NEIGHBOURHOOD COMPONENTS ANALYSIS

Sam Roweis

University of Toronto
Department of Computer Science

[Google: "Sam Toronto"]

with Jacob Goldberger, Geoff Hinton & Ruslan Salakhutdinov

Machine Learning Summer School, Taiwan **July24, 2006**

Basic Classifiers Perform Annoyingly Well

Machine Learning, 11, 63–91 (1993)
© 1993 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Very Simple Classification Rules Perform Well on Most Commonly Used Datasets

ROBERT C. HOLTE HOLTE@CSI.UOTTAWA.CA
Computer Science Department, University of Ottawa, Ottawa, Canada K1N 6N5

Editor: Bruce Porter

Abstract. This article reports an empirical investigation of the accuracy of rules that classify examples on the basis of a single attribute. On most datasets studied, the best of these very simple rules is as accurate as the rules induced by the majority of machine learning systems. The article explores the implications of this finding for machine learning research and applications.

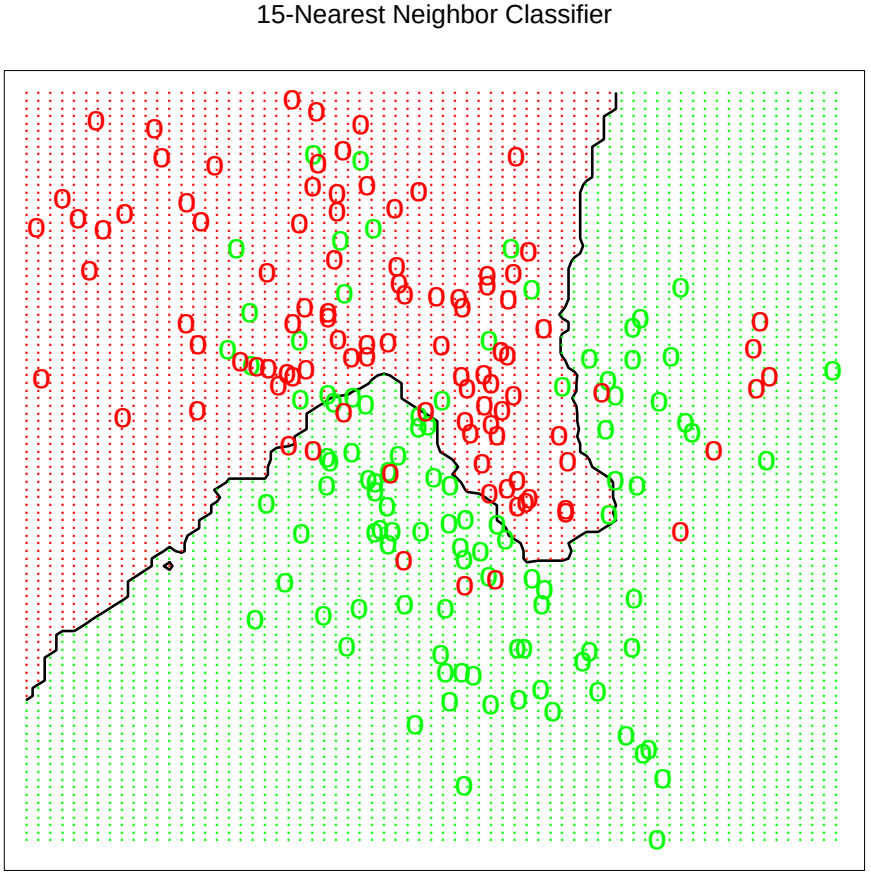
Keywords: empirical learning, accuracy–complexity tradeoff, pruning, ID3

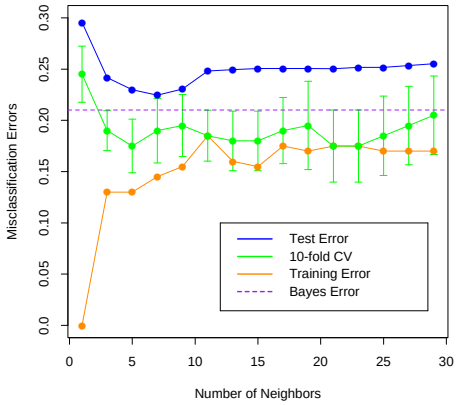
1. Introduction

The classification rules induced by machine learning systems are judged by two criteria: their classification accuracy on an independent test set (henceforth "accuracy"), and their complexity. The relationship between these two criteria is, of course, of keen interest to the machine learning community.

