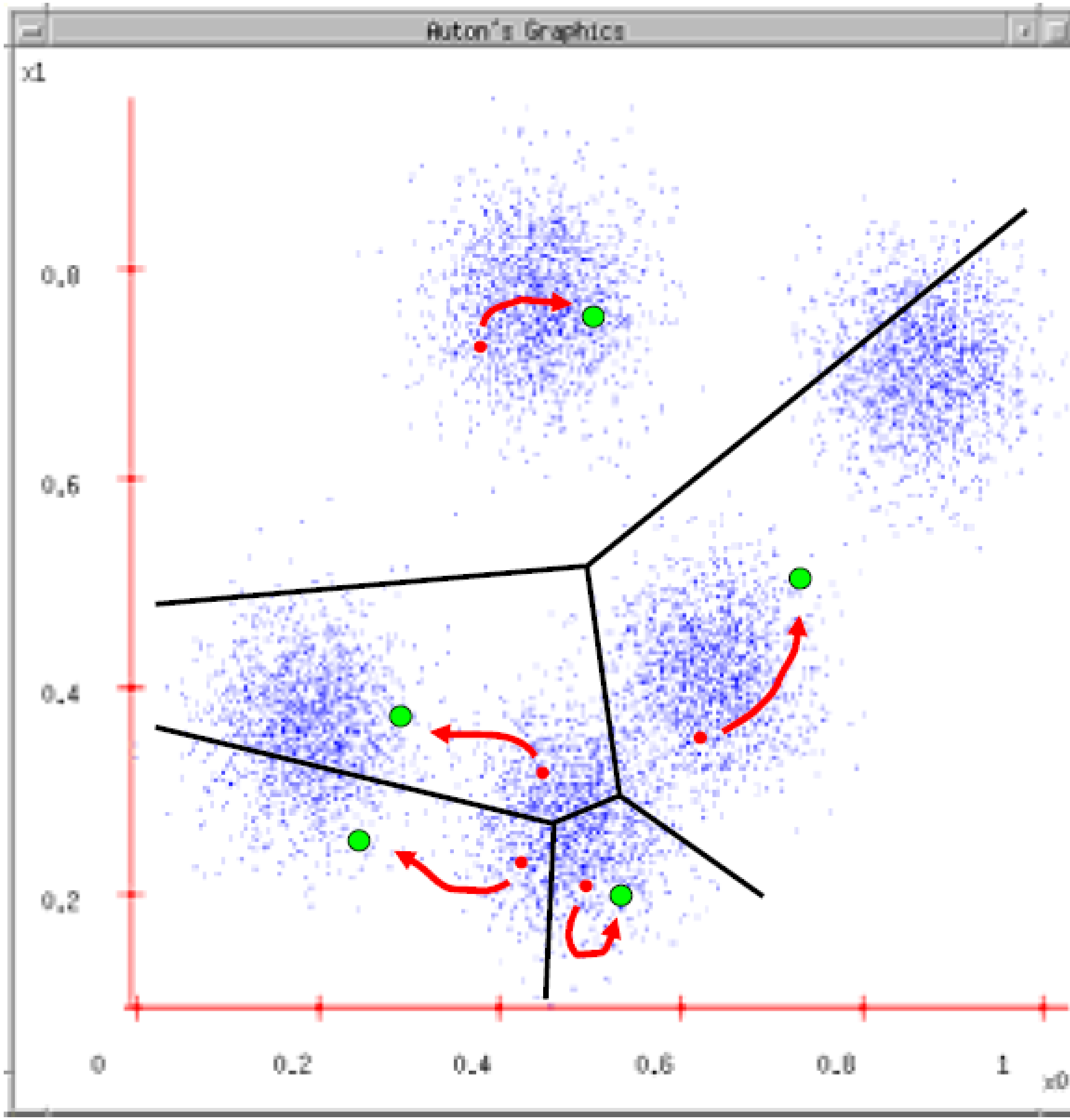
# K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



