Learning Quadratic Metrics for Classification

Jacob Goldberger, Amir Globerson
Sam Roweis

University of Toronto
Department of Computer Science

[Google: "Sam Toronto"]

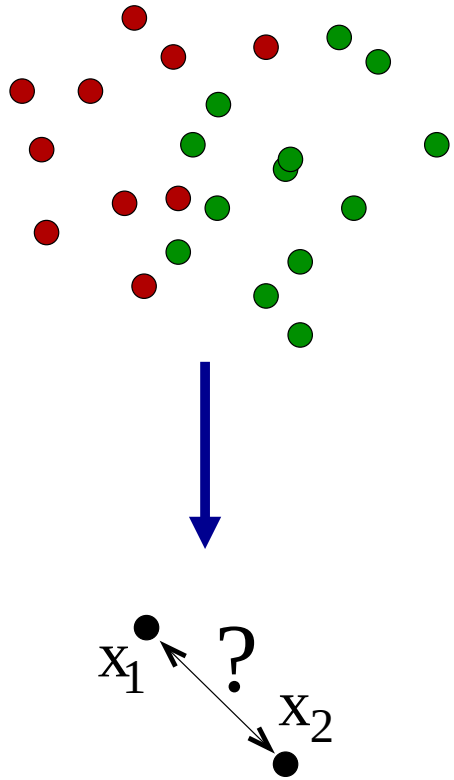
with Geoff Hinton & Ruslan Salakhutdinov

Learning to Compare Examples **NIPS Workshops 2006**

Distance Metric Learning

- Many (un)supervised machine learning algorithms rely on a distance measure (metric) which compares examples.
- Unless the problem structure strongly specifies this metric a-priori, the preferred approach is to **learn the metric** along with the rest of the parameters based on the training set.

- Today I'll focus on semi-parametric **classifiers**, e.g. KNN, SVMs, Gaussian Processes, Markov Networks, ...
- Given a labelled data set $\{\mathbf{x}_i, C_i\}$, how should we learn a metric $d[\mathbf{x}_1, \mathbf{x}_2]$ that will give **good generalization** when used in such a classifier?



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.

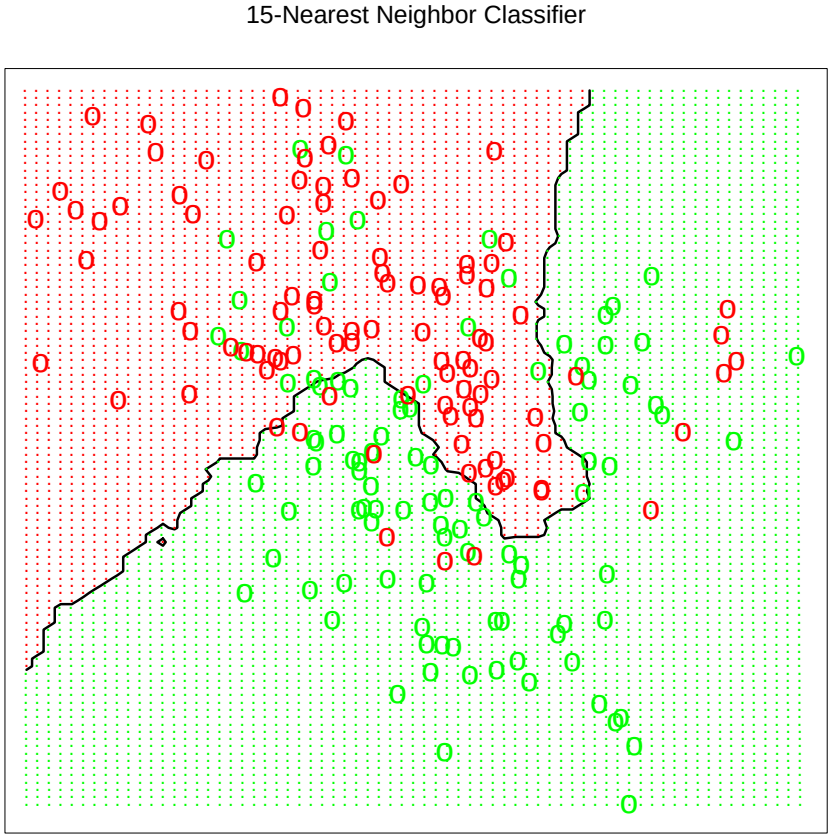
Keywords: empirical learning, accuracy–complexity tradeoff, pruning, ID3

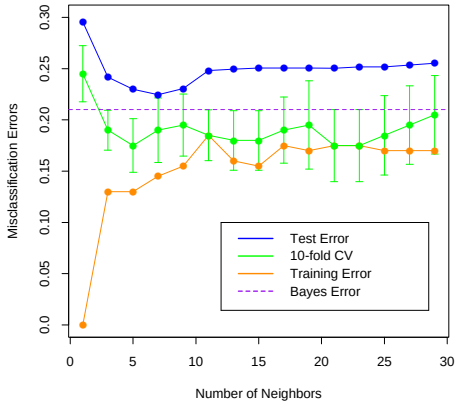
1. Introduction

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.