K Nearest Neighbour Classification

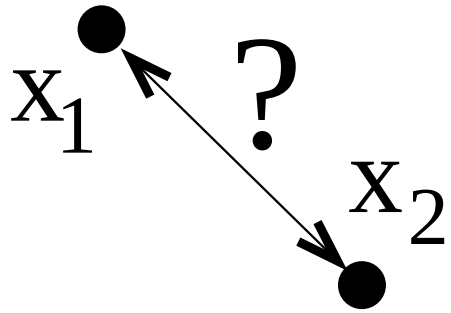
- K-NN is a simple yet surprisingly effective classification procedure
- Decision surfaces are **nonlinear**.
- Nonparametric, so **high capacity without training**.
- Quality of predictions automatically improves with more data. (Asymptotically optimal.)
- Only a single parameter K; easily tuned by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



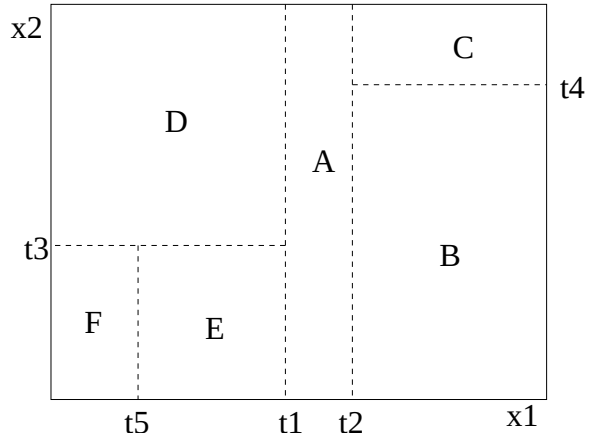


Problems with Nearest Neighbour

- What does "nearest" mean? Need to specify a **distance metric** on the input space.



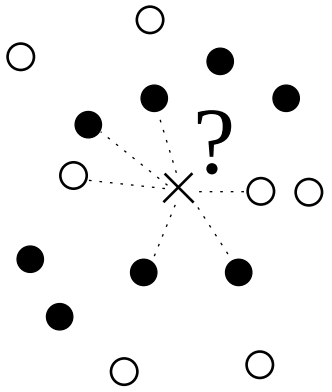
- **Computational cost**: must store and search through entire training set at test time.
  (In low dimensional input spaces, this can be alleviated by thinning training set and building fancy data structures like KD-trees.)



- Today I'll discuss how to learn a distance metric; and if you need to, how to significantly reduce computation cost at the expense of often small performance loss.

Learning a Distance Metric

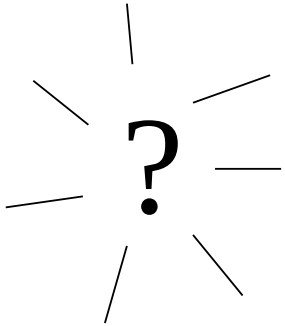
- Q: **What is the right distance metric for KNN classification?**
  A: **The one that optimizes test error!**



- Let's try to approximate this by the one which optimizes training error, defined using **leave-one-out cross validation**.
- So if I gave you a finite set of distance metrics to chose between (and I told you K), you could pick the best one.
- Obvious next question: if I gave you a **continuously parameterized** family of metrics to search through, could you find the one which maximizes LOO classification performance?
- And what about K...?

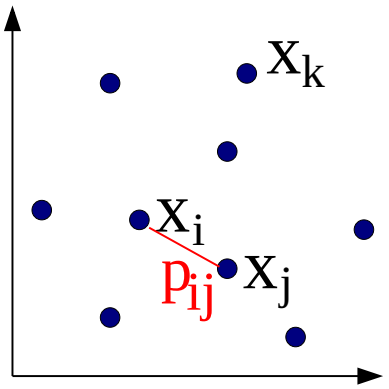
Cross-Validation Performance is Hard to Optimize

- Classification performance on held-out data is a very difficult cost function to optimize with respect to a distance metric.
- Why? Because LOO error is a **highly discontinuous** function of the distance metric and thus of the underlying parameters if the metric is from a continuous family.
- In particular, an infinitesimal change in the metric can alter the neighbour graph and thus change the validation performance by a finite amount.
- We need a **smoother** (or at least **continuous**) cost function.



Stochastic Neighbour Selection

- Idea: instead of picking a fixed number K of nearest neighbours, and voting their classes, **select a single neighbour stochastically**, and look at the **expected votes** for each class.



- Imagine that each point i selects other points j as its neighbour with a probability p_{ij} based on the **softmax of the distance** d_{ij}:

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

- The fraction of the time that i will be correctly labeled is:

$$p_i^+ = \sum_{j \in C_i} p_{ij}$$

Expected Leave-One-Out Error

- The **expected** leave-one-out classification performance is:

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

- This is the objective function we will try to maximize during learning. It is **much smoother with respect to the distances** $\{d_{ij}\}$ than the actual leave one out classification error.
- Notice that there is **no explicit parameter** K.
  (More on this later.)

Quadratic Metrics

- Now the idea is to learn the metric by adjusting the d_{ij} so as to maximize the **expected stochastic classification score** ϕ.
- We will restrict ourselves to the simplest possible metrics, namely **quadratic (Mahalanobis)** distance measures:

$$d_{ij} = (x_i - x_j)^\top \mathbf{Q}(x_i - x_j)$$

  where x_i is the input vector for the i^{th} training case and $\mathbf{Q}$ is a symmetric, positive semi-definite matrix.