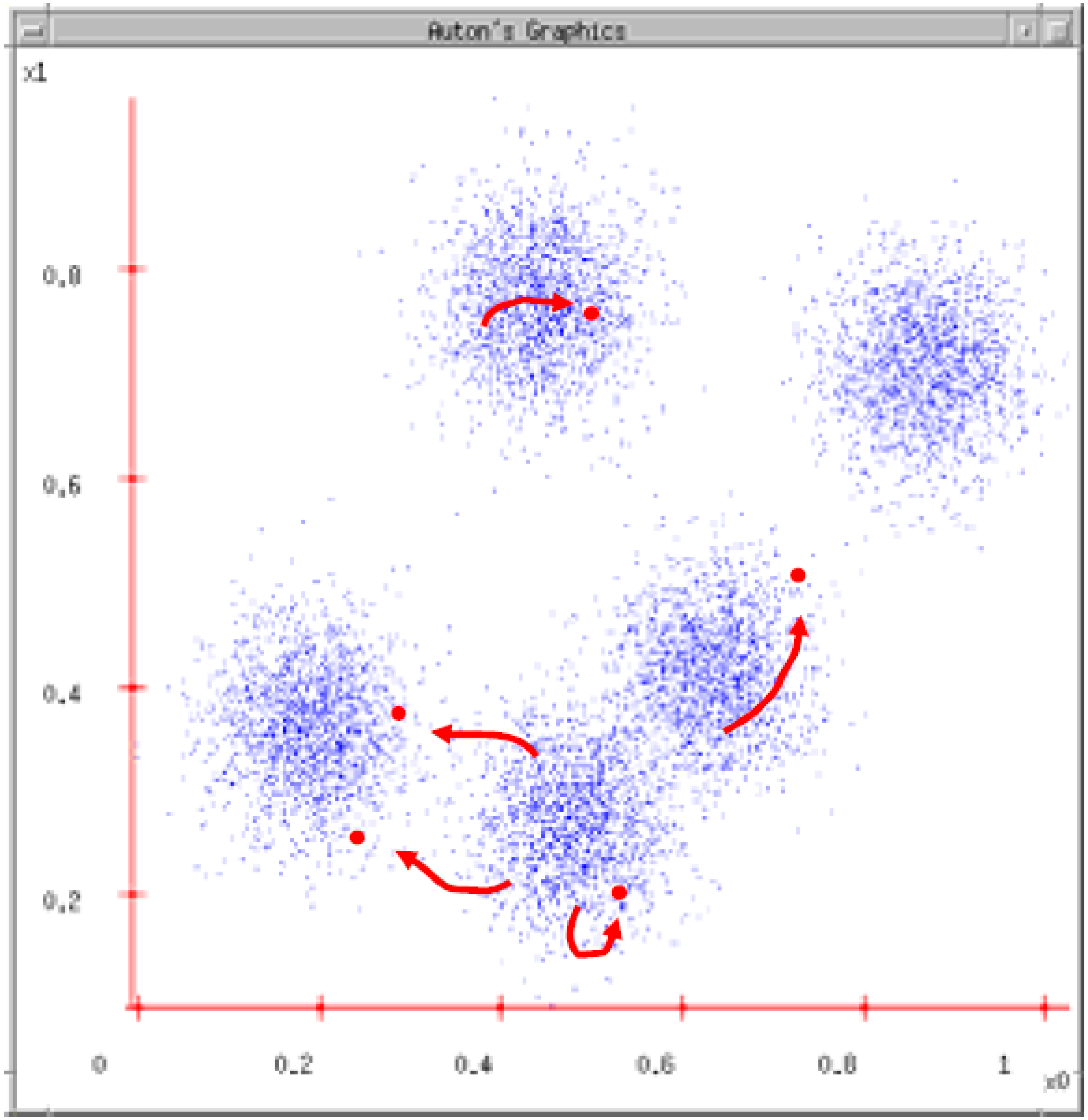
# K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



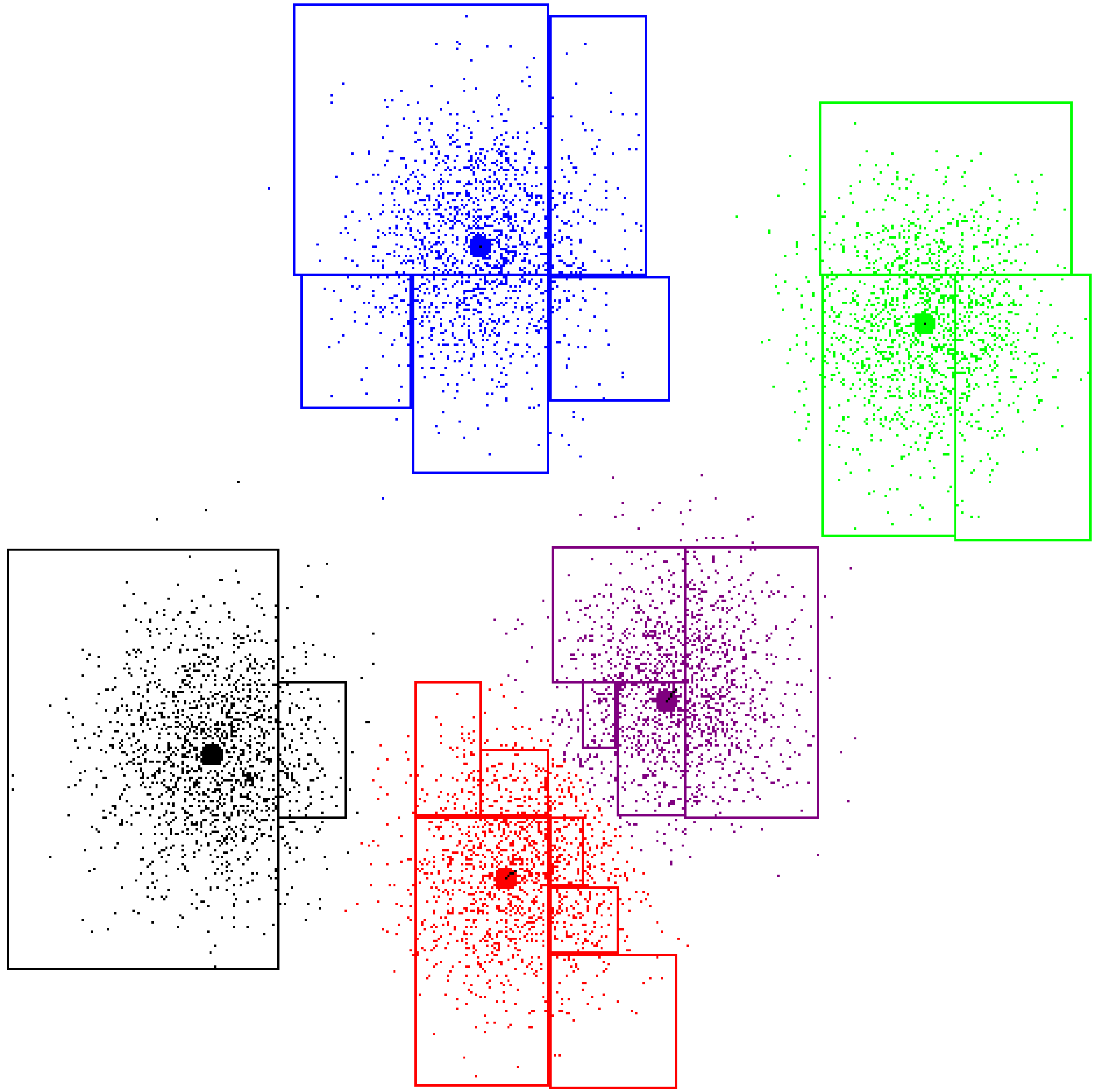
# K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!




