

AUTOMATIC VISUALIZATION AND CLASSIFICATION OF HIGH DIMENSIONAL DATA

Sam Roweis

University of Toronto
Department of Computer Science

[Google: “Sam Toronto”]

with Jacob Goldberger, Ruslan Salakhutdinov, Geoff Hinton

NPCDS – Fields Data Mining Workshop

November 11, 2005

Machine Learning Algorithms

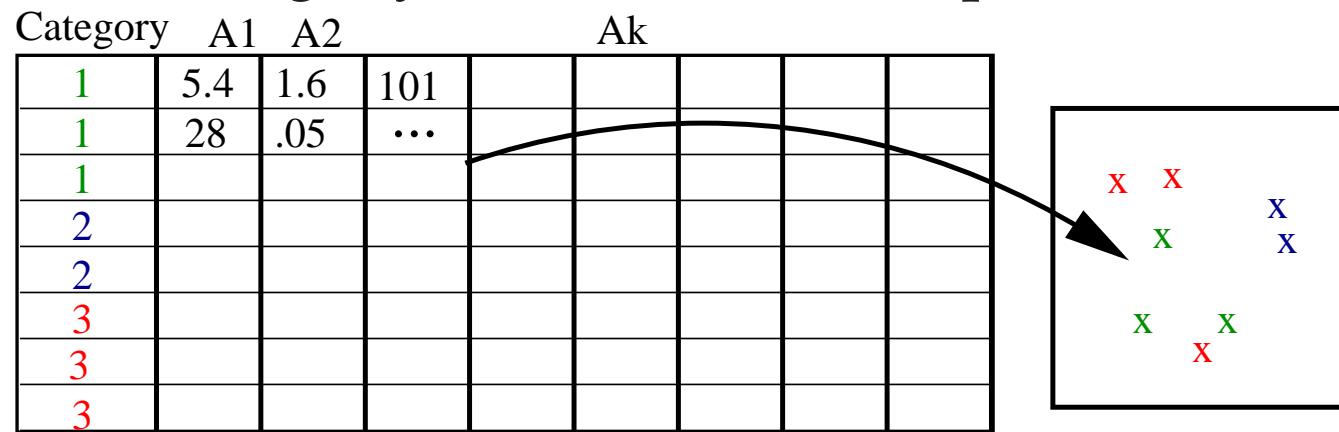
- In **machine learning** we start with a large dataset of measurements, and we want to design algorithms that automatically find some **interesting statistical structure**.
- Examples: spam vs. nonspam email detection; information extraction from web pages; tracking objects/people in videos.
- Today, I'll talk about trying to **automatically visualize** (layout in 2D or 3D) “labelled” datasets:

Category	A1	A2	Ak	...
1	5.4	1.6	101	
1	28	.05	...	
1				
2				
2				
3				
3				
3				
			Attributes (measurements)	:

- What does it mean to visualize such data?

Visualization of Labelled Datasets

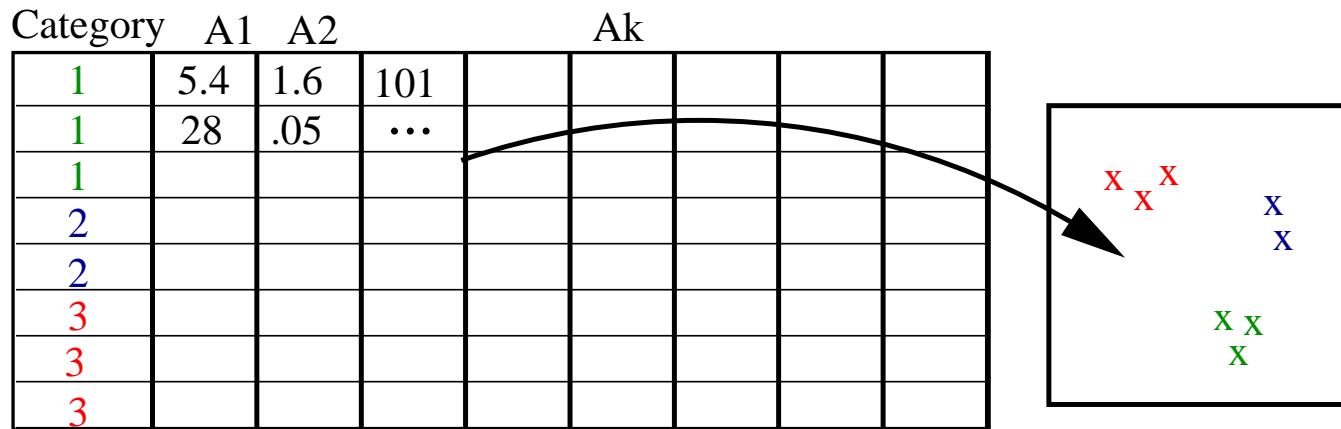
- Usually, by **visualization** we mean that we associate each record in the original database with (say) two numbers and display it as a **point in the plane**; the **category** of the record is also represented, e.g. by the **colour** of this point.



- We want the two numbers we use to display each item to depend only on the attributes of that record, but the original dataset may have hundreds (or thousands or more) attributes, so this is a form of (supervised) **dimensionality reduction**.
- To answer this question sensibly, we need to know the **goal** of visualization? What makes a **good picture** of a database?

Good Visualizations Keep Categories Together

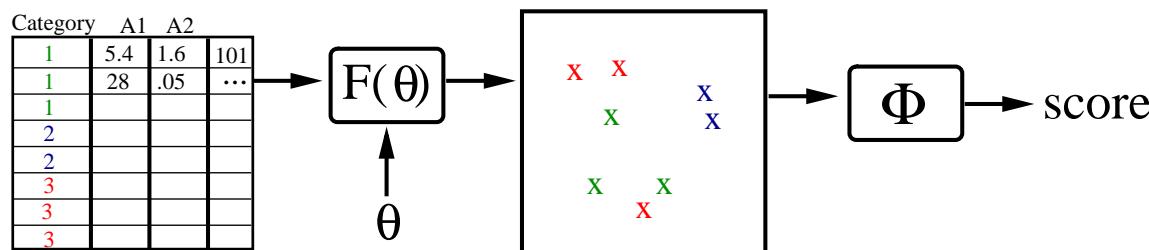
- For today, let's say that a good visualization is one which **keeps records of the same category near each other.**



- So I'm asking you to design a **function**, which takes as input the (many) attributes of a record (but not its label) and gives as output the (two) coordinates at which that record should appear in the visualization. I'll judge how well you do by looking at the picture you make and asking if **nearby points tend to have the same colour** (category).
- Furthermore, I want you to design this function **automatically**, by running an algorithm on the labelled database.