Instance/Memory Based Classification

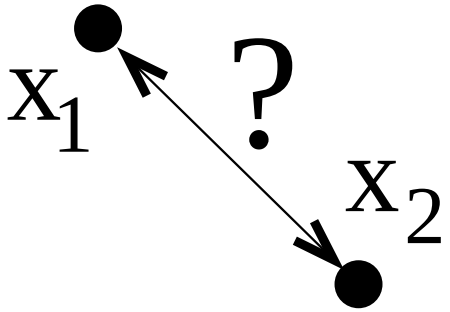
- Instance based classifiers are simple yet surprisingly effective.
- Decision surfaces are **nonlinear**.
- Non(semi)-parametric, so **high capacity without training**.
- Quality of predictions automatically improves with more data. (Asymptotically optimal.)
- Only a **few parameters to tune**, usually by simple optimization.



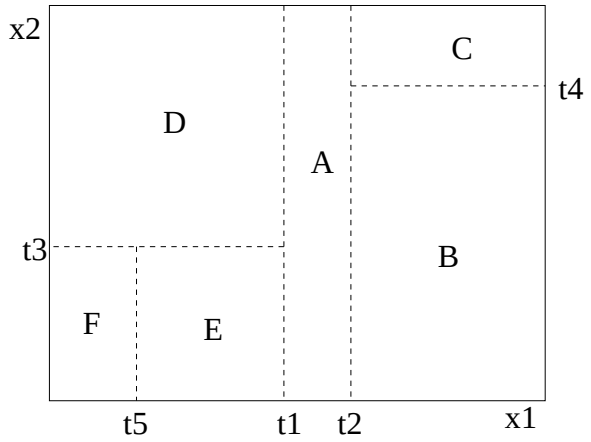


Problems with Semi-Parametric Classification

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



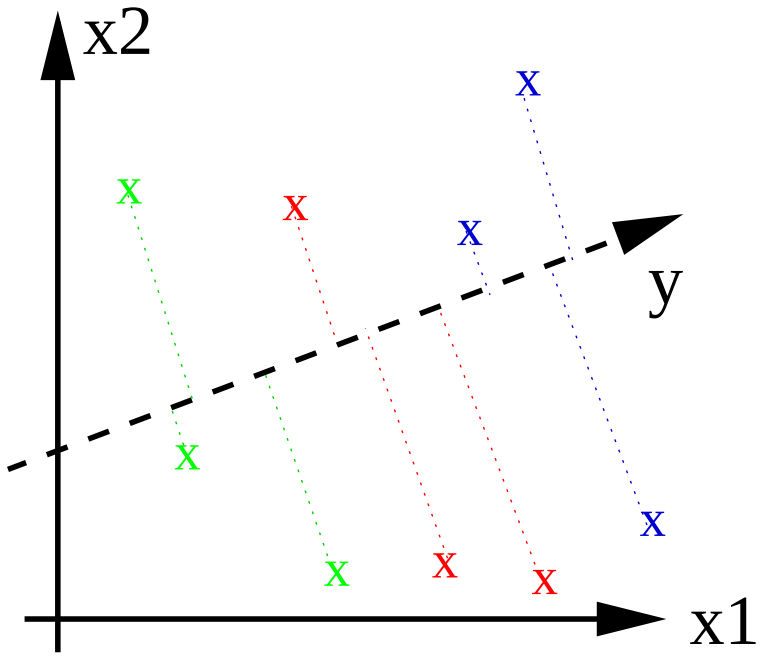
- **Computational cost**: must store and search through entire training set at test time.
  (In low dimensional input spaces, this can be alleviated by thinning training set and building fancy data structures like KD-trees.)



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

Link with Feature Extraction/Data Transformation

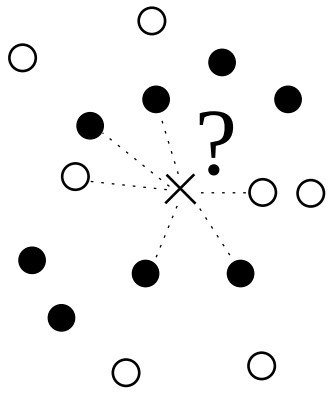
- We can either think of our goal as learning the metric $\mathbf{Q} = \mathbf{A}^\top \mathbf{A}$ or as learning a linear transformation/feature set $\mathbf{y} = \mathbf{A}\mathbf{x}$.



- In general, if we are learning a set of features $\mathbf{y}_k(\mathbf{x})$ we can always fix a simple distance measure $D[\cdot, \cdot]$ (e.g. Euclidean) and use this to define our distance measure d on the original input space x via $d[\mathbf{x}_1, \mathbf{x}_2] = D[\mathbf{y}(\mathbf{x}_1), \mathbf{y}(\mathbf{x}_2)]$.

Cross Validation for Metric Learning?

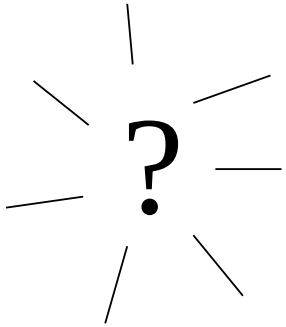
- Consider K-NN classification as an example.
- Q: **What is the right distance metric for KNN classification?**
  A: **The one that optimizes test error!**



- 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...?