Andrew Moore

# K-Means Example



# Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **intensity** similarity

Feature space: intensity value (1-d)

Slide credit: Kristen Grauman

K=2

K=3

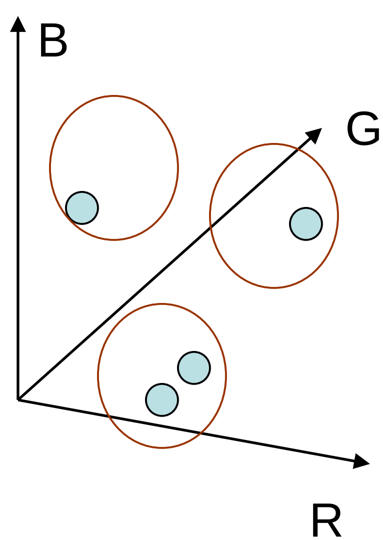
*quantization* of the feature space;
segmentation label map

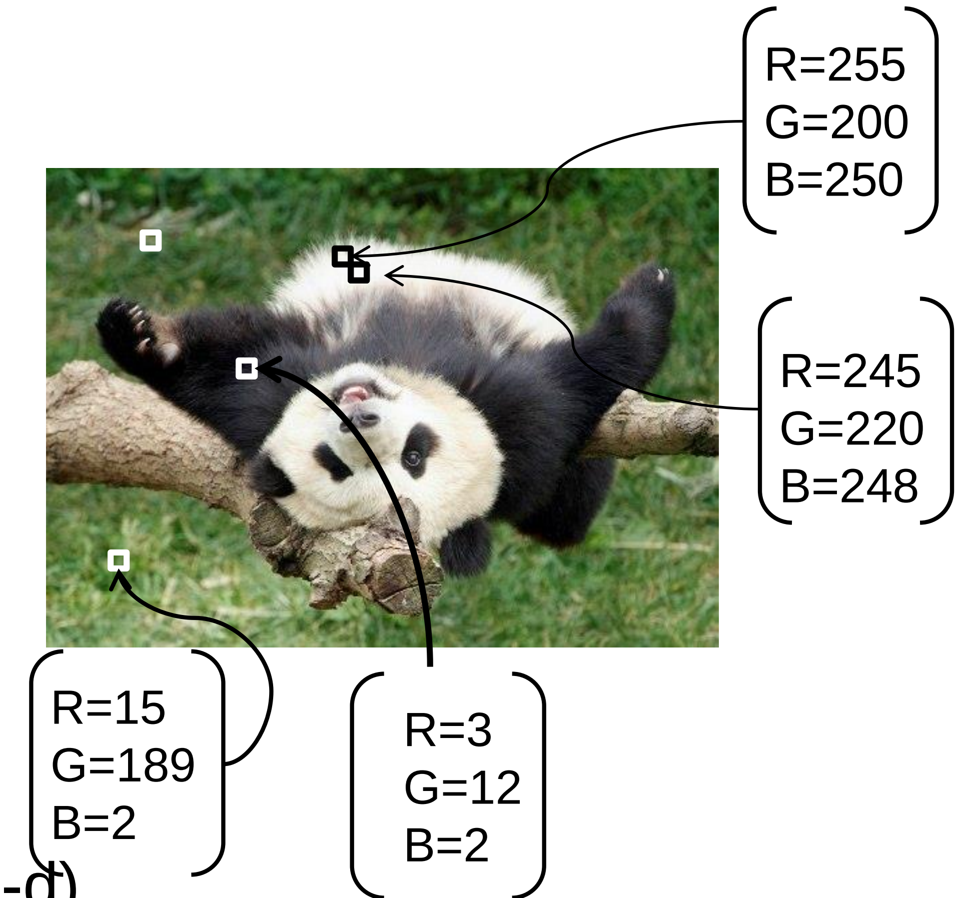
Slide credit: Kristen Grauman

# Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **color** similarity




Feature space: color value (3-d)

