Elemental Set Methods

David Banks
Duke University

1. Introduction

Data mining deals with complex, high-dimensional data. This means that datasets often combine different kinds of structure. For example:

- There might be several different subgroups within the data, each described by a different linear relationship.
- Some clusters might be tightly grouped with respect to one subset of variables, whereas other clusters are grouped according to different variables.
- A few outliers can distort otherwise simple structure; adaptive methods for ignoring them are wanted.

What one wants to do if sequentially extract the structures, one at a time.

To be concrete, suppose one has a large, complex dataset that two kinds of linear structures and pure noise, e.g.:

- 40% of the data follow $Y = \alpha_0 + \sum_{i=1}^{p} \alpha_i X_i + \epsilon$
- 30% of the data follow $Y = \beta_0 + \sum_{i=1}^{p} \beta_i X_i + \epsilon$
- 30% of the data are noise.

One can imagine that the two linear structures correspond to some unmeasured categorical covariate, and the pure noise represents a third category of cases.

What can one do to analyze cases like this? One should assume that the data miner has little prior knowledge of the kinds of structure that might be present.

The standard approach in linear regression is to use S-estimators, which look for the thinnest strip (think of a transparent ruler) which covers some prespecified (but larger than 50%) fraction of the data.

This strategy breaks down in high dimensions, or when the structures of interest contain less than 50% of the sample, or when fitting complex nonlinear models.

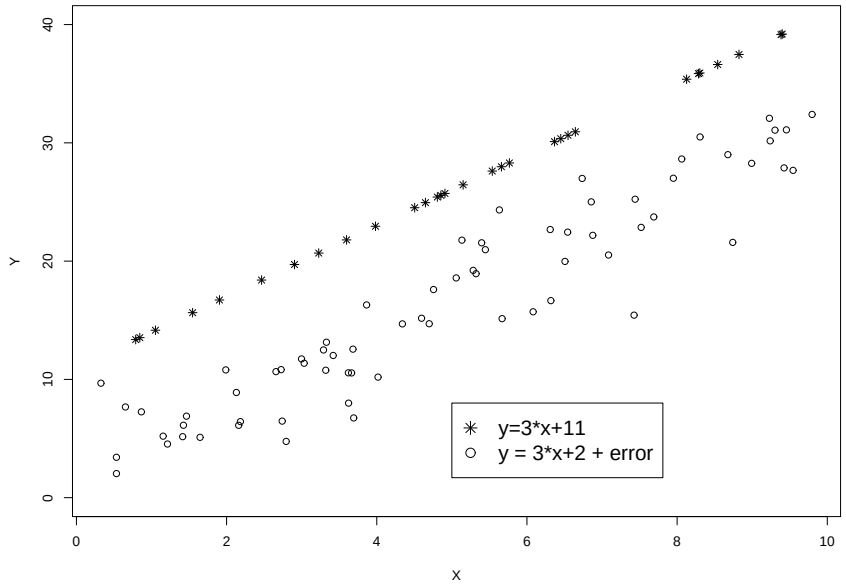
One wants a solution strategy that applies to more general cases, including:

- linear and non-linear regression in high dimensions,
- multidimensional scaling,
- cluster analysis.

The method described here can be extended to other cases as well.

2. Hidden Structure in Regression

Consider the graph below, for a simple regression problem. It is clear that the data come from two different models, and that any naive attempt at regression will miss both of the interesting structures and find some kind of average solution.



For linear regression, assume the observations are $\{Y_i, \mathbf{X}_i\}$ for $i = 1, \ldots n$ and that Q percent of these follow the model

$$Y_i = \beta_0 + \beta_1 X_{i1} + \ldots + \beta_p X_{ip} + \epsilon_i \quad \text{where} \quad \epsilon_i \sim N(0, \sigma)$$

where Q, $\boldsymbol{\beta}$, and σ are unknown. The rest of the data have no relationship between Y_i and $\boldsymbol{X}_i$.

One can refer to the $Q\%$ of the data as "good" and the rest as "bad".