Using Optimization to Achieve our Goal

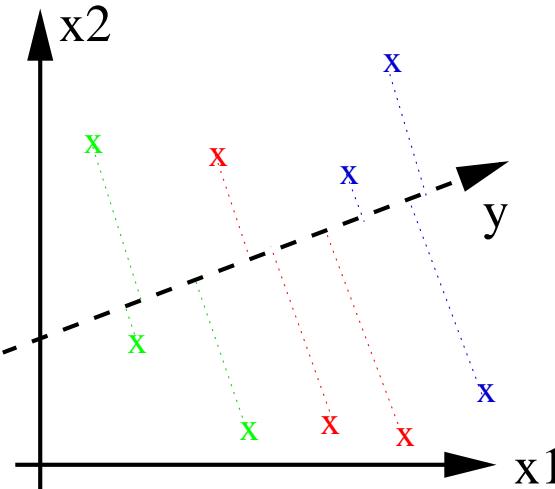
- We want to express our **goal** (design a function that makes a nice picture) as a very **precise** (quantitative) problem, which we can then get the computer to solve automatically for us.
- We'll take the standard approach of trying to convert this into a **numerical optimization** tasks.
- In order to do this, we need to specify two things:
 1. An **objective function** Φ , which will give a numerical score for how good any particular solution (picture) we make is.
 2. A **function class** $F(\theta)$ (i.e. a set of functions we are willing to consider), parameterized by some tunable knobs θ .



- The problem is now reduced to **search**: find a setting of the knobs θ so that the resulting score is as high as possible.

Function Class: Linear Projections

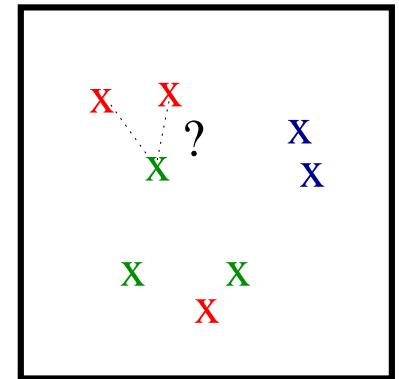
- What's the **simplest function class** you can think of that takes a large number of numerical inputs and returns two outputs?
- How about **linear projections**, i.e. matrices A of size 2 by K, where K is the number of attributes in the original data.
- Call the original attributes x and the visualization outputs y . We are going to use the simple equation $y = Ax$.
(It will be very easy to describe to someone else what we did.)



- The knobs θ are just the $2 \times K$ elements of the matrix A.

Objective: Neighbourhood Consistency

- What about the objective function Φ ? How should we quantify how good a particular projection (picture) is?
- One idea: try to measure the **neighbourhood consistency** of the category labels in the projection. For each point (item), look at a few of its nearest neighbours and ask, are they all (or most of them) in the same category as the point itself? If so, score 1, otherwise, score 0. Add up this score over all records.
- Using this idea, if I gave you a finite set of projections $\{A\}$ to chose between, you could pick the best by scoring each one.
- Obvious next question: in our **continuously parameterized** family of functions (projections), how can we search for the one which maximizes this neighbourhood consistency score?

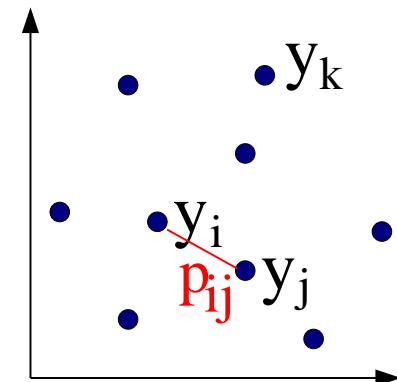


Consistency Score is Hard to Optimize

- Bad news: the consistency score I defined is a very difficult function to optimize with respect to A .
- Why? Because it depends only on the neighbourhood graph and thus is a **highly discontinuous** function of the projection. In fact, it is **piecewise constant**.
- In particular, most finite changes to the projection matrix A will not change the neighbourhood graph at all, and thus the score will remain constant; however at certain settings of A , an infinitesimal change *will* alter the neighbour graph and thus change the score by a finite amount.
- We need a **smoother** (or at least **continuous**) cost function.
- And what about the neighbourhood over which we measure consistency? How do we define that exactly?

Randomized Neighbour Selection

- Idea: instead of picking a fixed number of nearest neighbours for each point and examining their categories, **select a single neighbour stochastically**, and look at the **expected counts** for each category.



- Imagine that each point i selects other points j as its neighbour with a probability p_{ij} based on its **Euclidean distance in the projection** $d_{ij} = \|y_i - y_j\|^2 = \|\mathbf{Ax}_i - \mathbf{Ax}_j\|^2$:

$$p_{ij} = \frac{e^{-d_{ij}}}{\sum_{k \neq i} e^{-d_{ik}}} \quad p_{ii} = 0$$

- The expected score (frac. time we get 1) for point i is p_i^+ :

$$p_i^+ = \sum_{j \in C_i} p_{ij} \quad C_i = \text{points in same category as } i$$

- We introduced randomization, then averaged it away again.

New Objective: Expected Score

- Our new objective will be to maximize the **expected total score**:

$$\phi = \sum_i p_i^+ = \sum_i \sum_{j \in C_i} p_{ij} = \sum_i \sum_{j \in C_i} \frac{e^{-d_{ij}}}{\sum_{k \neq i} e^{-d_{ik}}}$$

where $d_{ij} = (\mathbf{A}x_i - \mathbf{A}x_j)^\top(\mathbf{A}x_i - \mathbf{A}x_j)$.