Slide credit: Kristen Grauman

# K-Means as Optimization

- Consider the total distance to the means:

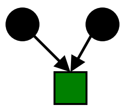
$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

points

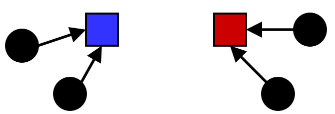
means

assignments

- Each iteration reduces phi



- Two stages each iteration:
  - Update assignments: fix means c,
    change assignments a
  - Update means: fix assignments a,
    change means c



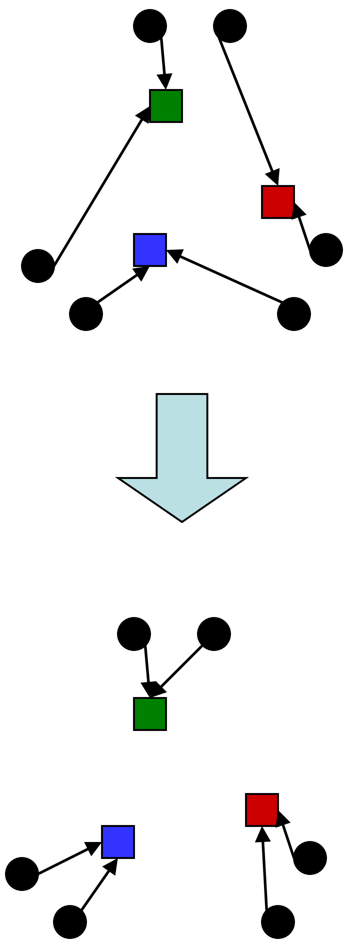
# Phase I: Update Assignments

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

$$a_i = \operatorname*{argmin}_k \mathrm{dist}(x_i, c_k)$$

- Can only decrease total distance phi!

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



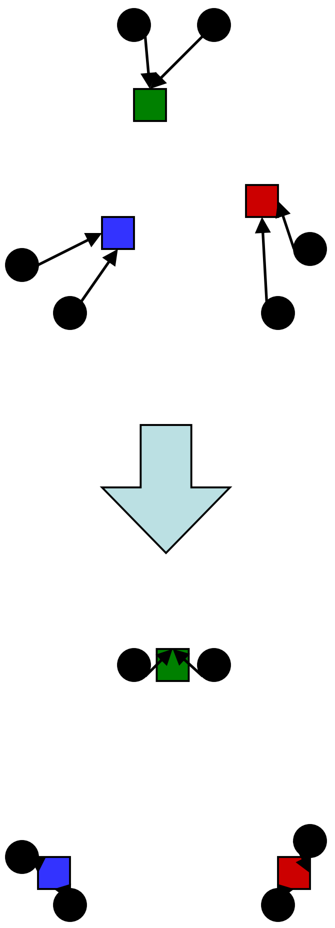
# Phase II: Update Means

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

$$c_k = \frac{1}{|\{i : a_i = k\}|} \sum_{i:a_i=k} x_i$$

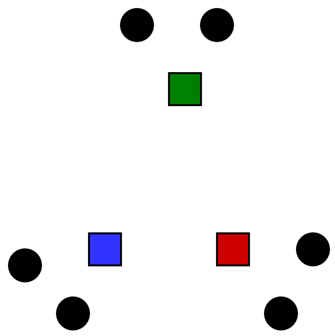
- Also can only decrease total distance… (Why?)

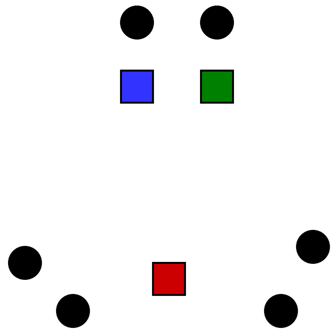
- Fun fact: the point y with minimum squared Euclidean distance to a set of points {x} is their mean



