DATA11002 Introduction to Machine Learning

Kai Puolamäki

27 November 2020

Announcements

- Recap lecture on **2 December**
  - I will ask for requests for topics shortly via Presemo
- Machine Learning Guest Lectures on **4 December** (poster)
  - Andreas Henelius (OP Financial Group): *Data Science in Finance - Machine learning for overdue invoice prediction*
  - Antti Ukkonen (Speechly): *From voice to meaning: Machine learning for spoken language understanding*
  - Discussion
- Term project and E3 DL on **6 December**
- Max. 2 min. video pitch on **9 December** (replaces 9 Dec lecture, which is canceled)
- Selected term project presentations on **11 December**
- Term project final report on **20 December**

Unsupervised learning

Unsupervised learning

- Unsupervised learning:
  - clustering *(previous lecture)*
  - dimensionality reduction *(today)*
  - visualization
- No response variable Y, only the variable X
- Several approaches, we will cover (to some extent):
  - principal component analysis (PCA)
  - multidimensional scaling (MDS)
  - introduction to manifold methods (t-SNE)

Dimensionality reduction

- Data points x_i are often high dimensional (vectors in $\mathbb{R}^p$) or their dimensionality is not known (e.g., DNA sequences or text documents of varying length etc.)
- *Dimensionality reduction* is set of methods which *embed* the data point x_i into low-dimensional representation y_i
  - we denote the original data point by x_i and its "pair" in lower-dimensional embedding by y_i
  - typically $y_i \in \mathbb{R}^k$, where $k \ll p$
- We denote the distance of data points i and j...
  - in original space as $d(i, j)$
  - in embedding as $\hat{d}(i, j)$
- We would like to find embedding such that $d(i, j) \approx \hat{d}(i, j)$
  - ...but some information will inevitably be lost!

Why dimensionality reduction?

- visualization
  - visualization is often part of iterative and interactive process (= one visualization is rarely the final answer)
- to help supervised learning algorithms not to over-fit
  - less attributes, less over-fitting
- in supervised learning, to take advantage of unlabeled data
  - you can reduce the dimensionality by using also unlabeled data and then train your model in fewer dimensions (simple form of semi-supervised learning!)