Simple Idea:

Start small, with a subsample of only **good** observations

$\Rightarrow$ add only **good** observations

$\Rightarrow$ end with a large subsample of **good** observations.

General procedure:

1. Strategically choose an initial set of d starting subsamples S_j, each of size m.
2. Grow the subsamples by adding consistent data.
3. Select the largest subsample.

The method for choosing the starting subsamples and growing them efficiently is important in practice.

One starts with a guess about Q, the fraction of good data. In general, this is unknown, so one might pick a value that is reasonable given

- domain knowledge about the data collection
- the point at which a fraction is so small that there is little scientific interest.

From the full dataset $\{Y_i, \mathbf{X}_i\}$ one selects, without replacement, d subsamples S_j of size m.

One needs to choose d and m to ensure that at least one of the starting subsamples S_j has a very high probability C of consisting entirely of good data (i.e., data that come from the same unknown structure).

Preset a probability C that determines the chance that the algorithm will work.

The value m, which is the size of the starting-point random subsamples, should be the smallest possible value that allows one to calculate a goodness-of-fit measure. In the case of multiple linear regression, that value is $p + 2$, and a natural goodness-of-fit measure is R^2.

One solves the following equation for d:

$$C = \mathbb{P}[\text{ at least one of } S_1, \ldots, S_d \text{ is all good }] = 1 - (1 - Q^{p+2})^d.$$

Example: $Q = .8$, $c = .95$, $m = 3$ (for simple linear regression, with $p = 1$):

$$.95 = 1 - [1 - (.8)^{p+2}]^d \quad \rightarrow \quad d = 5$$

Given the d starting-point subsamples S_j, one grows each one of them by adding observations that do not appreciably lower the goodness-of-fit statistic (R^2).

Conceptually, for a particular S_j, one could cycle through all of the observations, and on each cycle augment S_j by adding the observation that provided the largest value of R^2. This cycling would continue until no observation can be added to S_j without substantially decreasing the R^2.

One does this for all of the subsamples S_j. At the end of the process, each augmented S_j would have size m_j and goodness-of-fit R_j^2. The augmented subsample that achieves a large value of m_j and a large value of R_j^2 is the one that captures the most important structure in the data.

Then one can remove the data in S_j and iterate to find the next-most important structure in the dataset.

In practice, the conceptual algorithm which adds one observation per cycle is expensive when the dataset is large or when one is fitting a complex model (e.g., doing MARS fits rather than multiple regression). For this reason, we use a two-step procedure to add observations.

Fast Search

- Sequentially sweep through all observations not in S_i.
- If the obsevation improves the fitness measure (or perhaps only lowers it by a very small amount), then
  $\rightarrow$ add observation to S_j
  $\rightarrow$ set $m_j = m_j + 1$.
  If m_j is large, say $Qn/2$,then implement slow search.

Slow Search

- Add the observation that improves the FM the most or decreases the fitness measure by not more than some prechosen threshold.
- Repeat until no observation can be added.

The analyst may pick a threshold that seems appropriately small and a fraction of n that seems appropriately large. These choices determine the runtime of the algorithm and should reflect practical constaints.

The fast search is greedy, and the order of observations in the cycling matters. The slow search is less greedy; order does not matter, but it adds myopically. The fast search can add many observations per cycle through the data, but the slow search always adds exactly one.