# Initialization

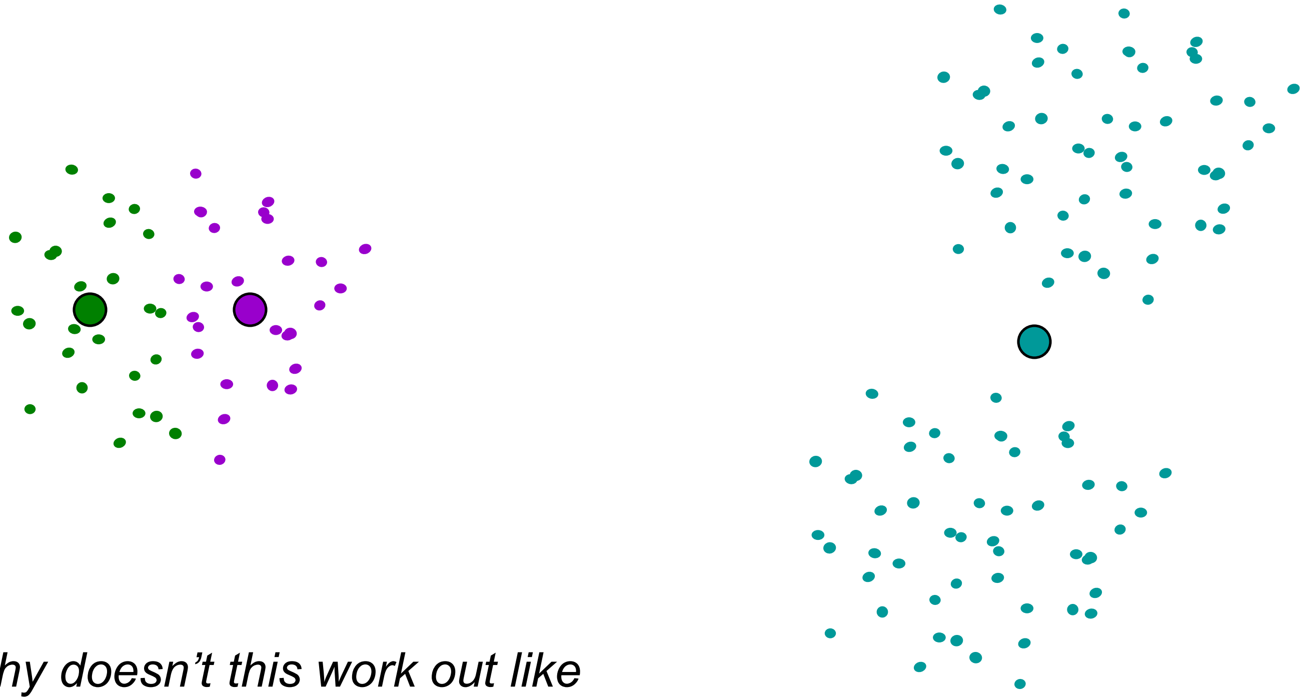
- K-means is non-deterministic
  - Requires initial means
  - It does matter what you pick!
  - What can go wrong?
  - Various schemes for preventing this kind of thing: variance-based split / merge, initialization heuristics