- We can rewrite this using the eigendecomposition of $\mathbf{Q}$:

$$\begin{aligned} d_{ij} &= (x_i - x_j)^\top \mathbf{A}^\top \mathbf{A}(x_i - x_j) \\ &= (\mathbf{A}x_i - \mathbf{A}x_j)^\top (\mathbf{A}x_i - \mathbf{A}x_j) \\ &= (y_i - y_j)^\top (y_i - y_j) \end{aligned}$$

- In other words, this is exactly equivalent to applying **a simple (spherical) Euclidean metric** to the points $\{y_i = \mathbf{A}x_i\}$.

Optimizing Expected Performance

- We want to maximize the **expected classification performance**:

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

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

- The **gradient** with respect to the transformation matrix $\mathbf{A}$ is

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

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

- An equivalent but more efficiently computed expression:

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

Neighbourhood Components Analysis (NCA)

- **Learns a linear transformation A of the input space after which nearest neighbour performs well.**
  The transformation scales up directions which are useful for discrimination and almost projects out dimensions which are not informative about class identity.
- In particular, optimize the **expected classification performance**

$$\phi = \frac{1}{N} \sum_i \sum_{j \in C_i} \frac{e^{-(\mathbf{A}x_i - \mathbf{A}x_j)^\top (\mathbf{A}x_i - \mathbf{A}x_j)}}{\sum_{k \neq i} e^{-(\mathbf{A}x_i - \mathbf{A}x_k)^\top (\mathbf{A}x_i - \mathbf{A}x_k)}}$$

  using your favourite **aggressive optimizer**.
- Use the learned $\mathbf{A}$ to project the training set, and store $y_i = \mathbf{A}x_i$.
- At test time, compute $y_{test} = \mathbf{A}x_{test}$ and perform NN classification on y_{test} using a **simple Euclidean norm**.
- But what about K ...?

Scale of A is also learned

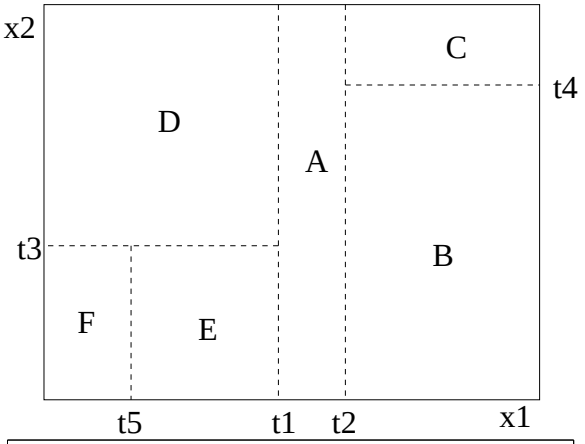
- Notice that not only the relative directions of the rows of $\mathbf{A}$ but also the **overall scale** of $\mathbf{A}$ is being learned.
- This means that we are effectively learning a real-valued estimate of the optimal neighbourhood size.
- Estimate = **average effective perplexity** of distributions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

$$\hat{K} = \exp(-\sum_{ij} p_{ij} \log p_{ij}/N) \quad or \quad (1/N) \sum_{i} \exp(-\sum_{j} p_{ij} \log p_{ij})$$

- 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.
- We can use this average perplexity as our K/ϵ at test time. Even better, we can use **local estimates of** K/ϵ (hard using CV).

Low Rank Metric $\equiv$ Nonsquare A

- Nothing in the above setup prevents us from restricting A to be a **nonsquare** matrix of size $d \times D$.
- In this case, the learned metric will be **low rank**, and the transformed inputs will lie in a lower dimensional space $\mathcal{R}^d$.
- Possible extra benefits beyond learning KNN distance metric:
  - If $d \ll D$ we can seriously **reduce storage/computation** requirements at test time by storing only the projections of the training points $y_n = Ax_n$ and using a KD-tree.
  - If $d = 2$ or $d = 3$ we can do (linear!) **visualization**.



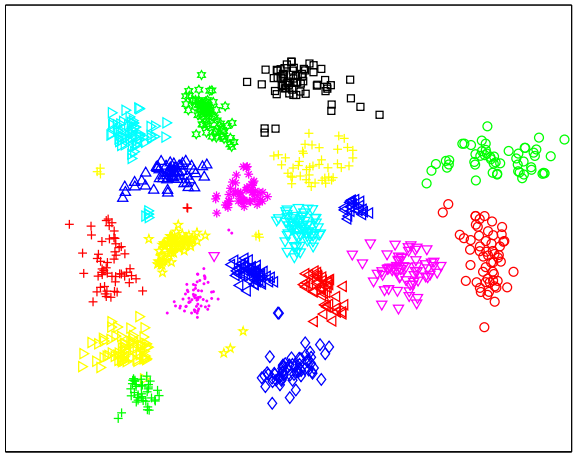
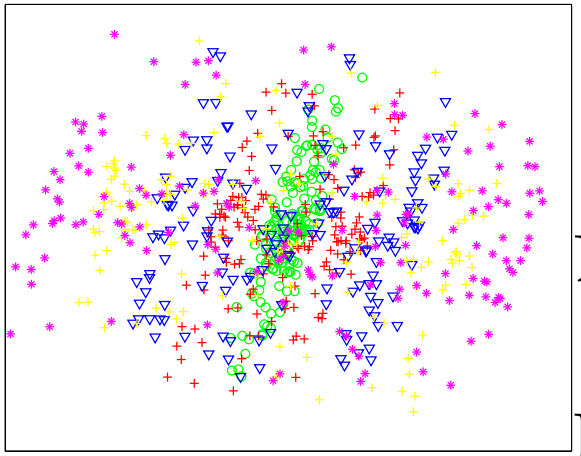