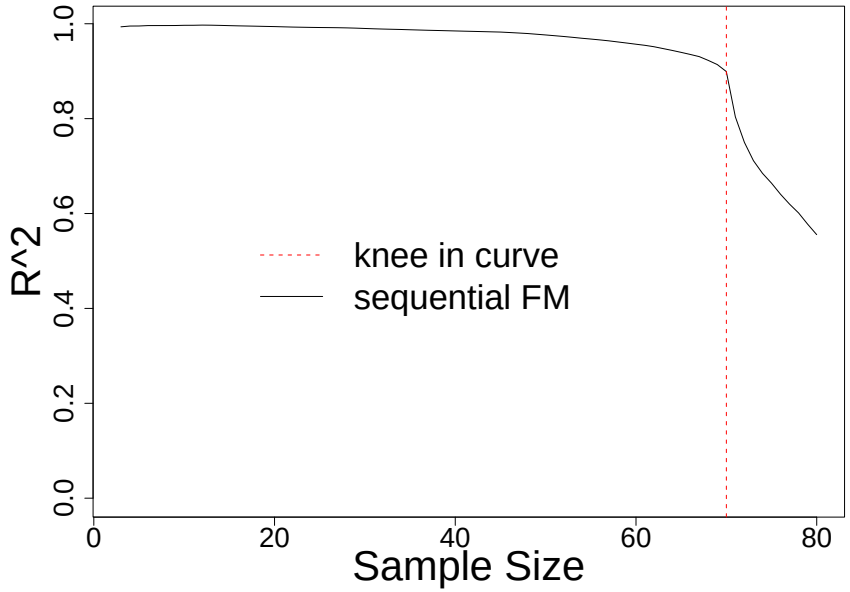
If speed is truly important, then there are other ways to accelerate the algorithm. A standard strategy would be to increase the number of starting-point subsamples and combine those that provide similar models and fits as they grow.

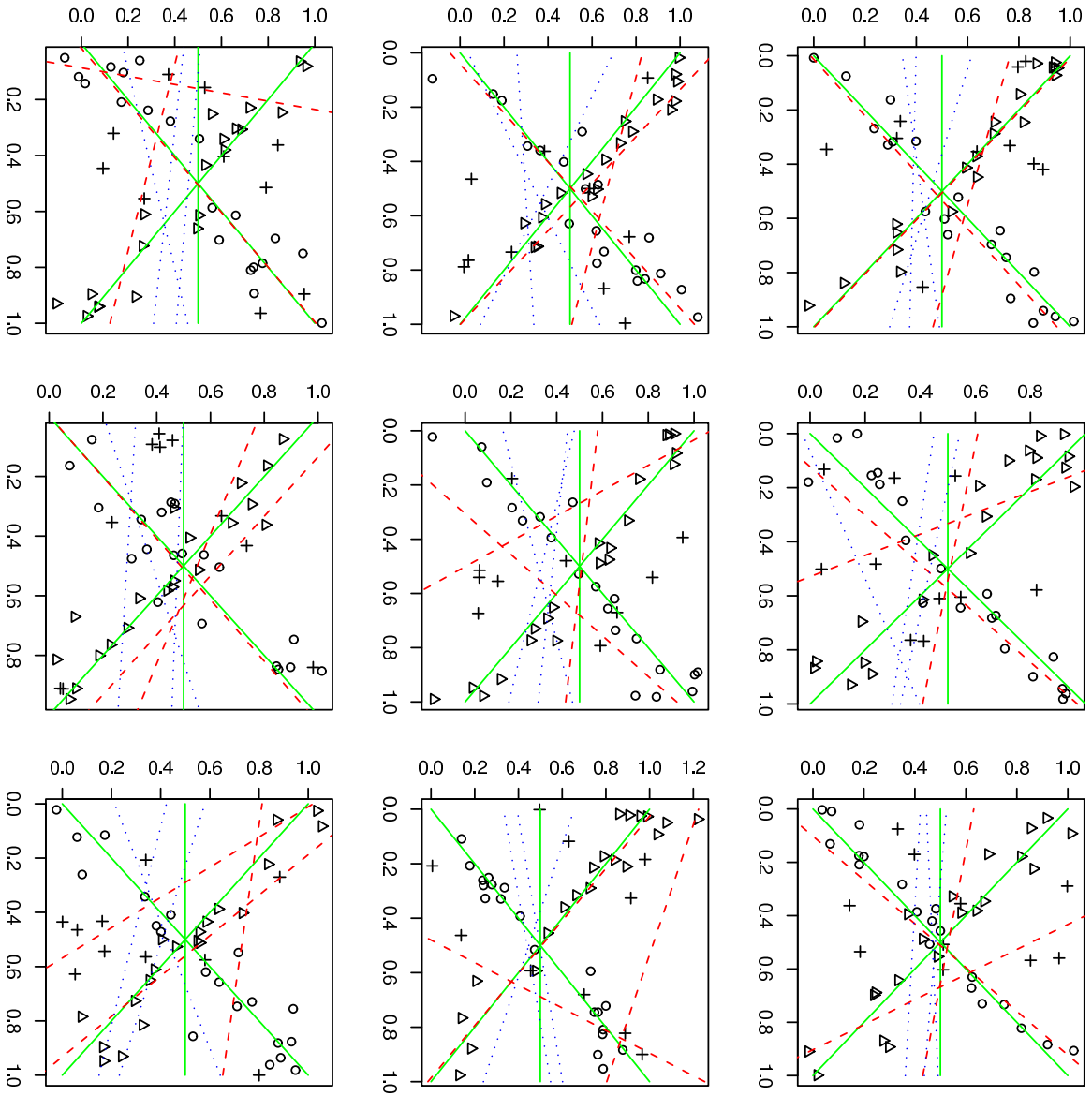
The main concern is not to enumerate all $\begin{pmatrix} n \\ \lceil Qn/100 \rceil \end{pmatrix}$ possible subsamples.