# 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?
- How many clusters to pick?
- Do people ever use it?

# Example: K-means for feature quantization

Image 1

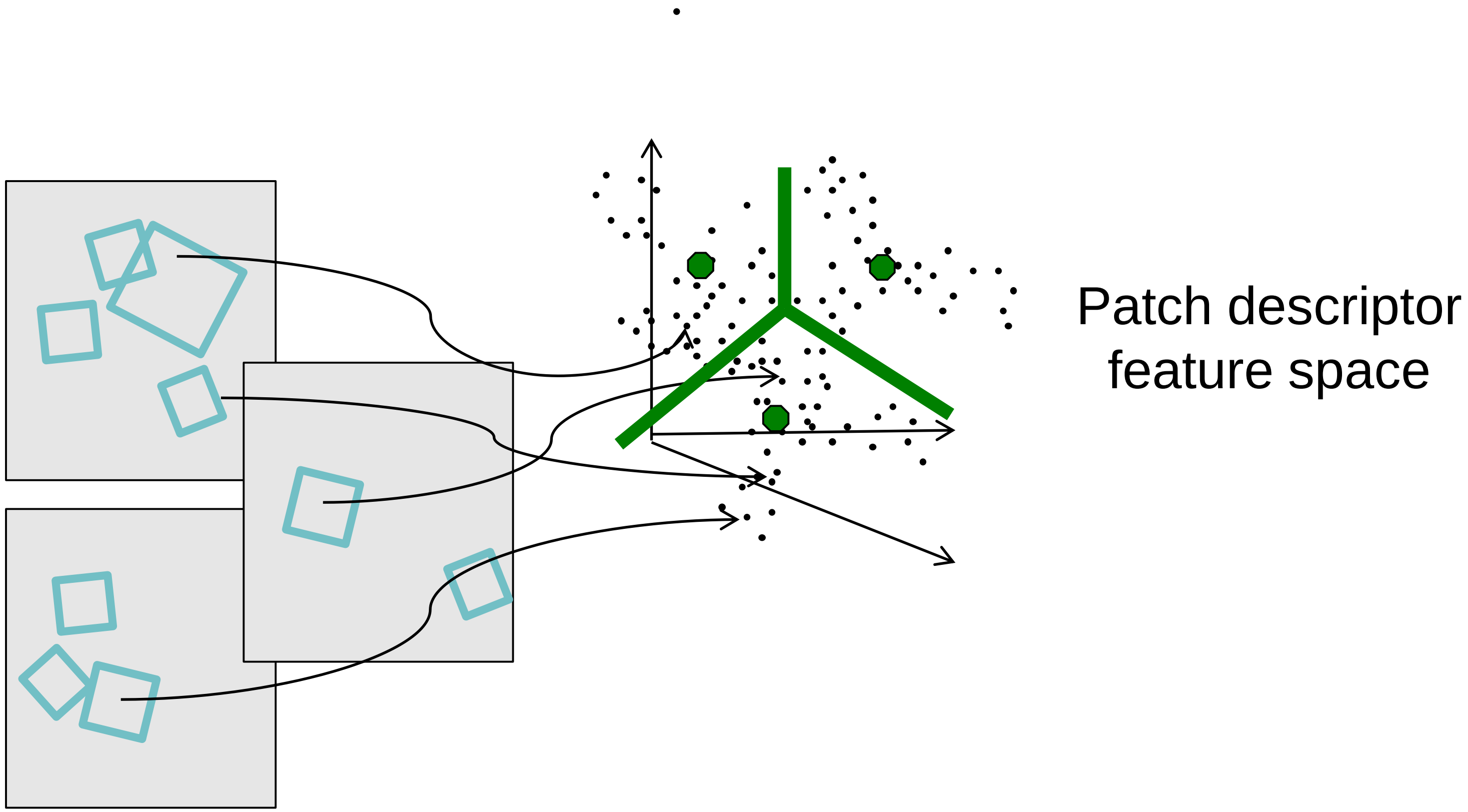
Image 2

Detecting local features

Slide credit: Kristen Grauman

# Example: K-means for feature quantization

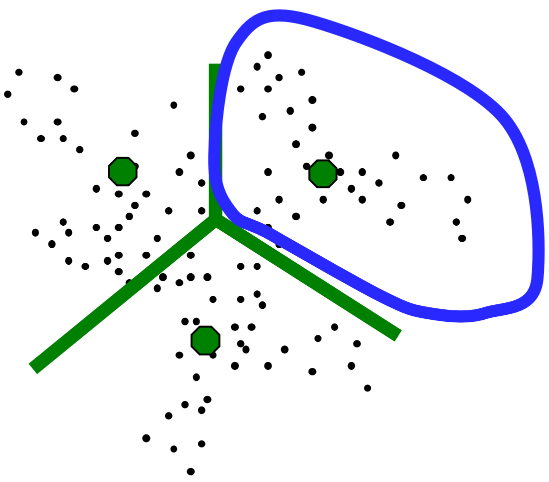
- Map high-dimensional descriptors to "visual words" by quantizing the feature space