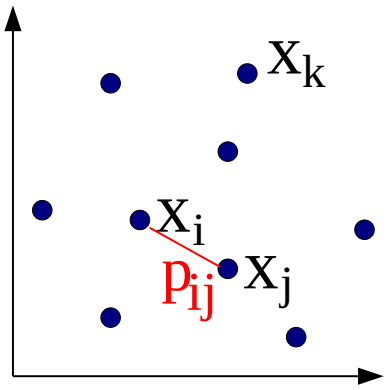
Cross-Validation Performance is Hard to Optimize

- 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 $\Leftrightarrow$ Linear Transforms

- 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 Transformation A is also learned

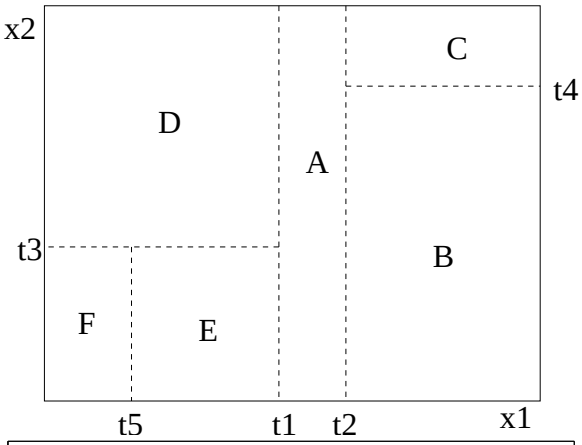
- Notice that not only the relative directions of the rows of **A** but also the **overall scale** of **A** is being learned.
- This means that we are effectively learning a real-valued estimate of the optimal neighbourhood size.
- Estimate = **average effective perplexity** of distributions p_{ij}:

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

- 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**.



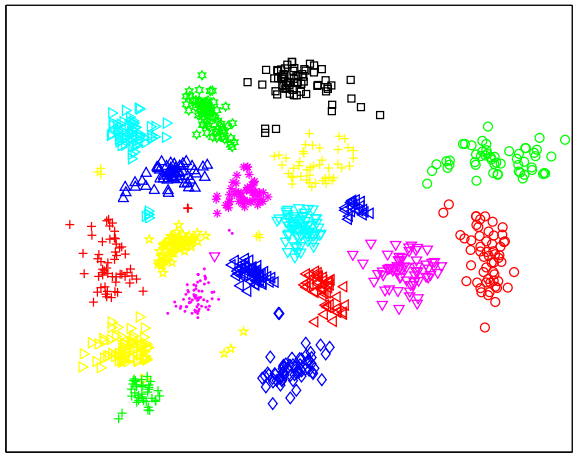
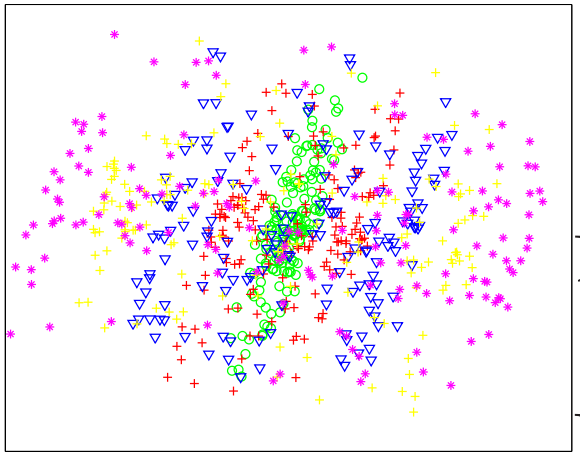
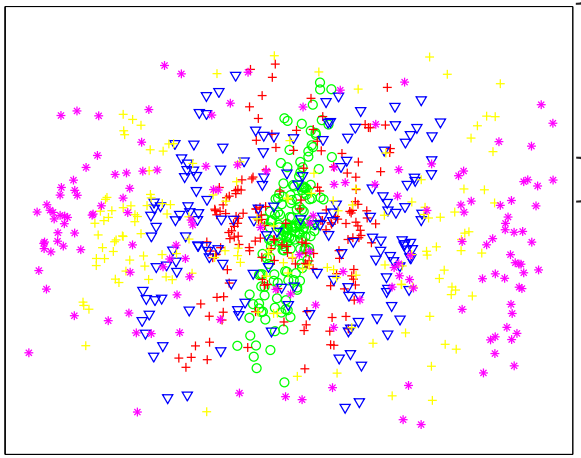