Note that:

1. One does not need to terminate the search at some preset fraction; one can just grow until the goodness-of-fit measure deteriorates too much.
2. The goodness-of-fit measure should not depend upon the sample size. For SLR this is easy, since R^2 is just the proportion of variation in Y explained by X. For larger p, if one is doing stepwise regression to select variables, then one wants to use an AIC or Mallows' C_p statistic to adjust the tradeoff in fit between the number of variables and the sample size.
3. Other measures of fit are appropriate for nonparametric regression, such as cross-validated within-subsample squared error. But this adds to the computational burden.
4. One can and should monitor the fit as new observations are added. When one starts to add bad data, this is quickly visible in a plot—there is a clear "slippery-slope" effect.

To see how the slippery-slope occurs, and the value of monitoring fit as a function of order of selection, consider the plot below. This plot is based on using R^2 for fitness with the double-line data shown previously. The total sample size is 80, and 70 observations were generated with small noise from a line; the knee in the curve clearly shows when one should stop adding observations.





3. Hidden Structure in Multidimensional Scaling

Multidimensional scaling (MDS) starts with a proximity matrix that gives approximate distances between all pairs in a set of objects. These distances are often close to a true metric.

The purpose of MDS is to find a low-dimensional plot of the objects such that the inter-object distances are as close as possible to the values given in the proximity matrix. That representation automatically puts similar objects near each other. This is done in terms of a least squares fit to the values in the proximity matrix, by minimizing the stress function:

$$\textbf{Stress}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n) = \left[\sum_{i \neq i'} (d_{ii'} - \|\boldsymbol{z}_i - \boldsymbol{z}_{i'}\|) \right]^{1/2}$$

where $\boldsymbol{z}_i$ is the location assigned to pseudo-object i in the low-dimensional space and $d_{ii'}$ is the entry in proximity matrix.

The classic example is to take the entries in the proximity matrix to be the drive-time between pairs of cities. This is not a perfect metric, since roads curve, but it is approximately correct. MDS finds a plot in which the relative position of the cities looks like it would on a map (except the map can be in any orientation; north and south are not relevant).

MDS solutions are extremely susceptible to bad data. For example, if one had a flatt tire while driving from Baltimore to DC, this would create a seemingly large distace. The MDS algorithm would distort the entire map in an effort to put Baltimore far from DC and still respect other inter-city drive times.

A very small proportion of outliers, or objects that do not fit well in a low-dimensional representation, can completely wreck the interpretability of an MDS plot. In many applications, such as text retrieval, this is a serious problem.

3.1 MDS Example

To test cherry-picking for MDS, consider the latitudes and longitudes of 99 eastern U.S. cities. The Euclidean distances between these cities gave the proximity matrix; the only stress in the MDS map is due to the curvature of the earth.