Example: PCA and Boston data

```
## Loading required package: ggplot2
```

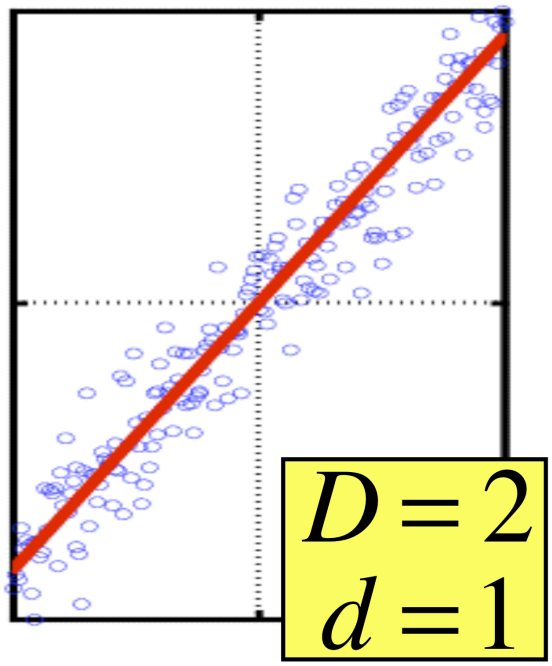
Goodness criteria

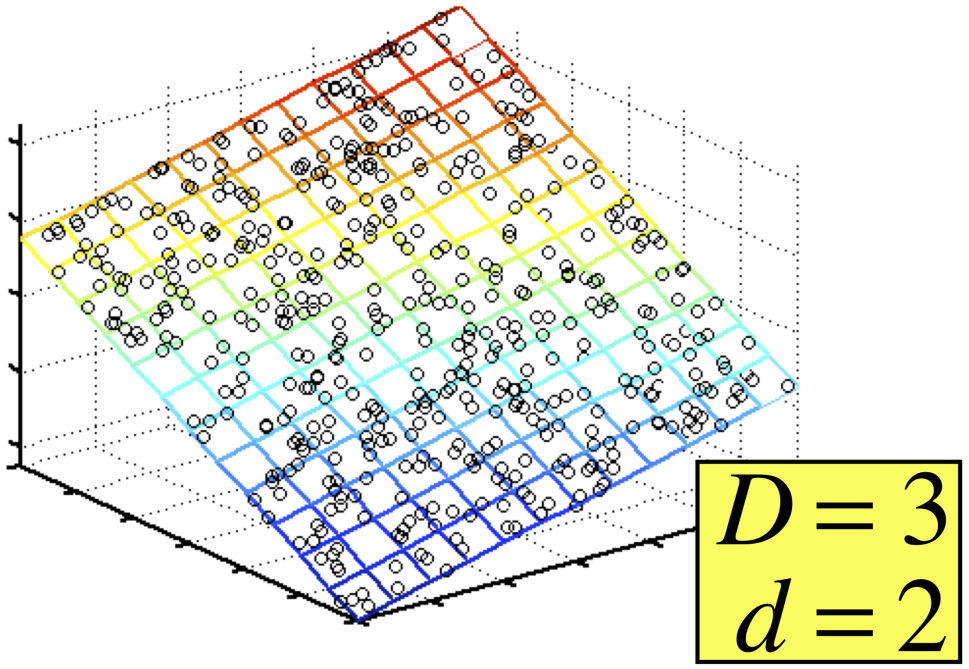
- How do we measure goodness of the embedding?
- Different numeric criteria:
  - **global distances** *(global measure)*: preserve long distances faithfully (e.g., PCA, MDS)
    - e.g., find embedding such that $\sum_{i=1}^{n} (\hat{d}(i,j) - d(i,j))^2$ is **minimised**.
  - **precision** *(local measure)*: if points are nearby in embedding they should be nearby also in the embedding (e.g., ISOMAP, t-SNE)
    - e.g., find embedding such that $\sum_{i=1}^{n} \mathrm{pr}(i)/n$ is **maximized**, where the precision $\mathrm{pr}(i)$ is a fraction of points that are in the neighbourhood of i in the original space of the points that are in the neighbourhood of the point i in the embedding.
  - other measures such as preservation of angles etc. may be important in some contexts.
  - **Validation loss**, if used for model complexity tuning

Global distance preserving methods

- Principal component analysis (PCA)
  - the "default" dimensionality reduction method to use
  - example of a projection pursuit method
- Multidimensional scaling (MDS)
  - non-linear cousin of PCA
  - comparison to PCA: less interpretable, can find non-linear stuctures.

Principal component analysis (PCA)





Projection pursuit methods

- Projection of data to unit vector $u \in \mathbb{R}^p$: project data vectors $x_i \in \mathbb{R}^p$ into one dimension by $y_i = x_i^T u$.
- We fant to find a direction u that maximizes some quantity $g(y)$, where $y = (y_1, \ldots, y_n)$.
  - if g is **variance**, we have **PCA**
  - if g is non-Gaussianity, we have independent component analysis (**ICA**) etc.
- Algorithm to find k-dimensional projections:
  - first find u_1 by maximizing $g(u_1)$.
  - find u_2 by maximizing $g(u_2)$ with the constraint that $u_1^T u_2 = 0$.
  - find u_3 by maximizing $g(u_3)$ with the constraints that $u_1^T u_3 = 0$ and $u_2^T u_3 = 0$ etc.