Slide credit: Kristen Grauman

# Example: K-means for feature quantization

- Example visual words: each group of patches belongs to the same visual word



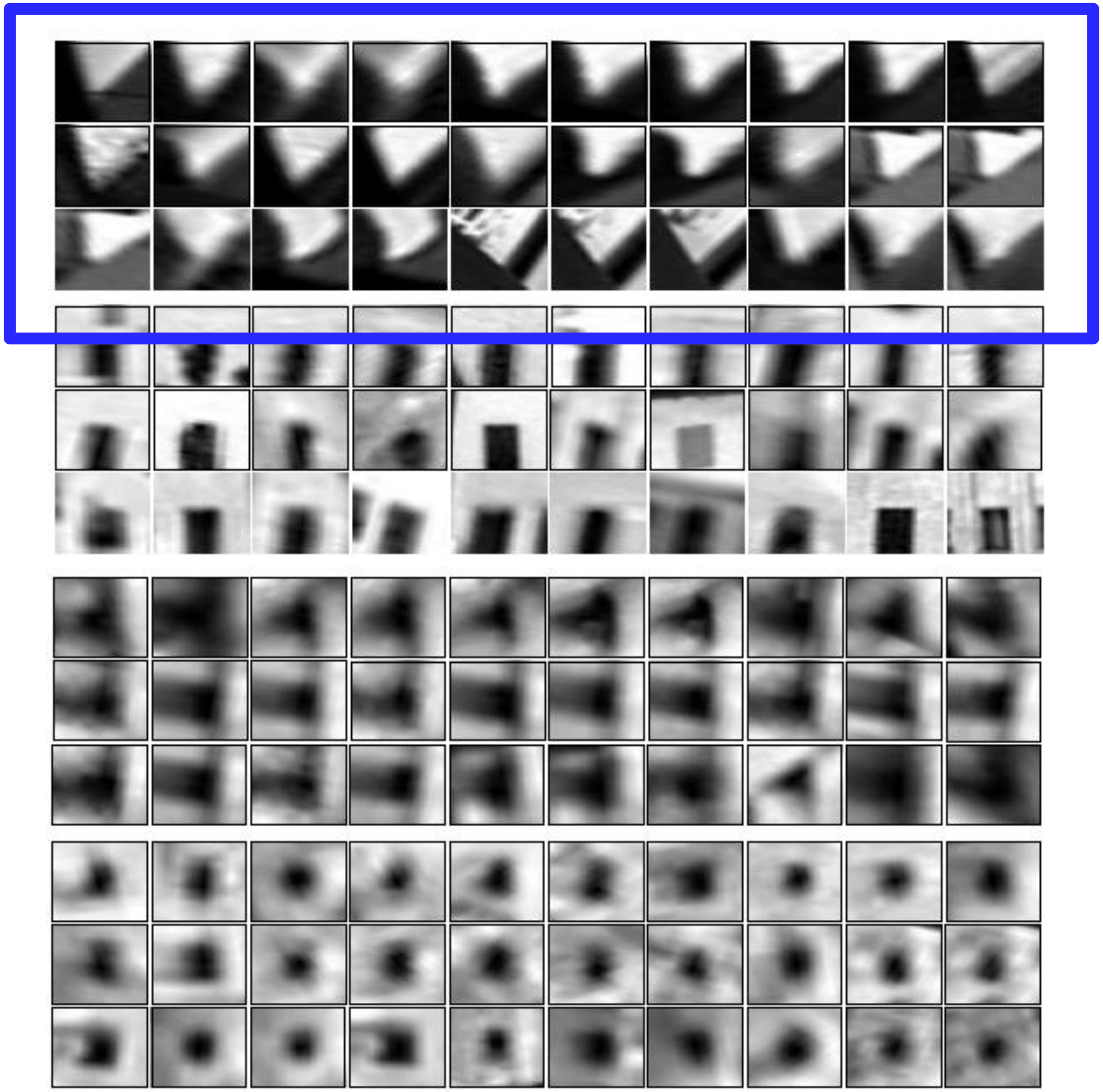
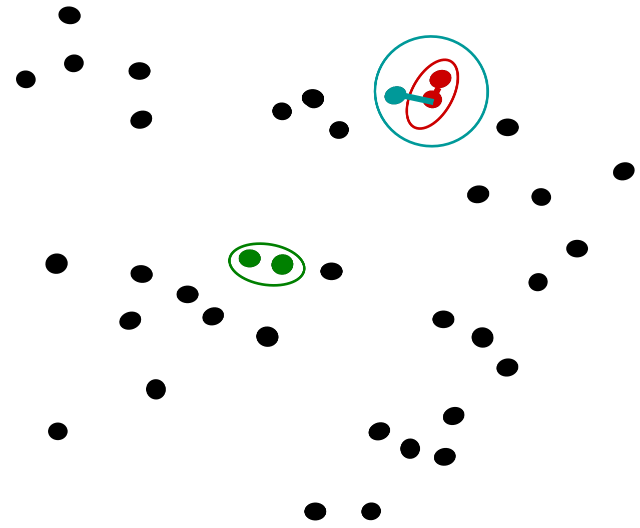


Figure from Sivic & Zisserman, ICCV 2003

Slide credit: Kristen Grauman

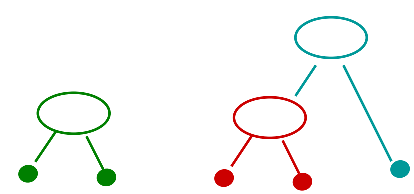
# Agglomerative Clustering

- 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

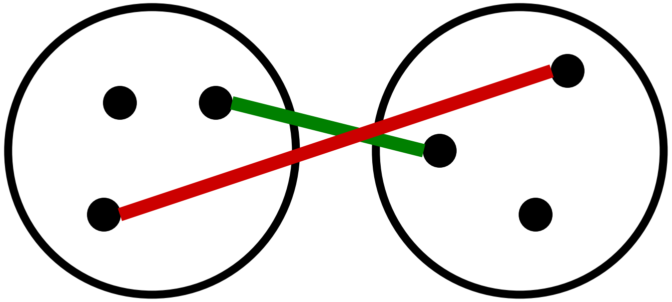


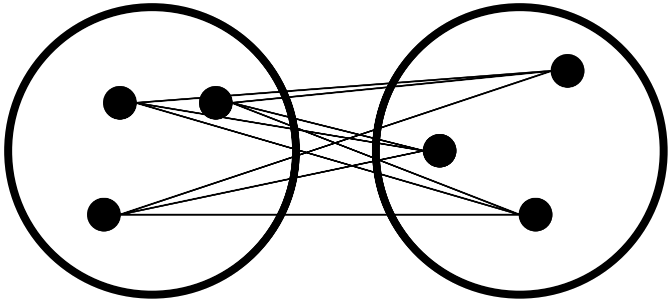
# Agglomerative Clustering

- How should we define "closest" for clusters with multiple elements?