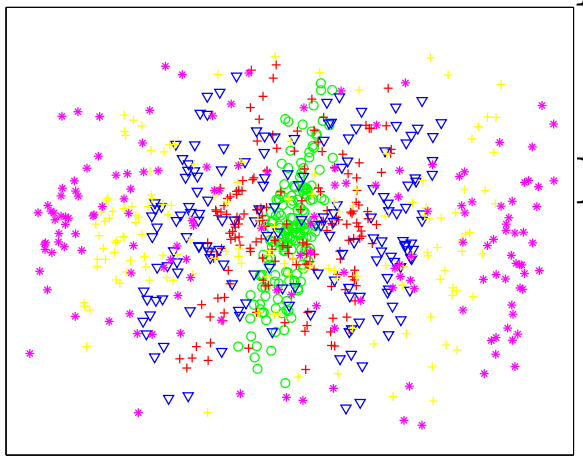
Illustration: Concentric Rings

- Synthetic data, 2 dimensions contain concentric rings, all other dimensions contain noise:

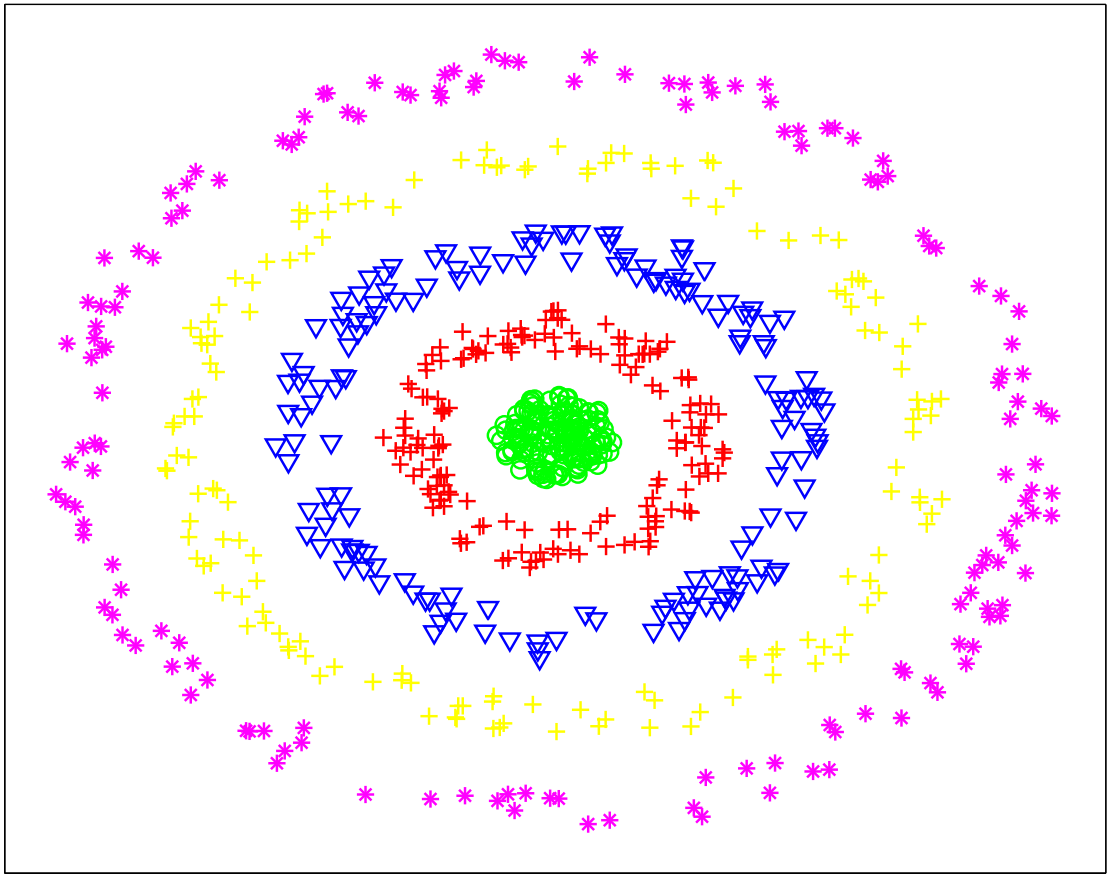


LDA

NCA