Illustration: Concentric Rings

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

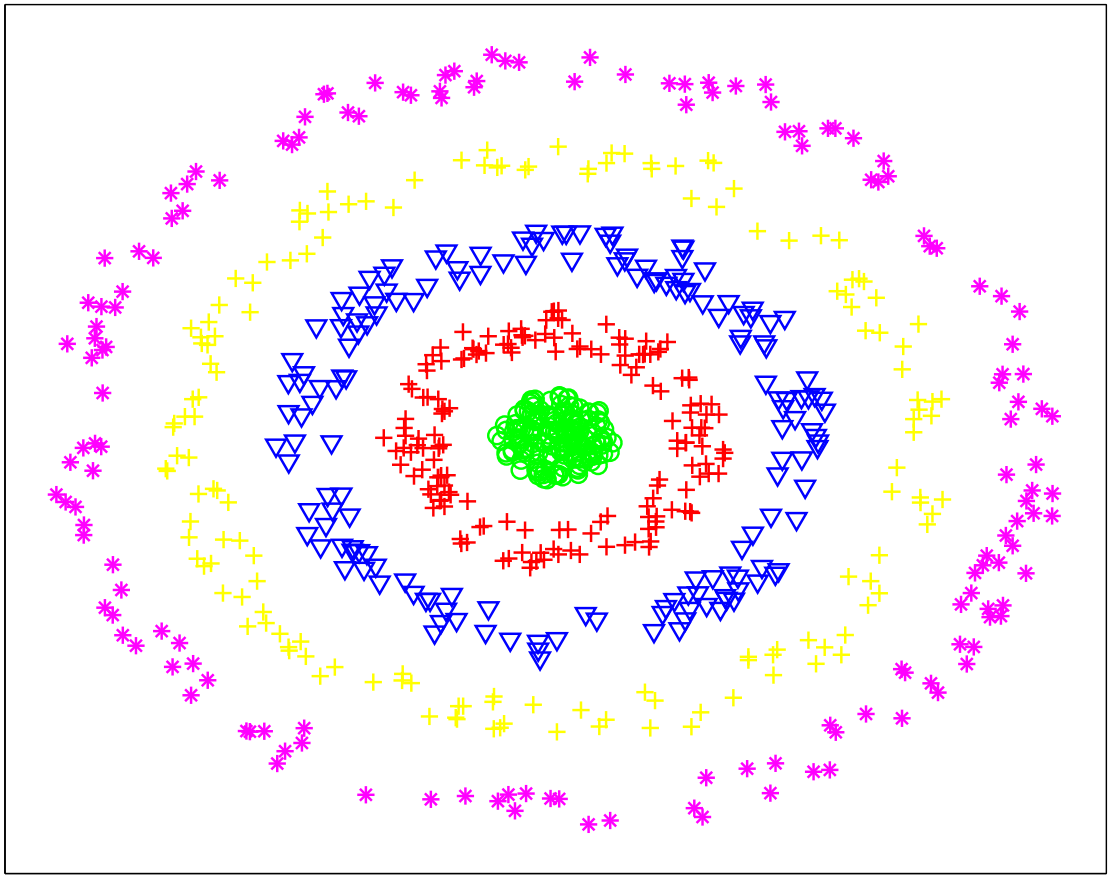


LDA



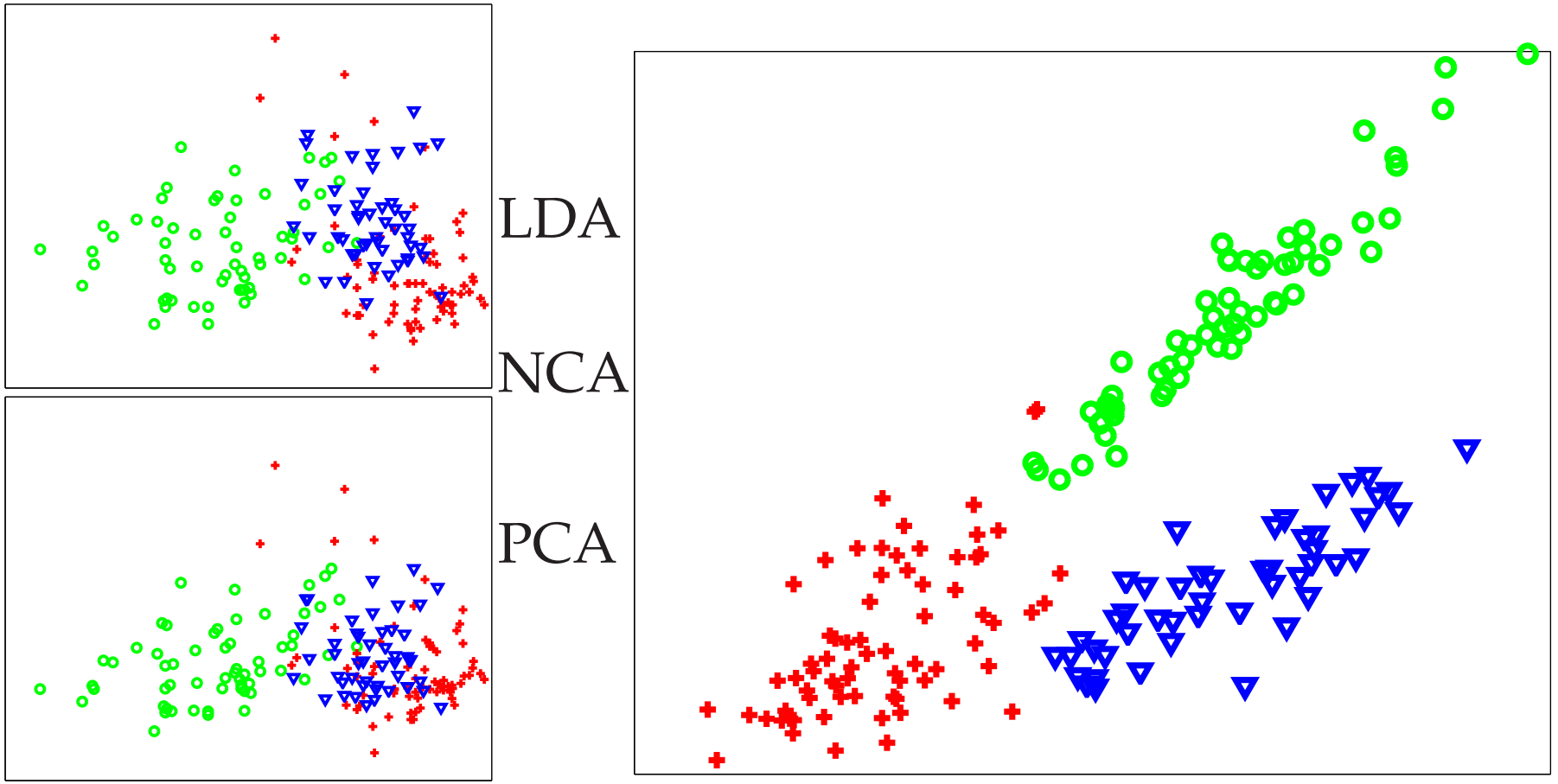
PCA

NCA



Toy Data: UCI Wine

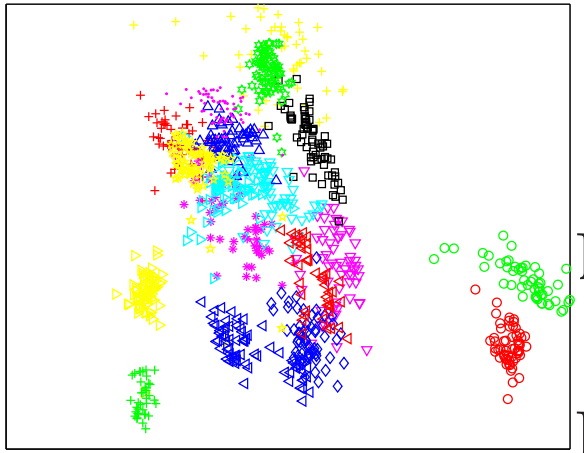
- UCI "Wine", N=178, D=13, 3 classes. Half train, half test.
- Test errors using d=D=13, and K chosen by LOO:
  Euclidean=30%;Whiten=25%;NCA=7%