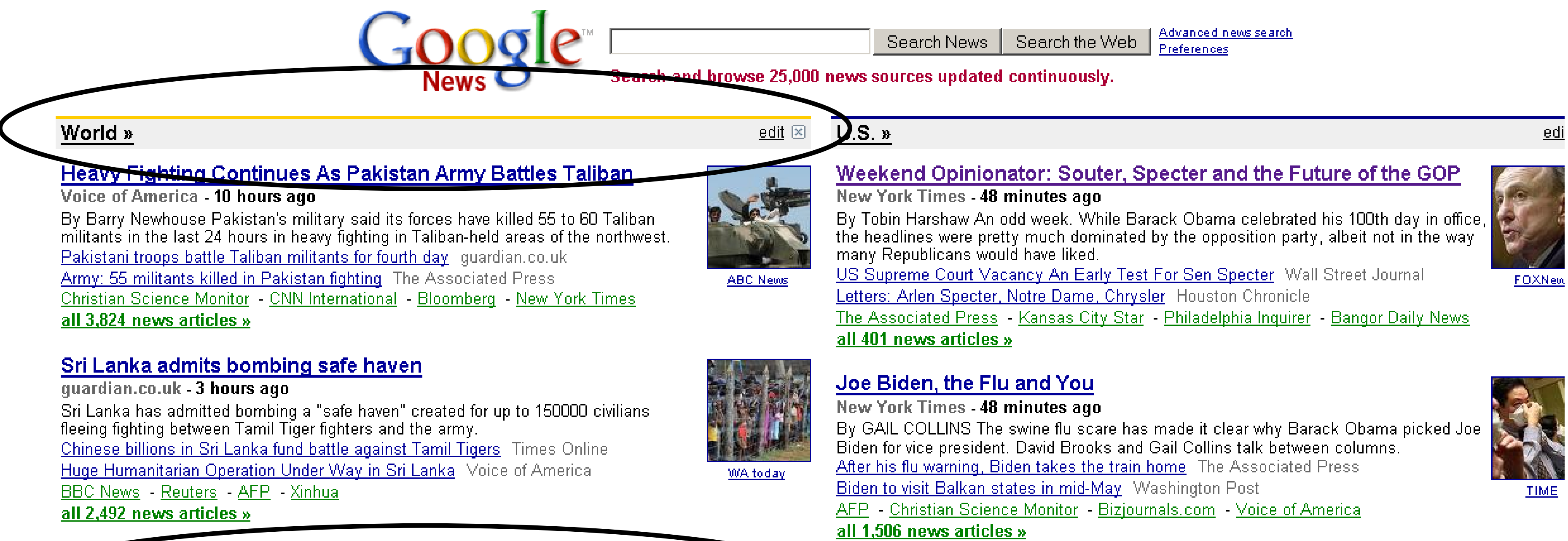
- Many options
  - Closest pair (single-link clustering)
  - Farthest pair (complete-link clustering)
  - Average of all pairs

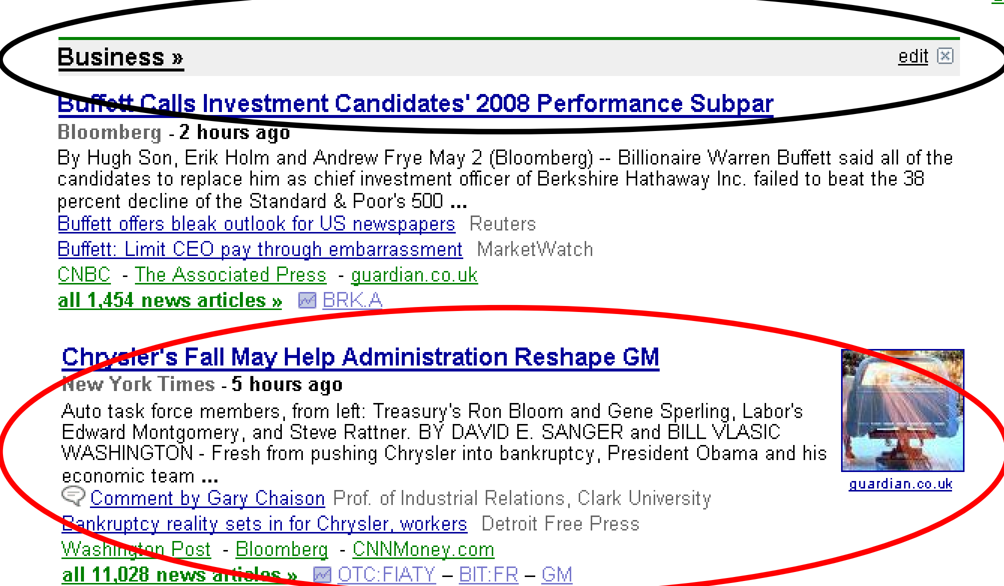
- Different choices create different clustering behaviors





# Clustering Application




Top-level categories: supervised classification

Story groupings: unsupervised clustering

# Recap of today

- Building on perceptrons:
  - MIRA
  - SVM
  - Non-parametric – kernels, dual perceptron
- Nearest neighbor classification
- Clustering
  - K-means
  - Agglomerative

# Coming Up

- Neural networks
- Decision trees

- Advanced topics: applications,…