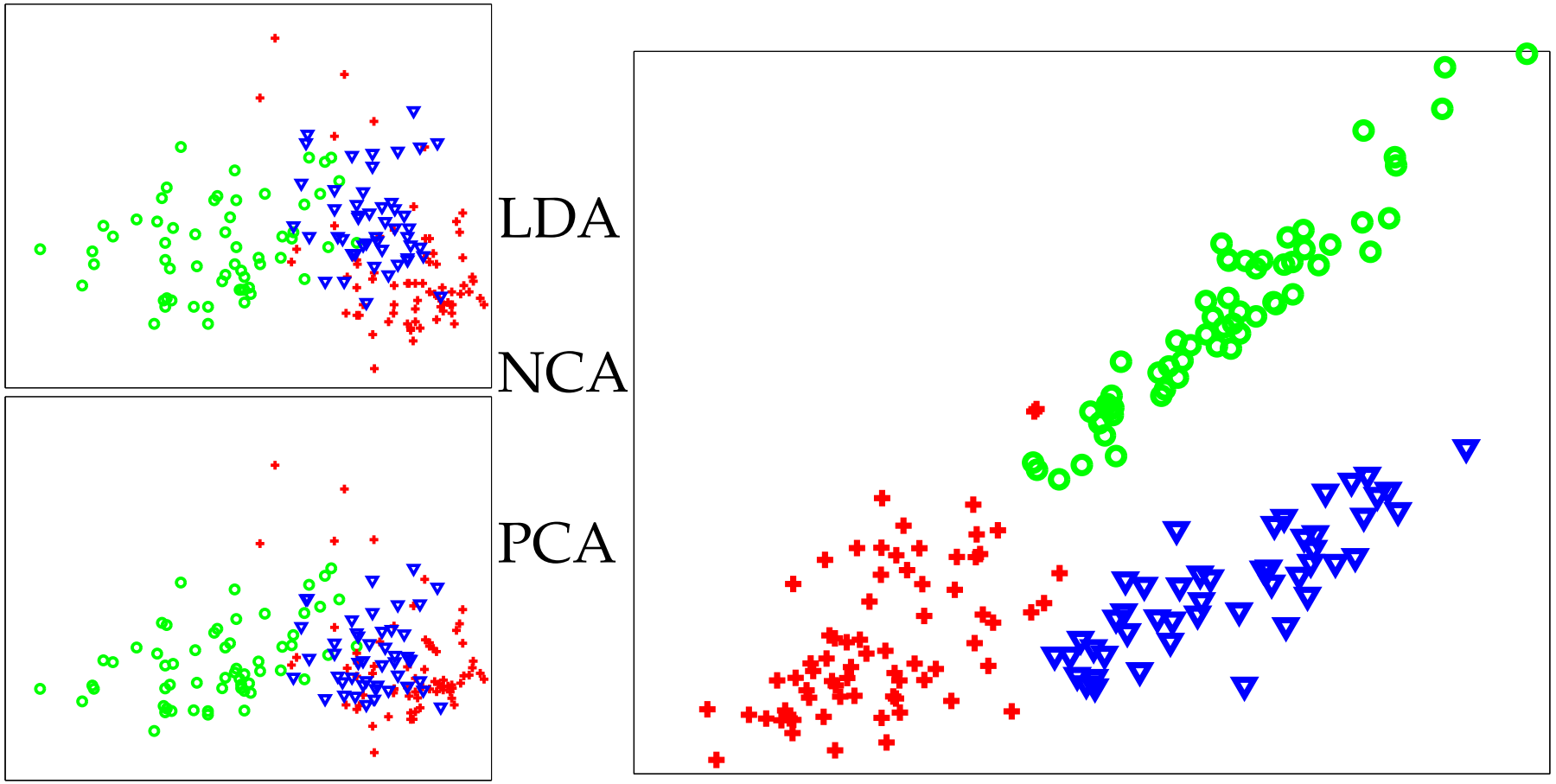


PCA



Toy Data: UCI Wine

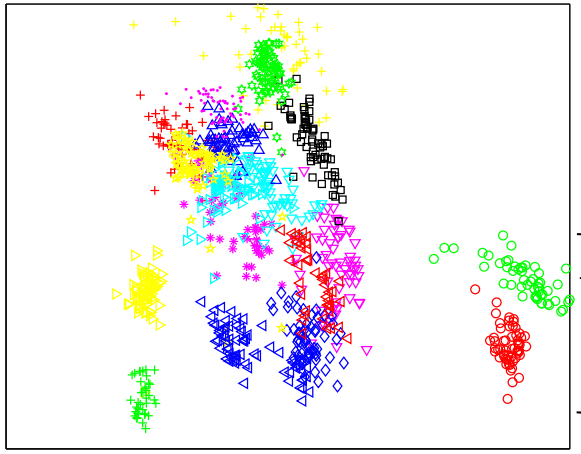
- UCI "Wine", N=178, D=13, 3 classes. Half train, half test.
- Test errors using d=D=13, and K chosen by LOO:
  Euclidean=30%;Whiten=25%;NCA=7%