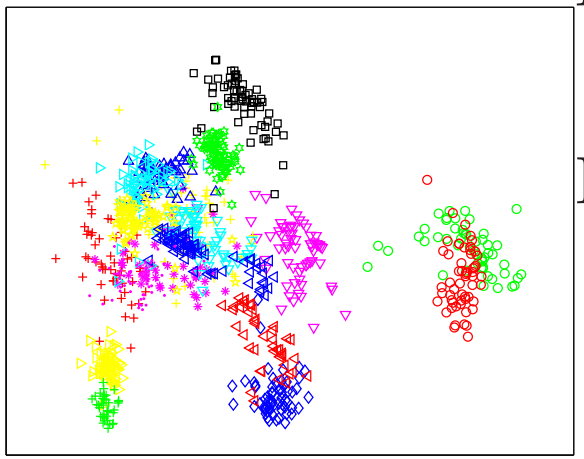
- Test errors using KNN in 2D: LDA=28%; PCA=31%; NCA=5%

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%

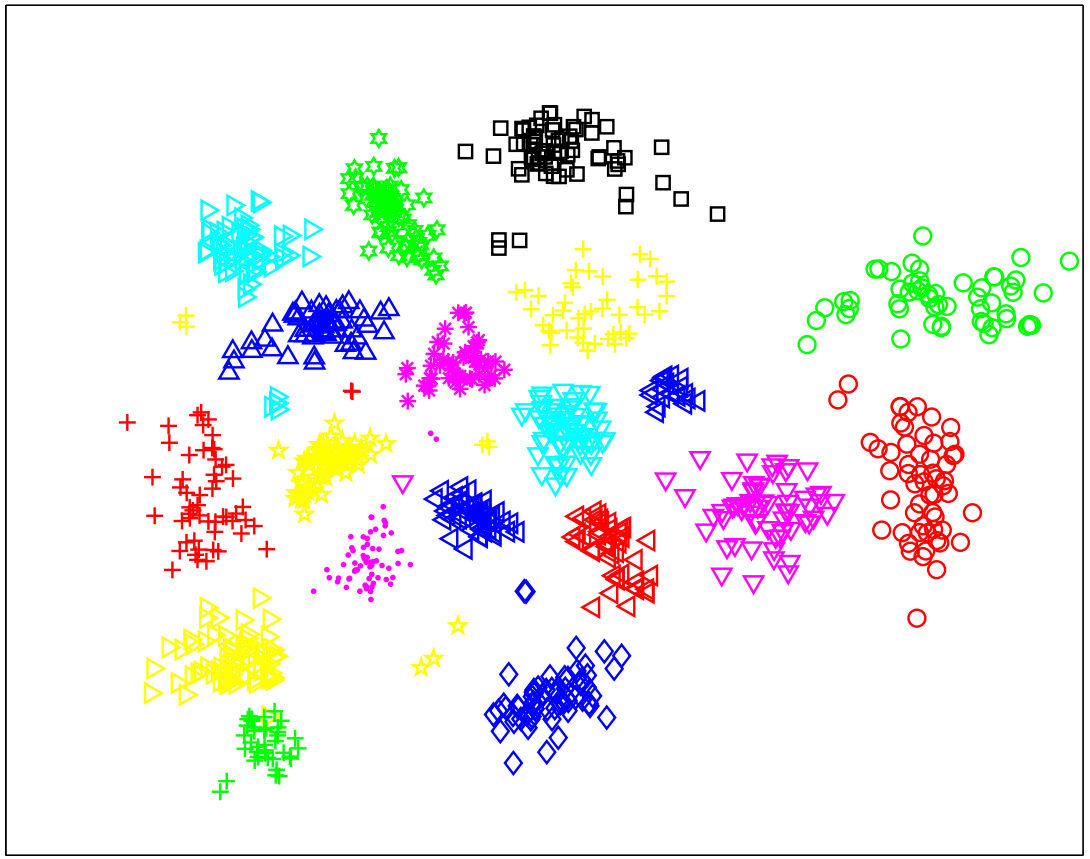


LDA



NCA

PCA



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

Related Objective Functions

- The (log of) the original NCA objective maximizes the **expected number of correct labels**:

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

- We could also maximize the expected $\log$ probability of correct classification, which is the **chance of a perfect labelling**:

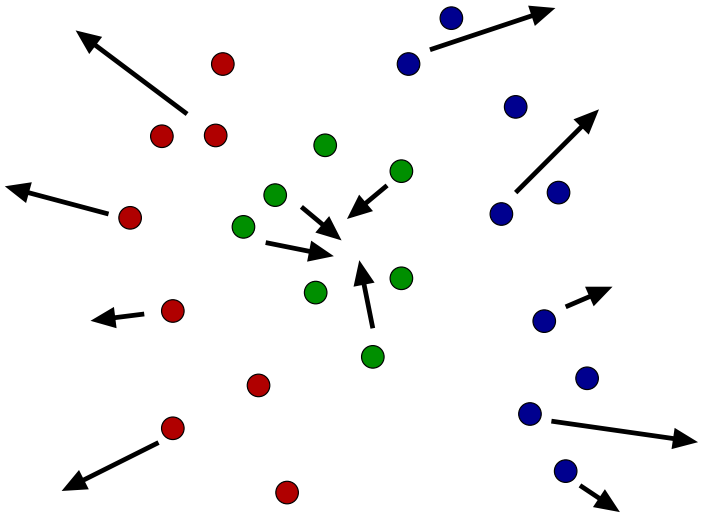
$$\phi = \sum_i \log p_i^+ = \sum_i \log \sum_{j \in C_i} \frac{e^{-d_{ij}}}{\sum_{k \neq i} e^{-d_{ik}}}$$

- Results are very similar with this cost for clean data.
  But for data with **outliers**, the original cost is more robust.
- There is one more variation which is interesting:

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

Geometric Intuition – Class Collapsing

- A good metric is one under which points of the **same class** look **close** to each other and look **far** from all points in **different classes**.



- Under this intuition, a "perfect" metric would **collapse** all points in the same class to a **single point** and simultaneously push all points of other classes **infinitely far away**.
- Of course, this assumes each class distribution is **unimodal**, but doesn't assume all classes have the shape.
- To convert this geometric intuition into profit, we will enlist the help of probability theory to obtain a convex numerical optimization problem.

Maximally Collapsing Metrics (MCM)

- 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:

$$p^{\text{ideal}}(j|i) = \begin{cases} \text{const} & \text{if} \quad C_i = C_j \\ 0 & \text{if} \quad C_i \neq C_j \end{cases}$$

- 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 $\mathbf{A}$:

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

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

MCM Learning is a Convex Optimization Problem

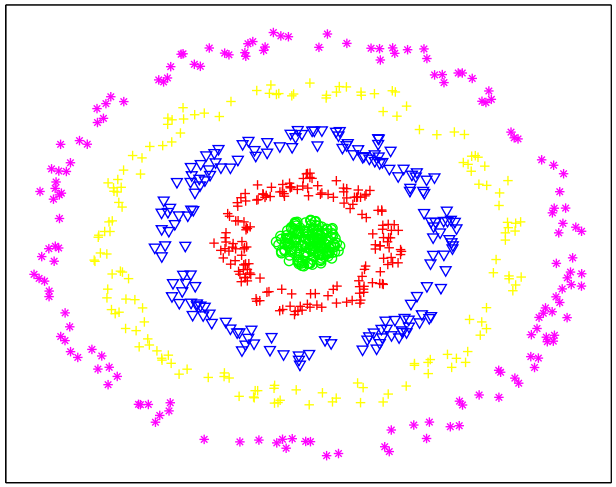
- The set of symmetric positive definite matrices is a **convex set**.
- As well, the KL objective above is **convex in the metric** $\mathbf{Q}$! (But not in the transformation $\mathbf{A}$ itself.)

$$\mathrm{KL} = \mathrm{H}(p^*) - \sum_{i,j:y(j)=y(i)} \log p(j|i) = \mathrm{H}(p^*) + \sum_{i,j:y(j)=y(i)} d_{ij}^{\mathbf{Q}} + \sum_i \log \sum_{k \neq i} e^{-d_{ik}}$$

- Consequence: there is a single, well-defined, globally optimal **"maximally collapsing metric"** $\mathbf{Q}^*$ for any (non-trivially) labelled dataset (in general position). (Assuming we add a small amount of regularization for the norm or trace of $\mathbf{Q}$.)
- We can find it by various methods (as far as I know the optimization problem above does not have a common name).
- For the experiments I'll show, we just used gradient descent followed by projection back onto the PSD cone.