- This is a much smoother function of \mathbf{A} than before, and it can be maximized using various techniques that rely on the derivatives of ϕ wrt each element of \mathbf{A} .
- In particular, the exact **gradient** of Φ with respect to the transformation matrix \mathbf{A} can be easily computed:

$$\frac{\partial \phi}{\partial A} = 2A \sum_i \left[p_i^+ \sum_{k \in C_i} p_{ik} x_{ik} x_{ik}^\top - p_i^- \sum_{j \in C_i} p_{ij} x_{ij} x_{ij}^\top \right]$$

where $x_{ij} = (x_i - x_j)$ and C_i is the category of point i .

Neighbourhood Components Analysis (NCA)

- NCA is a new visualization algorithm which **learns a linear projection A of the original measurements, after which nearby points tend to have the same category.**
The projection preserves directions which are useful for category discrimination and squashes attributes which are not informative about category identity.
- To do this, NCA maximizes the **expected consistency score**

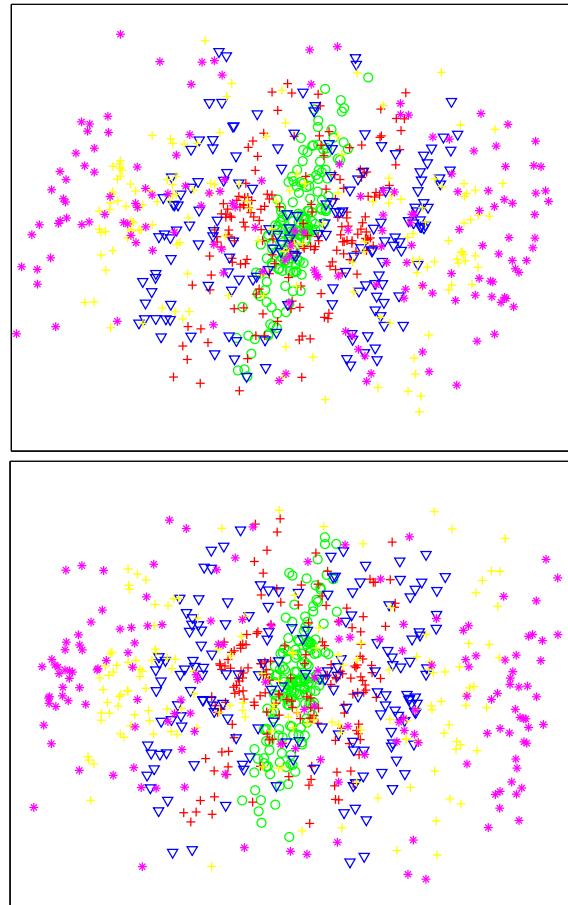
$$\phi = \sum_i \sum_{j \in C_i} \frac{e^{-(\mathbf{A}x_i - \mathbf{A}x_j)^\top (\mathbf{A}x_i - \mathbf{A}x_j)}}{\sum_{k \neq i} e^{-(\mathbf{A}x_i - \mathbf{A}x_k)^\top (\mathbf{A}x_i - \mathbf{A}x_k)}}$$

with respect to the matrix A. (e.g. using gradient descent)

- To make the picture, just plot the two dimensional coordinates \mathbf{y} where $\mathbf{y} = \mathbf{A}\mathbf{x}$, and \mathbf{x} are the original attributes.

Example 1: Concentric Rings

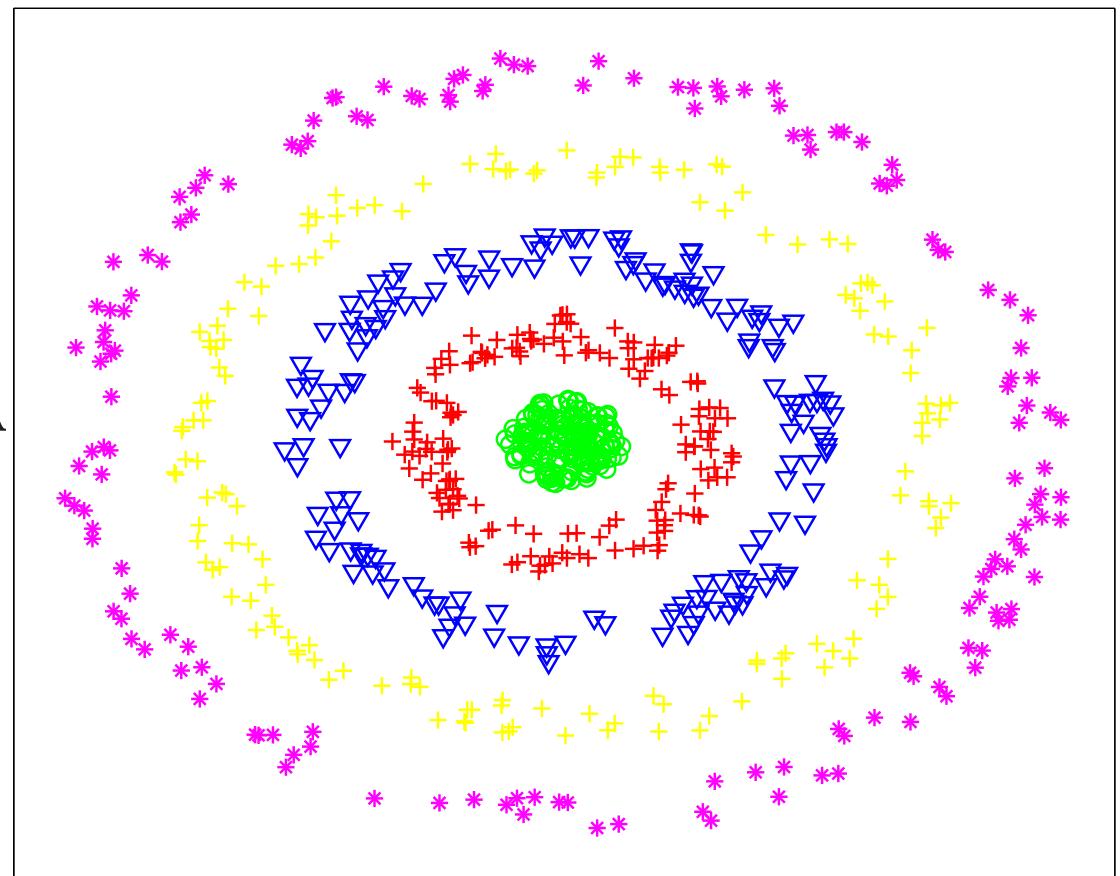
- Synthetic data, D=500 attributes, 2 of which hide 5 concentric rings (categories), all others contain noise of different scales.