Perturb the proximity matrix by inflating a random proportion $1 - Q$ of the entries:

Bad Data	Distortion (%)	Stress
2	150	1.028
	500	2.394
10	150	1.791
	500	28.196
30	150	3.345
	500	9.351

To make things more interesting, we use not the traditional MDS using the stress measure defined previously, but rather Kruskal-Shephard non-metric scaling, in which one finds $\{\boldsymbol{z}_i\}$ to minimize

$$\textbf{Stress}_{KS}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n) = \frac{\sum_{i \neq i'} [\theta((\|\boldsymbol{z}_i - \boldsymbol{z}_{i'}\|) - d_{ii'}]^2}{\sum_{i \neq i'} d_{ii'}^2}$$

where $\theta(\cdot)$ is an arbitrary increasing function fit during the minimization. The result is invariant to monotonic transformations of the data, which is why it is nonparametric.

This minimization uses an alternating algorithm that first fixes $\theta(\cdot)$ and finds the $\{\boldsymbol{z}_i\}$, and then fixes the $\{\boldsymbol{z}_i\}$ and uses isotonic regression to find $\theta(\cdot)$. This shows that the algorithm can be used in complex fits.

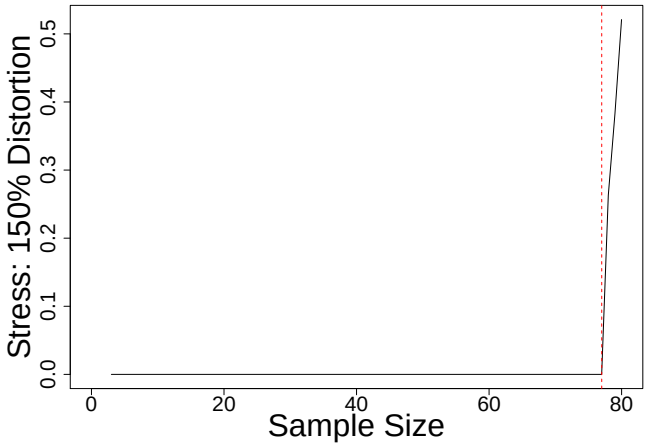
Our goal is to cherry-pick the largest subset of cities whose intercity distances can be represented with little stress.

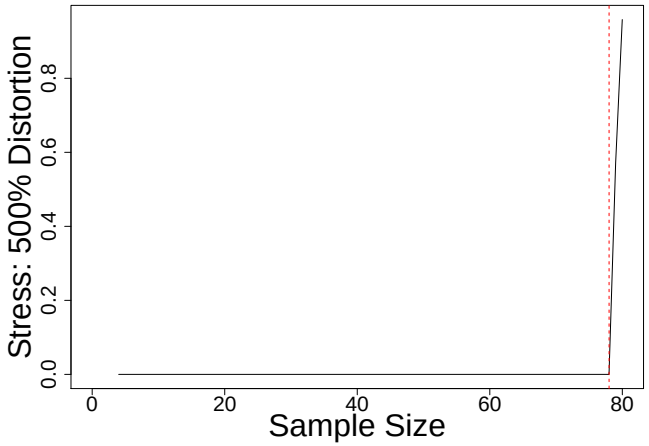
In MDS, the size m of the initial subsamples is 4 (since three points are always coplanar). We took $C = .99$ as the prespecified chance of getting at least one good subsample, and for $Q = .8$ this implies we need 9 starting samples. The results are in the table.

True $1-Q$ (%)	Distance Distortion (%)	Original Stress	n^a	n^*	Final Stress
2	150	1.028	80	80	4.78e-12
	500	2.394	80	80	4.84e-12
10	150	1.791	80	80	4.86e-12
	500	28.196	80	80	4.81e-12
30	150	3.345	80	77	4.86e-12
	500	9.351	80	78	4.78e-12

- Note: The stress of the undistorted dataset was 8.42×10^{-12}.