Relationship to Fisher's Discriminant (LDA)

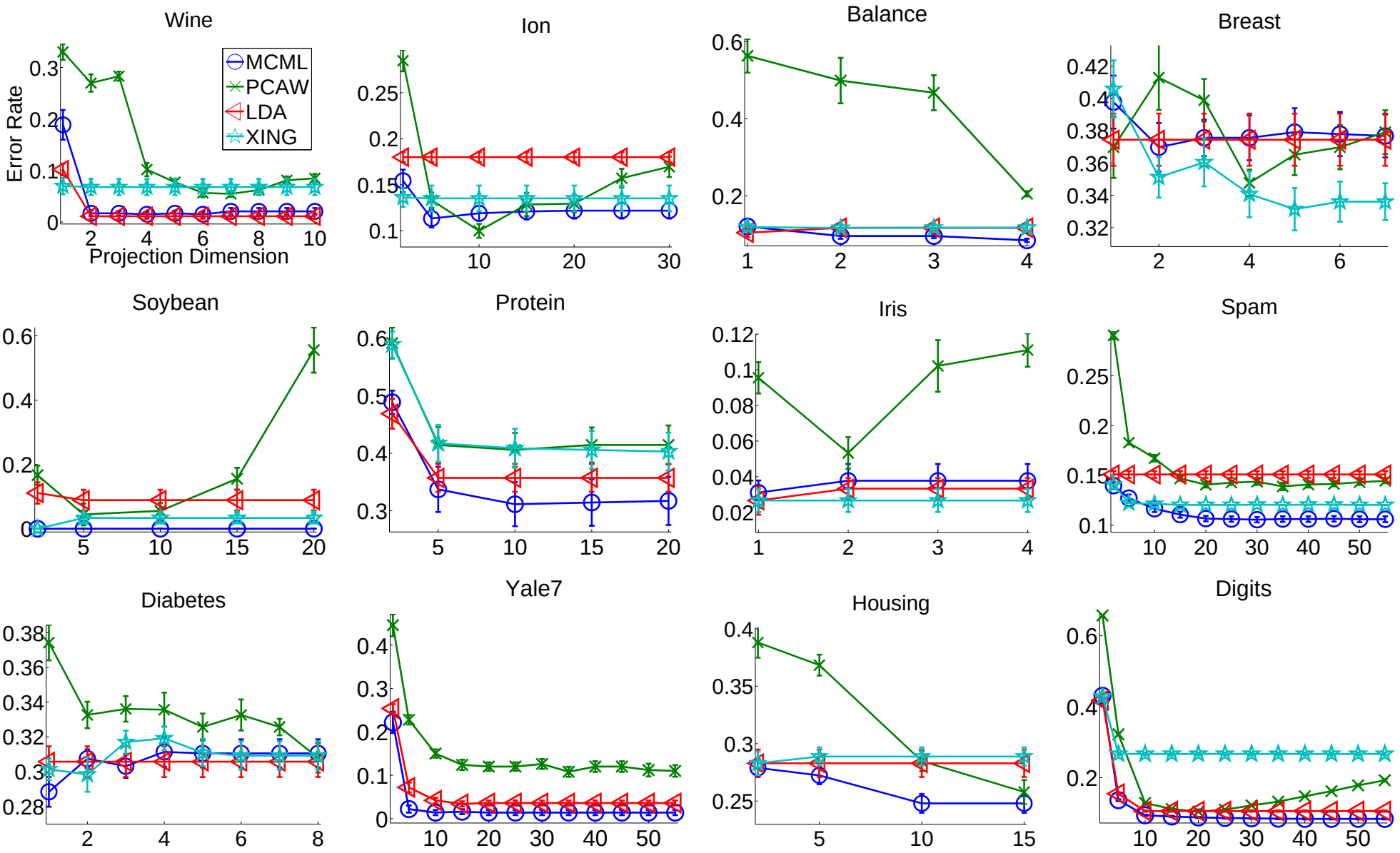
- MCM is similar to Fisher's Discriminant (LDA) in that it tries to **minimize within class distances** (variance) and **maximize between class distances**.
- But LDA is a purely second order ("Gaussian") method; it depends only on the mean of each class and on covariance of points **within the same class**.
- MCM is a **generalization of LDA** that makes only a much weaker assumption, namely that each class is distributed as a **unimodal** blob, which can be separated (under the right metric) from the other class blobs.
- But (like LDA) MCM will fail, e.g. on concentric rings. However, the probabilistic setup can be extended to a mixture model with hidden variables giving an EM-like procedure to deal with this case.