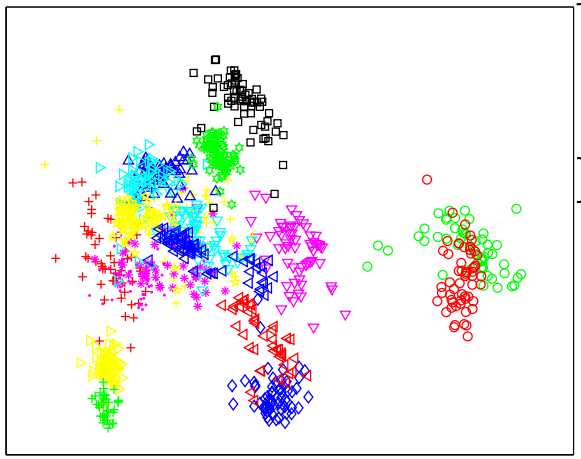
- Test errors using KNN in 2D: LDA=28%; PCA=31%; NCA=5%

Face Data

- Grayscale images of faces taken from frames of a 20x28 video. (18 people as class labels, D=560, N=100 for training, 900 test).
- Test errors using d=D=560, and K chosen by LOO:
  Euclidean=18%;Whiten=22%;NCA=4%

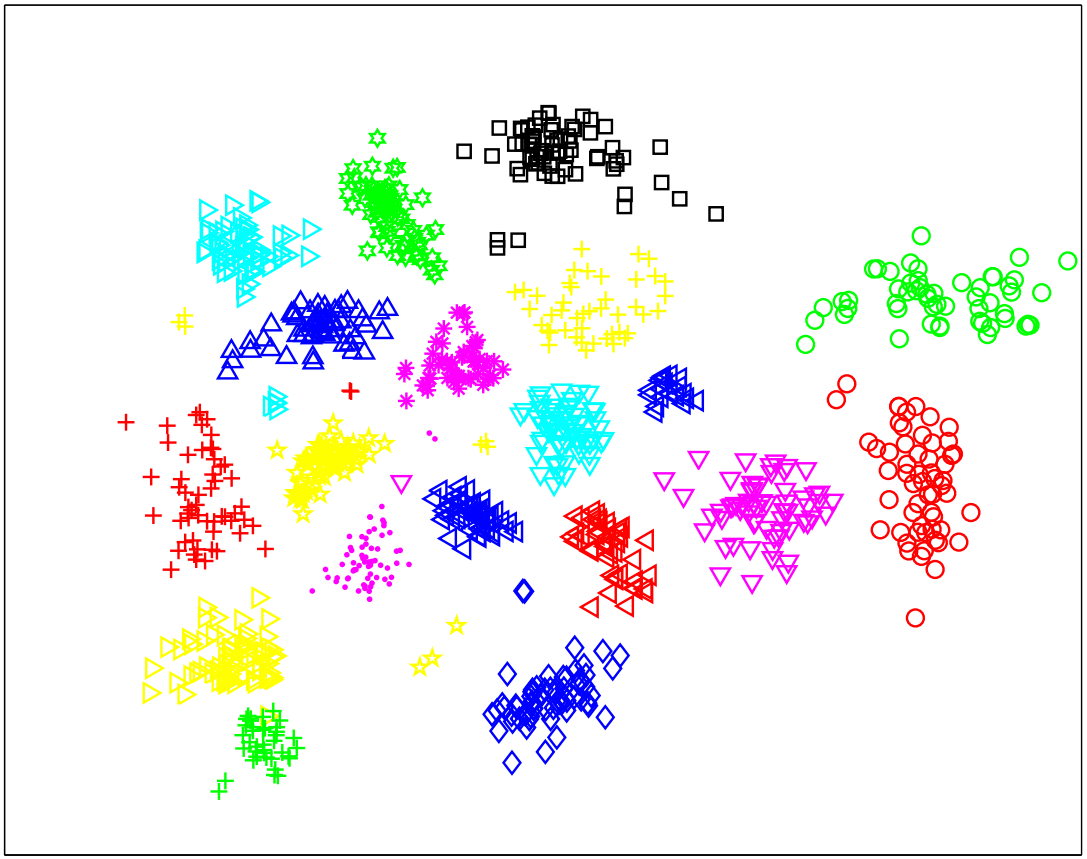


LDA



NCA

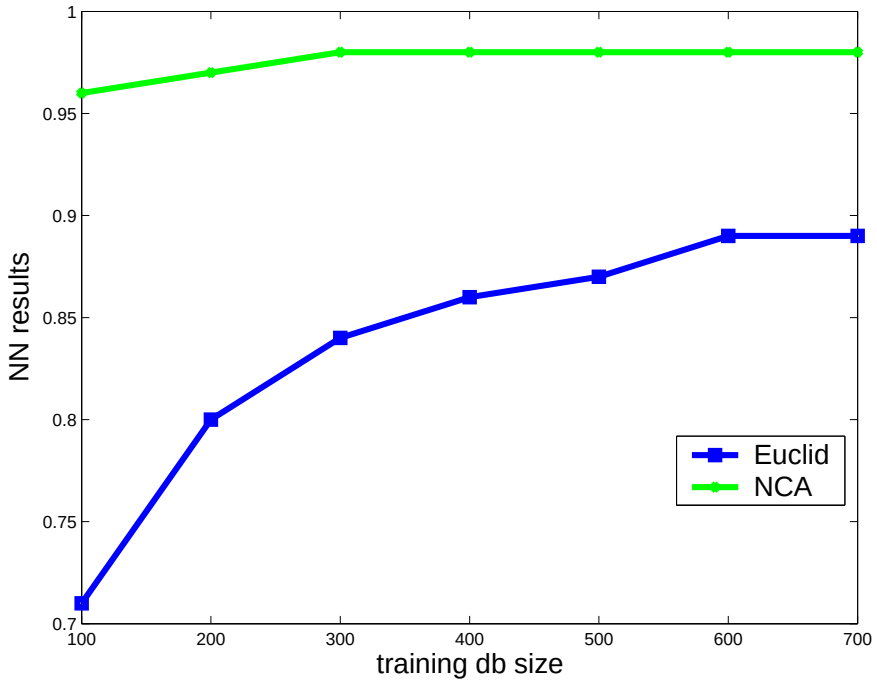
PCA



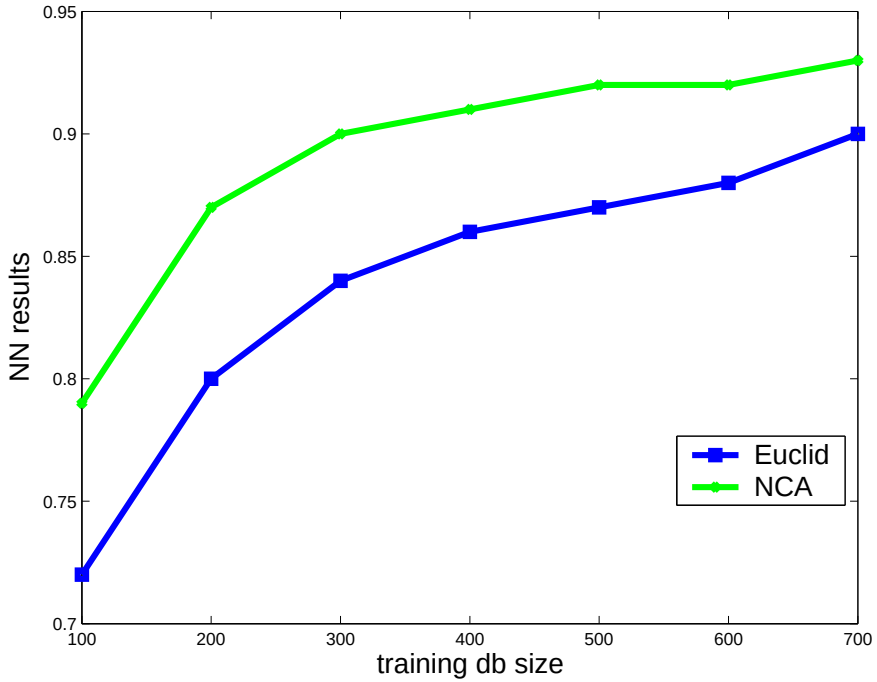
- Test errors using KNN in 2D: LDA=25%; PCA=37%; NCA=5%