As before, one should inspect order-of-entry plots that display the stress against the cities chosen for inclusion. The following two plots are typical, and show the knee in the curve that occurs when one begins to add bad cities.



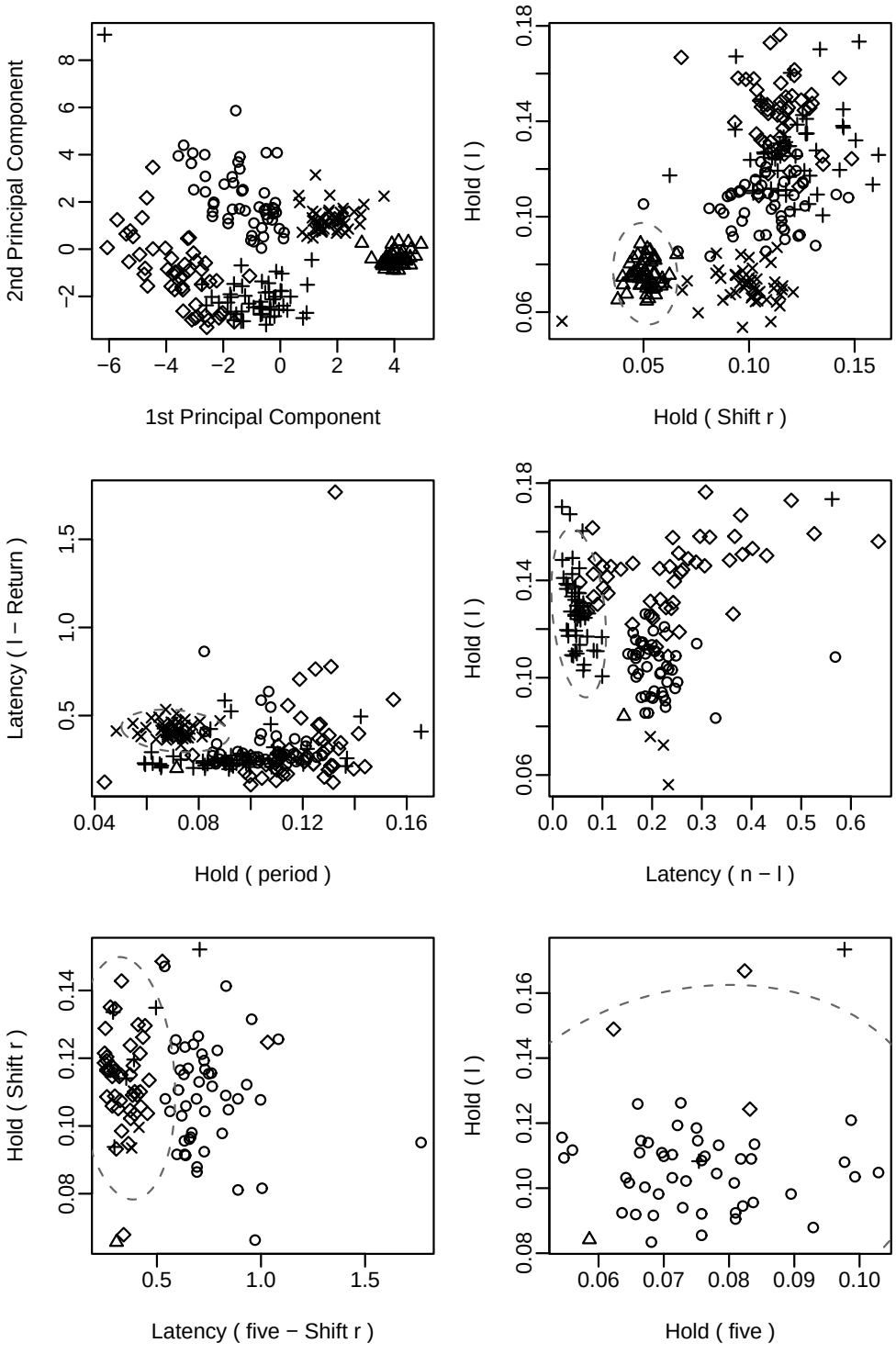


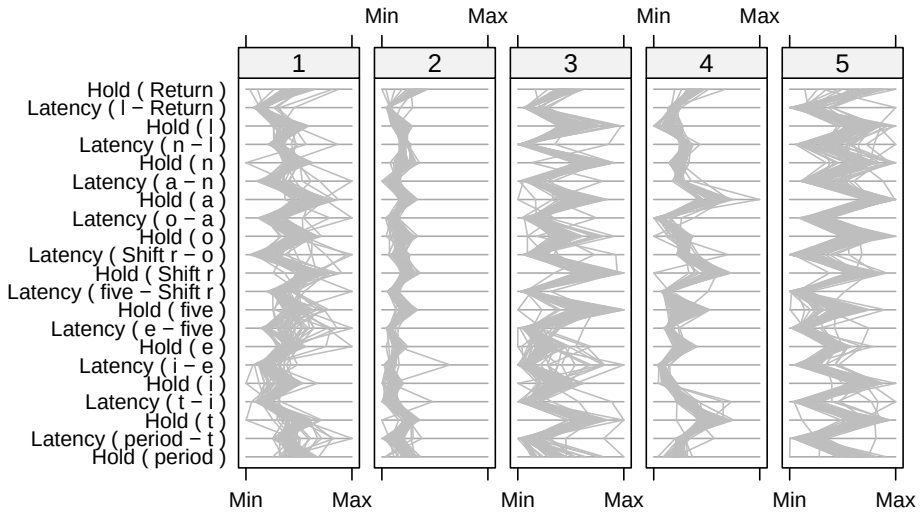
4. Other Applications

Cherry-picking seems to work for regression, and the extension to non-linear and nonparametric regression is straightforward. It also works for several different kinds of MDS.

Another natural application is cluster analysis. If one lets each cluster be defined by a separate covariance matrix, then one needs starting samples that contain just enough data to estimate the Mahalanobis metric. This is $p(p-1)/2$ which quickly gets large and implies that the number of starting samples also needs to be large.

One strategy for addressing this problem is to define the metric in terms of only two or three of the explantory variables, selected to give the tightest cluster structure.


2nd Principal Component
1st Principal Component
Hold (l)
Hold (Shift r)
Latency (l − Return)
Hold (period)
Hold (l)
Latency (n − l)
Hold (Shift r)
Latency (five − Shift r)
Hold (l)
Hold (five)


Min
Max
Min
Max
1
2
3
4
5
Hold (Return)
Latency (l − Return)
Hold (l)
Latency (n − l)
Hold (n)
Latency (a − n)
Hold (a)
Latency (o − a)
Hold (o)
Latency (Shift r − o)
Hold (Shift r)
Latency (five − Shift r)
Hold (five)
Latency (e − five)
Hold (e)
Latency (i − e)
Hold (i)
Latency (t − i)
Hold (t)
Latency (period − t)
Hold (period)
Min
Max
Min
Max
Min
Max

In the context of classification, the problem is similar to regression. For the simple case of linear discriminant analysis, one successively looks for hyperplanes that carve off portions of the dataset that have the same class.

If one uses all the variables, then the starting samples must each be of size $p + 2$. (This defines the plane and picks one side.) One hopes one of the starting samples is nearly all of the same class, and that everything that lies on the same side as it of a discriminant plane is also of the same class.

It doesn't matter how heterogeneous the sample on the other side of the classifying surface might be, provided one side is almost pure.

It is not yet clear how to make the probability calculation, since it depends upon assumptions about the geometry of the training sample.

5. Summary

- We have described a strategy for iterative structure discovery in complex datasets.
- It works in computer-intensive applications, but one needs smart search algorithms; scaling to large datasets is feasible but requires care.
- We can make probabilistic statements about the chance of having a good starting-point subsample, and this almost leads to a probabilistic guarantee on the result, but not quite.
- Simulation shows good performance in many applications.
- The method is fairly straightforward for regression and nonparametric regression, MDS, and cluster analysis.