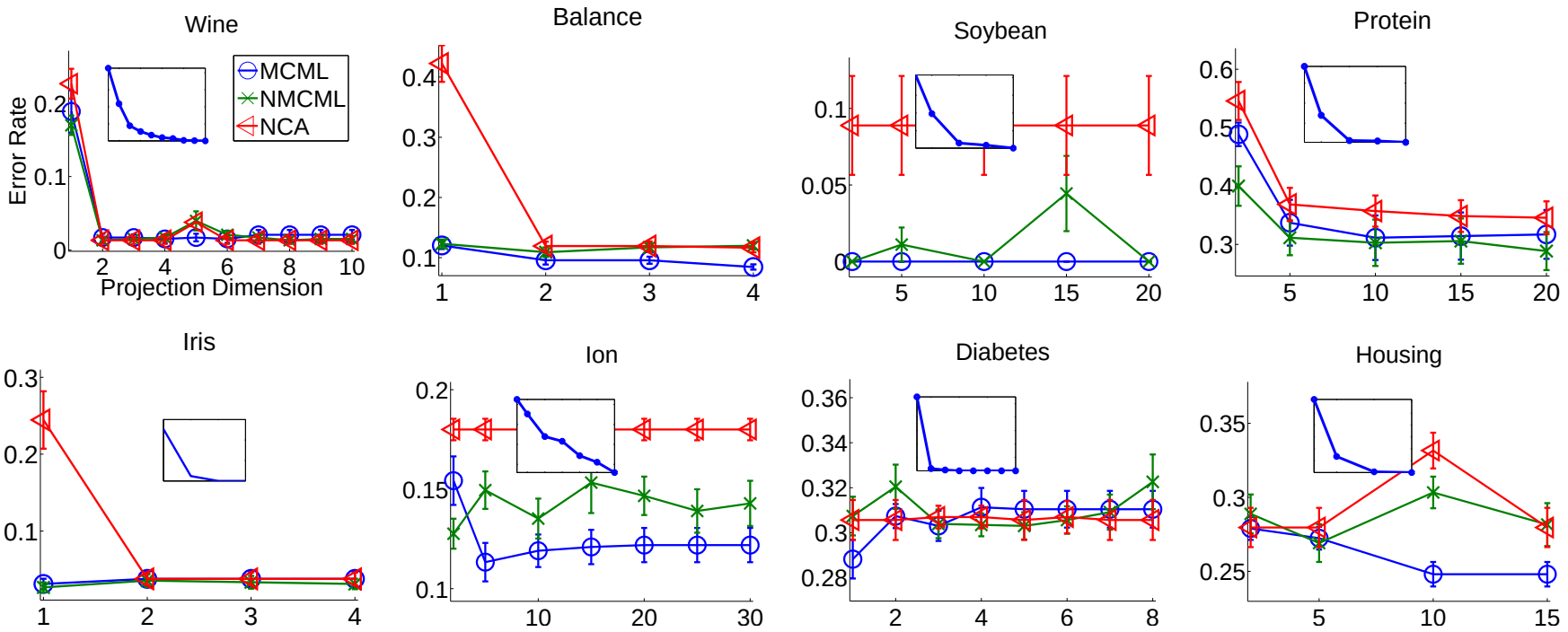
Learning Low-Rank Collapsing Metrics

- Unfortunately, the set of matrices of rank at most k is **not** a convex set. (Think of the average of two vector outer products.)
- Hence, the optimization problem no longer has the strong guarantee of global optimality.
- There are two ways to proceed:
  1. Find the optimal full-rank metric, and keep only its k largest eigenvalues, zeroing out all the others. This procedure is well defined, and has no local minima. It will be close to optimal if the full rank metric has a **rapidly decaying eigenspectrum**.
  2. Explicitly parameterize the set of low rank matrices (e.g. using the LU decomposition of $\mathbf{Q}$) and optimize the KL objective using traditional **local search methods**.
- We have experimented with both of these, and found them both to give good results since often the exact metric has many small singular values.

Results for 1-NN Classification & Eigen-Truncation



Results for 1-NN Classification & Local-Search

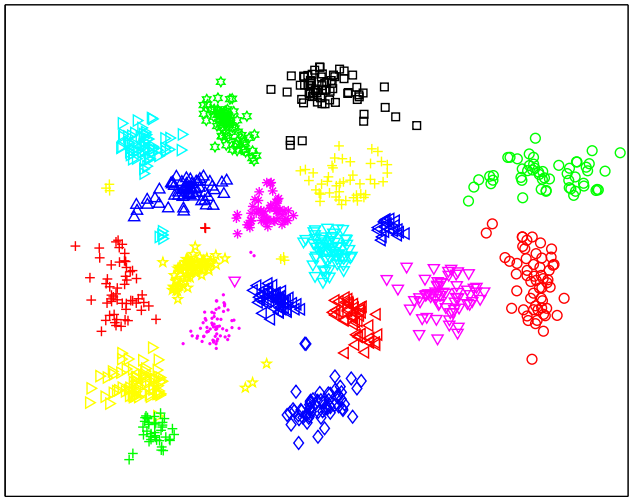


10 random splits, 70% train, 30% test. Initialized with LDA.

For most of these datasets each class is a single connected blob, we just need to learn how to warp the space to separate the blobs nicely.

Conclusions: NCA and MCM Learning

- In the absence of strong prior knowledge of **how to pick your classification distance metric**, learning seems like a good idea.
- Learning **low rank metrics** gives a natural way to reduce memory & computation while still doing well on 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)
- MCM assumes unimodality for each class, but gives a convex objective



- **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.

Kernel Maximally Collapsing Metrics

- Kernelization: by adding the term Trace[$\mathbf{Q}$] to the objective, we can get a function which depends only on dot products of transformed input vectors and hence learn a metric function (of size $N \times N$) for use with any Mercer kernel.

MCM Dual is Constrained Entropy Maximization

- Our convex optimization problem has an equivalent **convex dual**, which in this case is **constrained entropy maximization**:

$$\max_{p(j|i)\ i,j=1\ldots n} \sum_i H[p(j|i)] \qquad s.t. \quad \sum_j p(j|i) = 1 \quad \forall i \quad \text{and}$$

$$\sum_i E_{p^*(j|i)}[v_{ji} v_{ji}^T] - \sum_i E_{p(j|i)}[v_{ji} v_{ji}^T] \succeq 0 \quad ; \quad v_{ji} = x_j - x_i$$

- Each point wants to hedge its bets over neighbours as much as possible under $\mathbf{Q}$ without accruing more variance than it would accrue if it used all the same-class points as neighbours.
- The matrices $E_{p(j|i)}[v_{ji} v_{ji}^T]$ play the role of the within-class and between-class covariances, but are computed at each point not just once per class.
- Computationally, dual is much less attractive since it scales as the square of the number of datapoints.

What does the metric "look like"?