LDA

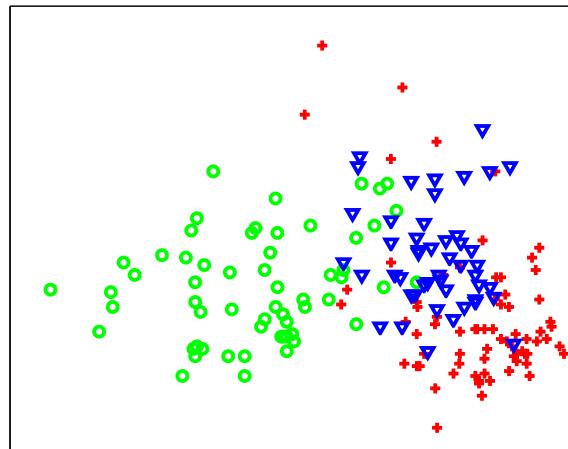
NCA

PCA

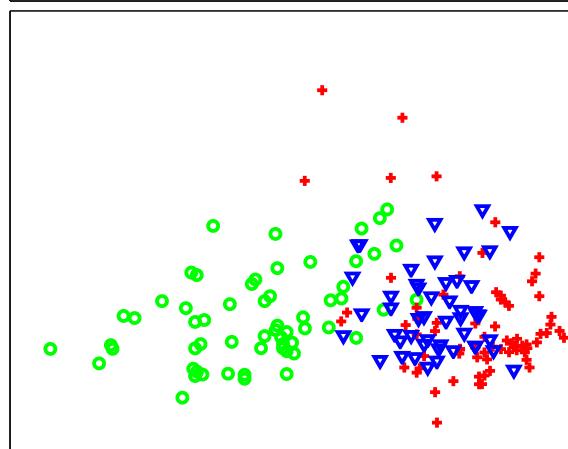


Example 2: UCI Wine

- UCI “Wine” data, D=13 attributes, 3 categories.