Preliminary Experiments: Digits

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



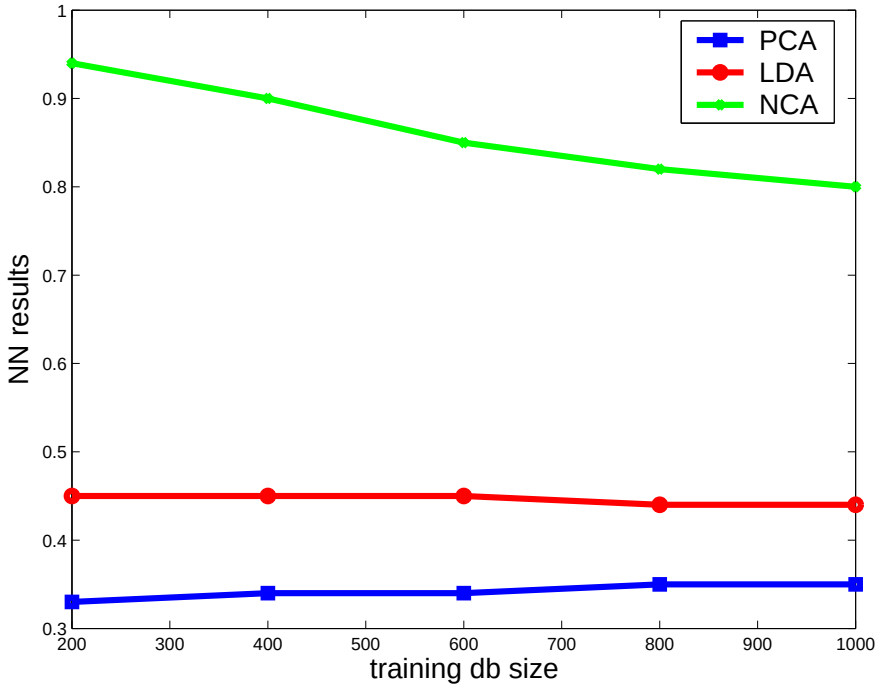
Training Error



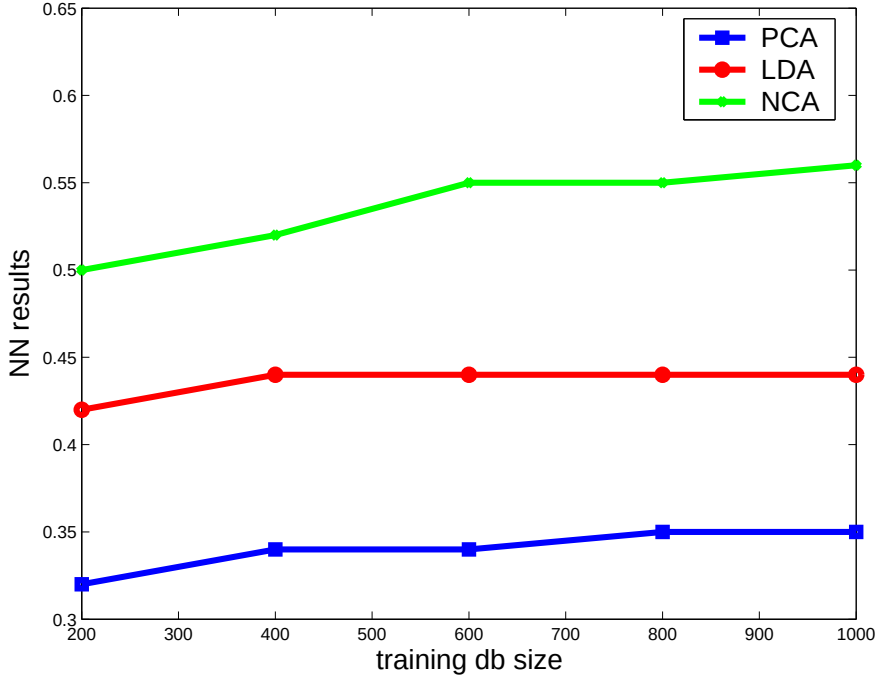
Testing Error

Preliminary Experiments: Digits with Rank-2 Metric

- The inevitable digits. USPS 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:



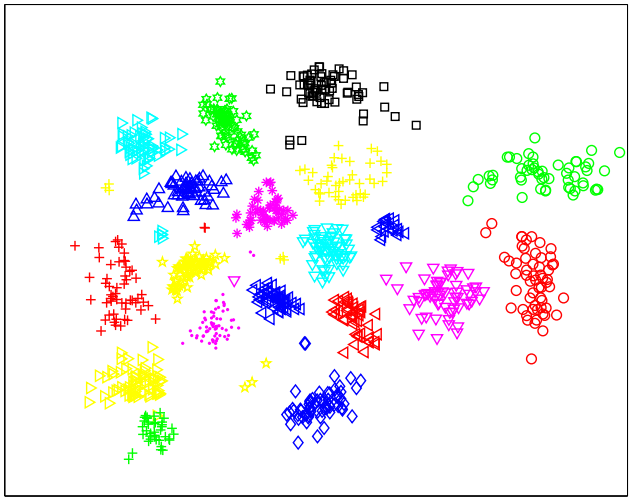
Training Error



Testing Error

Summary of NCA

- In the absence of strong prior knowledge of **how to pick your KNN distance metric**, learning seems like a good idea.
- If you are up against the wall at test time, it is an **easy way to go fast** while still doing pretty well in terms of classification.
- NCA assumes nothing about:
  – the form of the class distributions
  (e.g. Gaussian, connected, convex)
  – the shape of the separating surface
  (e.g. linear, linear in feature space)



- **Very surprising how far you can go with just linear mapping.**
  Hard to overfit, compact to represent, **fast at test time**.
  It turns out extremely good linear mappings exist, and now we have a handle on how to find them.

Maximally Collapsing Metrics

- What would the "ideal" metric $\mathbf{A}^\top\mathbf{A}$ do to the data?
  It would make the distances between all points of the same class look very small (assuming unimodality) and the distances between all points of differing classes look very large.
- If this were actually achieved, then our stochastic nearest neighbour method would induce the following distribution:

$$p^0(j|i) \propto \begin{cases} 1 & C_i = C_j \\ 0 & C_i \neq C_j \end{cases}$$

- Here's the idea: Let's find $\mathbf{A}^\top\mathbf{A}$ by minimizing the average KL divergence from this "ideal" distribution to the actual distribution induced by $\mathbf{A}$:

$$\min \sum_i \mathrm{KL}[p^0(j|i) \| p^{\mathbf{A}}(j|i)]$$

- Good news: the above objective is convex in $\mathbf{A}^\top\mathbf{A}$.

Adaptive Gaussian Kernels for SVMs

- Recent work (with Nati Srebro): a related problem is learning the shape of a Gaussian Kernel in a SVM classifier.
- Define $K(x_1, x_2|\mathbf{A}) = \exp(-\|\mathbf{A}(x_1 - x_2)\|^2)$
- Try to learn $\mathbf{A}$ by alternating:
  - fix $\mathbf{A}$ and solve for the SVM α's (Lagrange multipliers)
  - fix the α's and minimize the SVM objective wrt. $\mathbf{A}$ using BFGS (and a differentiable approx. to hinge loss)