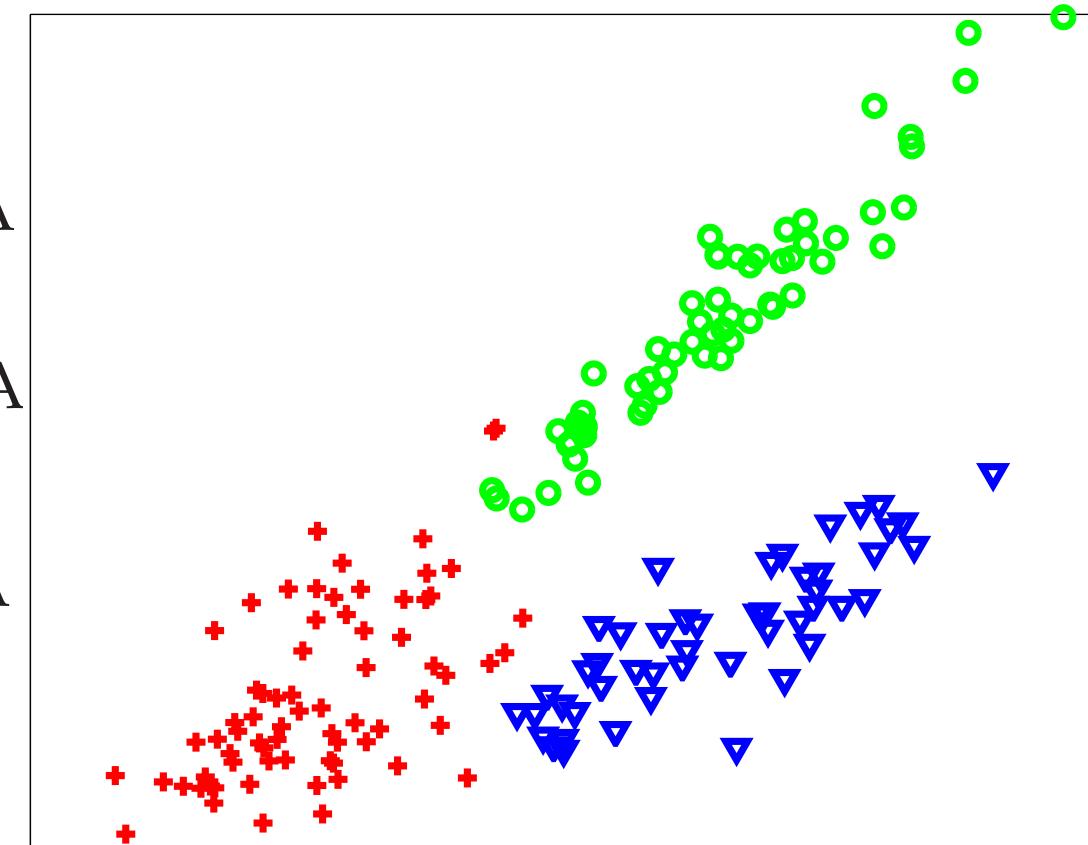


LDA



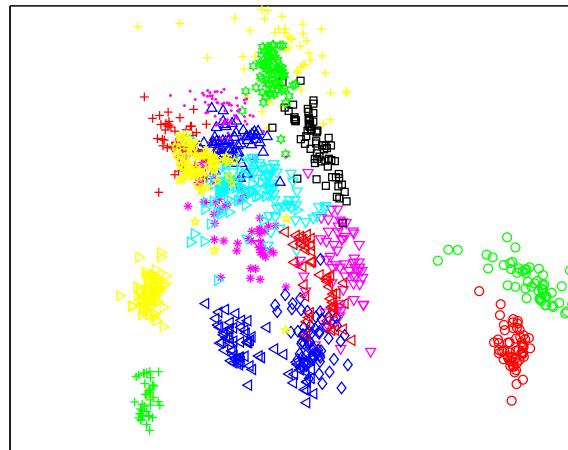
NCA

PCA

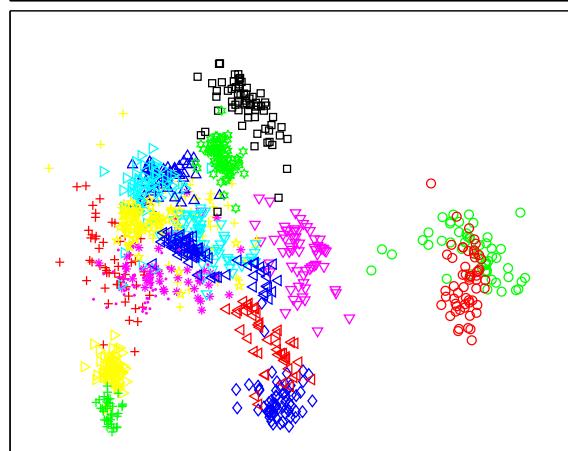


Example 3: Faces

- Grayscale images of faces taken from frames of a 20×28 video.
(18 people as category labels, $D=560$ attributes, one per pixel)

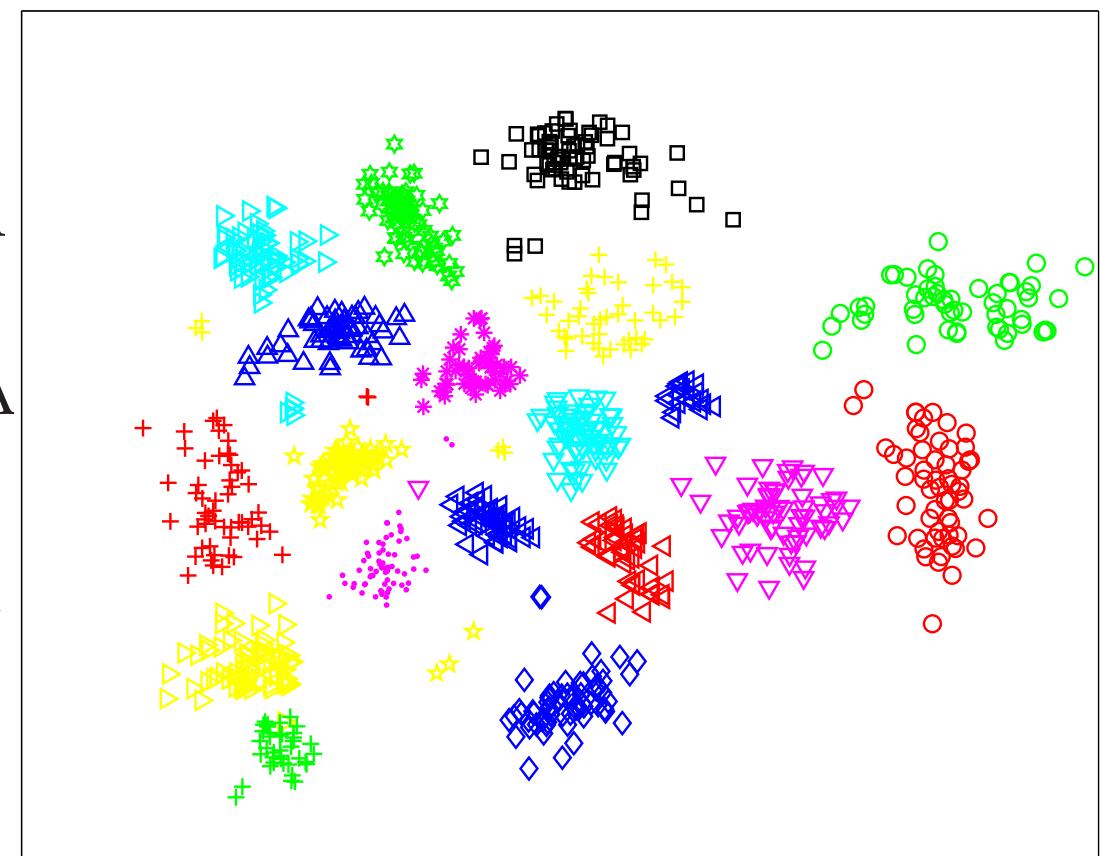


LDA



NCA

PCA



Scale of A is also learned

- Notice that not only the relative directions of the rows of A but also the **overall scale** of A is being learned.
- This means that we are effectively learning a real-valued estimate of the neighbourhood size around each point.
- Estimate = **average effective perplexity** of distributions p_{ij} :

$$\hat{K} = \exp\left(-\sum_{ij} p_{ij} \log p_{ij}/N\right) \quad \text{or} \quad (1/N) \sum_i \exp\left(-\sum_j p_{ij} \log p_{ij}\right)$$

- If the learning procedure wants to **reduce** the effective perplexity (use fewer neighbours) it can **scale up** A uniformly; similarly by **scaling down** all the entries in A it can **increase** the perplexity of and effectively average over more neighbours during the stochastic selection.

But there's even more good news...

There are just two rules for success:

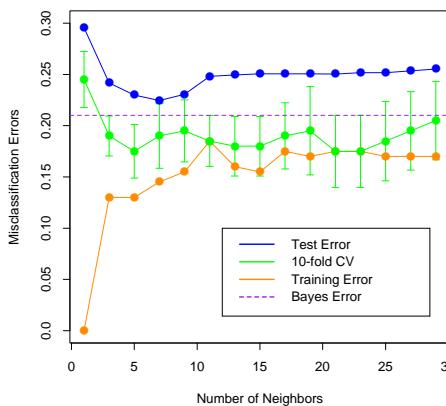
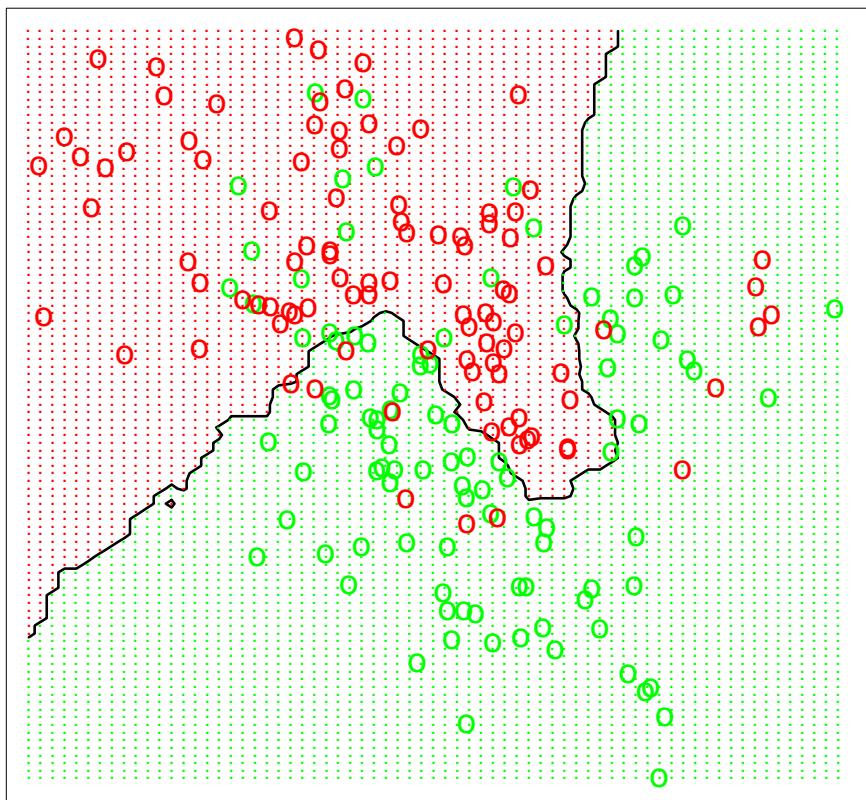
1. Never tell all you know.

- Roger H. Lincoln

Nearest Neighbour Classification

- KNN is a simple yet surprisingly effective classification procedure
- Decision surfaces are **nonlinear**.
- Nonparametric, so **high capacity without training**.
- Quality of predictions automatically improves with more data.
(Asymptotically optimal.)
- Only a single parameter K ; easily tuned by cross-validation.

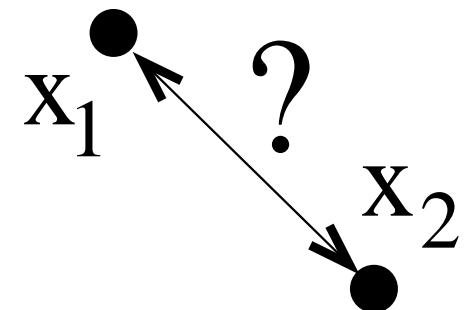
15-Nearest Neighbor Classifier



Problems with Nearest Neighbour

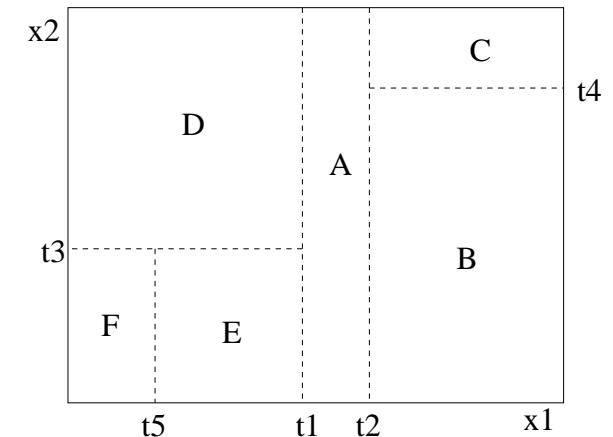
- What does “nearest” mean?

Need to specify a **distance metric** on the input space.



- **Computational cost:** must store and search through entire training set at test time.

(In low dimensional input spaces, this can be alleviated by thinning training set and building fancy data structures like KD-trees.)



- We can use NCA again to learn a distance metric for KNN classification by making A a square matrix. If need be, we can keep A rectangular and significantly reduce computation at the expense of often small performance loss.

Classification Results

- UCI “Wine”, N=178, D=13 attributes, 3 categories.
89 cases for training, 89 cases for testing.

Test errors using A=13x13: Euclid.=30%;Whiten=25%;NCA=7%

Test errors using A=2x13: LDA=28%; PCA=31%; NCA=5%

- Grayscale images of faces taken from frames of a 20x28 video.
(18 people as category labels, D=560 attributes)
100 cases for training, 900 for testing

Test errors using A=560x560:

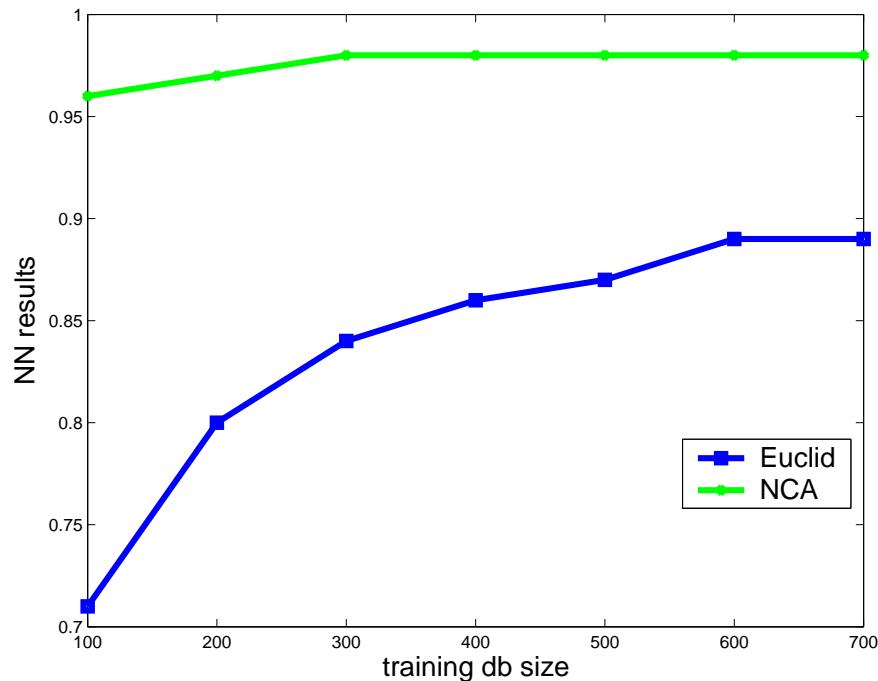
Euclid.=18%;Whiten=22%;NCA=4%

Test errors using A=2x13: LDA=25%; PCA=37%; NCA=5%

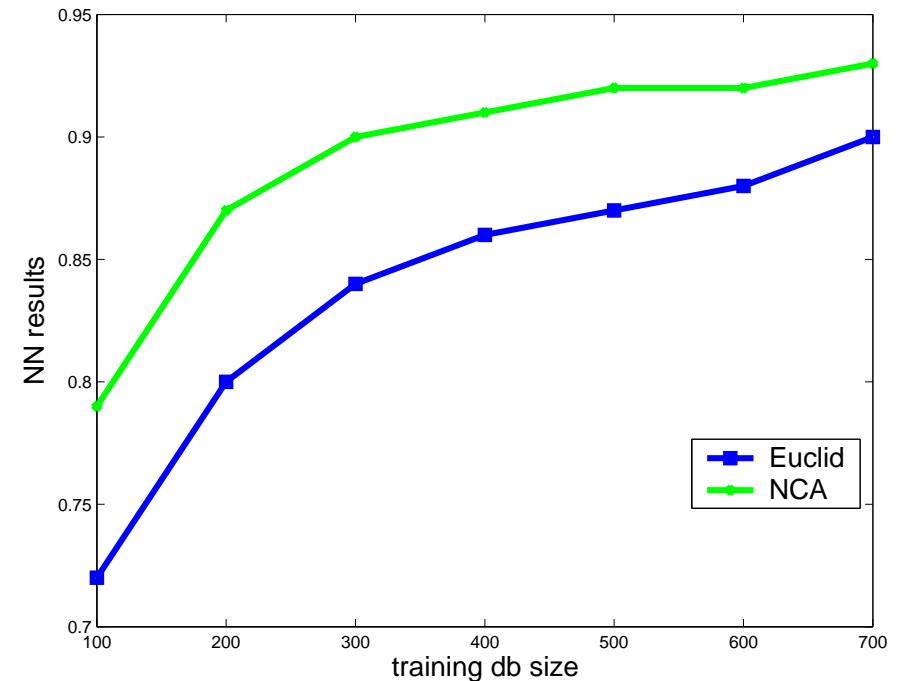
- In the above K was chosen by leave-one-out cross validation.
But NCA estimates K for us automatically using the scale of A;
we can either take the average perplexity as a global K or, even
better, we can store **local estimates of K** (hard using CV).

Preliminary Experiments: Digits

- 8x8 grayscale (8bit) images of handwritten digits. $D=64$.
- First, we use $d = D = 64$ and just learn a metric:



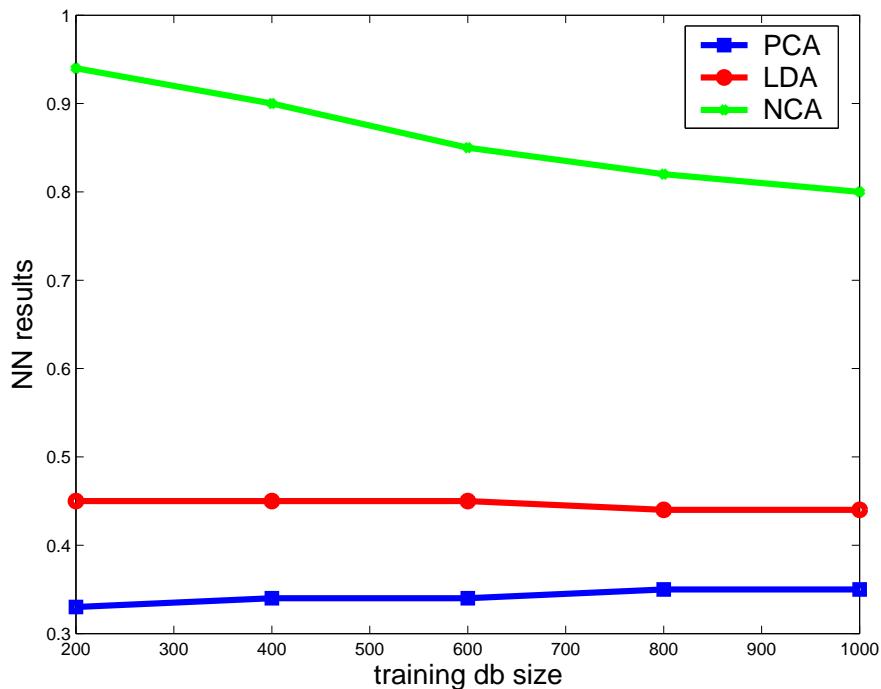
Training Error



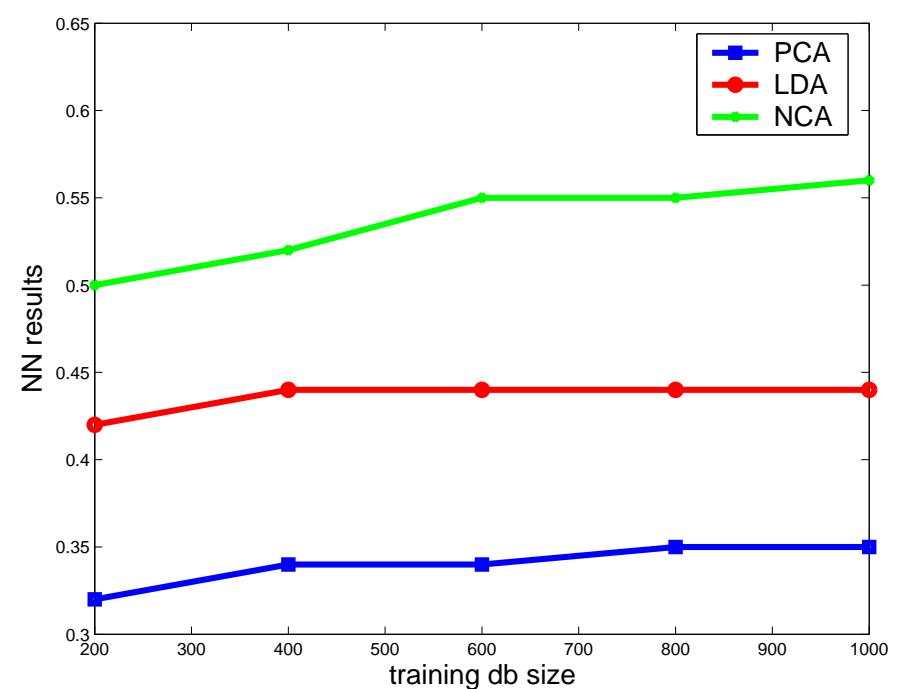
Testing Error

Preliminary Experiments: Digits

- 8x8 grayscale (8bit) images of handwritten digits. $D=64$.
- Now, we use $d = 2$ and see how well we can do even under such a severe constraint:



Training Error



Testing Error

Fast KNN using Learned A

- By using an highly rectangular \mathbf{A} ($d \ll D$), we significantly reduce the computational load of KNN.
Price: **restrict allowable metrics to be those of rank at most d .**
- Approach: apply NCA to find the transformation \mathbf{A} ; store only the projections of the training points $y_n = Ax_n$ (as well as their labels).
- At test time, we classify a new point x_{test} by first computing its projection $y_{test} = Ax_{test}$ and then doing KNN classification on y_{test} using the y_n and a simple Euclidean metric.
- If d is relatively small (say less than 10), we can preprocess the y_n by building a **KD-tree** or a **ball-tree** to further increase the speed of search at test time.
- The storage requirements of this method are $O(dN) + Dd$ compared with $O(DN)$ for KNN in the original input space.

Discussion Points

- Why restrict to **linear transformations** ?
Hard to overfit, compact to represent, **fast at test time**.
- In fact, we can even restrict to **diagonal A** for KNN.
- We could also maximize the expected log probability of correct classification, which is the **chance of a perfect labelling**:

$$\phi = \frac{1}{N} \sum_i \log p_i^+ = \frac{1}{N} \sum_i \log \sum_{j \in C_i} \frac{e^{-d_{ij}}}{\sum_{k \neq i} e^{-d_{ik}}}$$

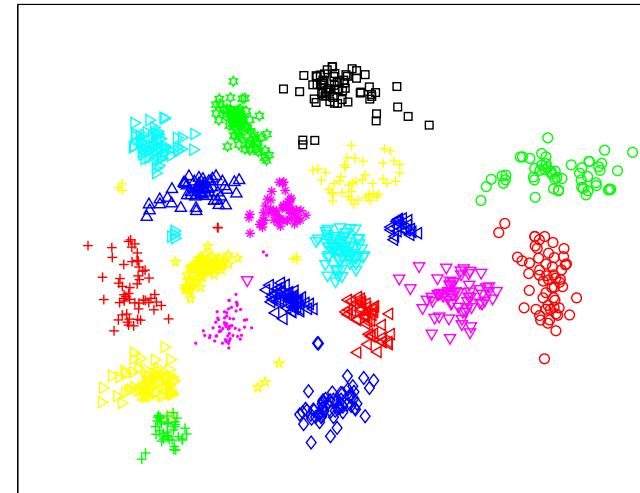
(This corresponds to KL dist. from the true distribution.)
Results are very similar with this cost if the training data does not have any outliers.

Still not convex, but $\sum_{ij} \log p_{ij}$ is...

- Observation: for both costs, **tall maxima are good**.
In other words, we rarely see overtraining.

Conclusions

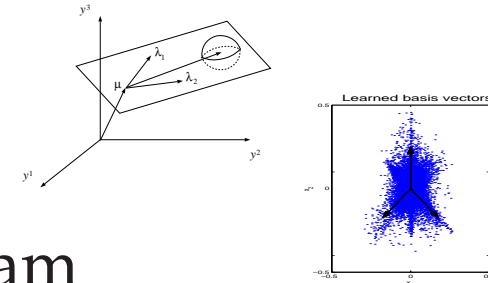
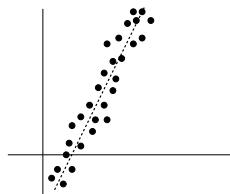
- Often we want to massively reduce the size of a database, by summarizing all the attributes for each record using only a few (say 2) numbers. Also useful for fast KNN classification.
- Today I told you about an algorithm called NCA which does this automatically by learning a good linear projection.
- NCA assumes nothing about:
 - the form of the class distributions (e.g. Gaussian, connected, convex)
 - the shape of the separating surface (e.g. linear, polynomial)
- **Very surprising how far you can go with just linear mapping.**
It turns out extremely good mappings exist, and now we have a handle on how to find them.



Other Linear Dimensionality Reduction

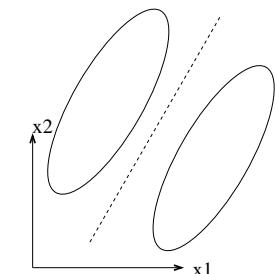
- Unsupervised:

- PCA: chases variance
 - Factor Analysis: chases correlation
 - ICA: chases heavy/light tails in histogram



- Supervised:

- CCA/PLS: chase cross-correlation
 - LDA (Fisher's Discriminant)
(makes a Gaussian assumption about each class)



Other Related Work

- Local Methods:
 - Flexible Metric NN [Friedman]
 - Tangent distance [Simard et. al], local hyperplane [Vincent & Bengio], discriminant adaptive NN [Hastie & Tibshirani]
 - SMV Regularized [Domeniconi & Gunopoulos]
- Global Methods:
 - L_2 cost function, diagonal A [Lowe]
 - Normalized PCA/LDA [Koren & Carmel]
 - Relevant Components Analysis [Weinshall et. al]
 - Density Driven Feature Extraction [Torkkola]
 - Kernel Dimensionality Reduction [Fukumizu, Bach, Jordan]
 - Information Bottleneck, Sufficient Dimensionality Reduction [Tishby et. al]