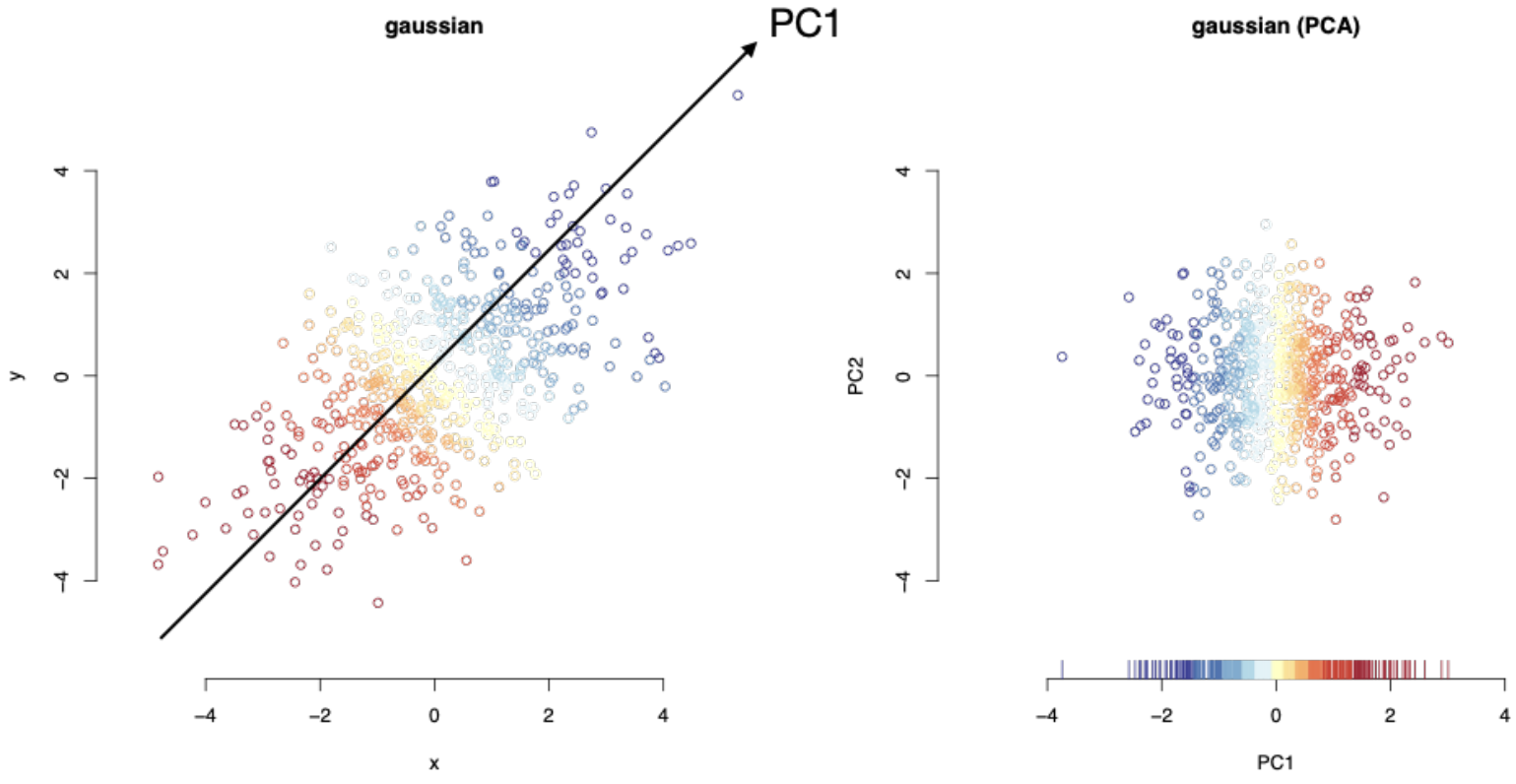
Principal component analysis (PCA)

- In PCA we are finding maximal variance directions ($g(t)$ being the variance of the points in vector t), which is actually the same as finding eigenvectors (and values) of the covariance matrix.
- Assume **data is zero-centered** (and if not, center it: $x_i \leftarrow x_i - \mu$, where $\mu = \sum_{i=1}^{n} x_i / n$.)
- Covariance matrix $\Sigma = \sum_{i=1}^{n} x_i x_i^T / n$ (for zero mean data!).
- Covariance matrix has eigenvalues of $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$ and (orthogonal unit) eigenvectors $u_1, \ldots, u_n$.
- We can show that the projection pursuit algorithm would output vectors $u_1, \ldots, u_d$, where the eigenvalues $\lambda_1, \ldots, \lambda_p$ are the variances to the direction of the eigenvector!
- We call u_1 the first **principal component**, u_2 the second principal component etc.
- The embedding of point i to principal component j is given by $y_i = u_j^T x_i$.
- Read Sec. 10.2.1 from James et al., if unclear.

Toy examples

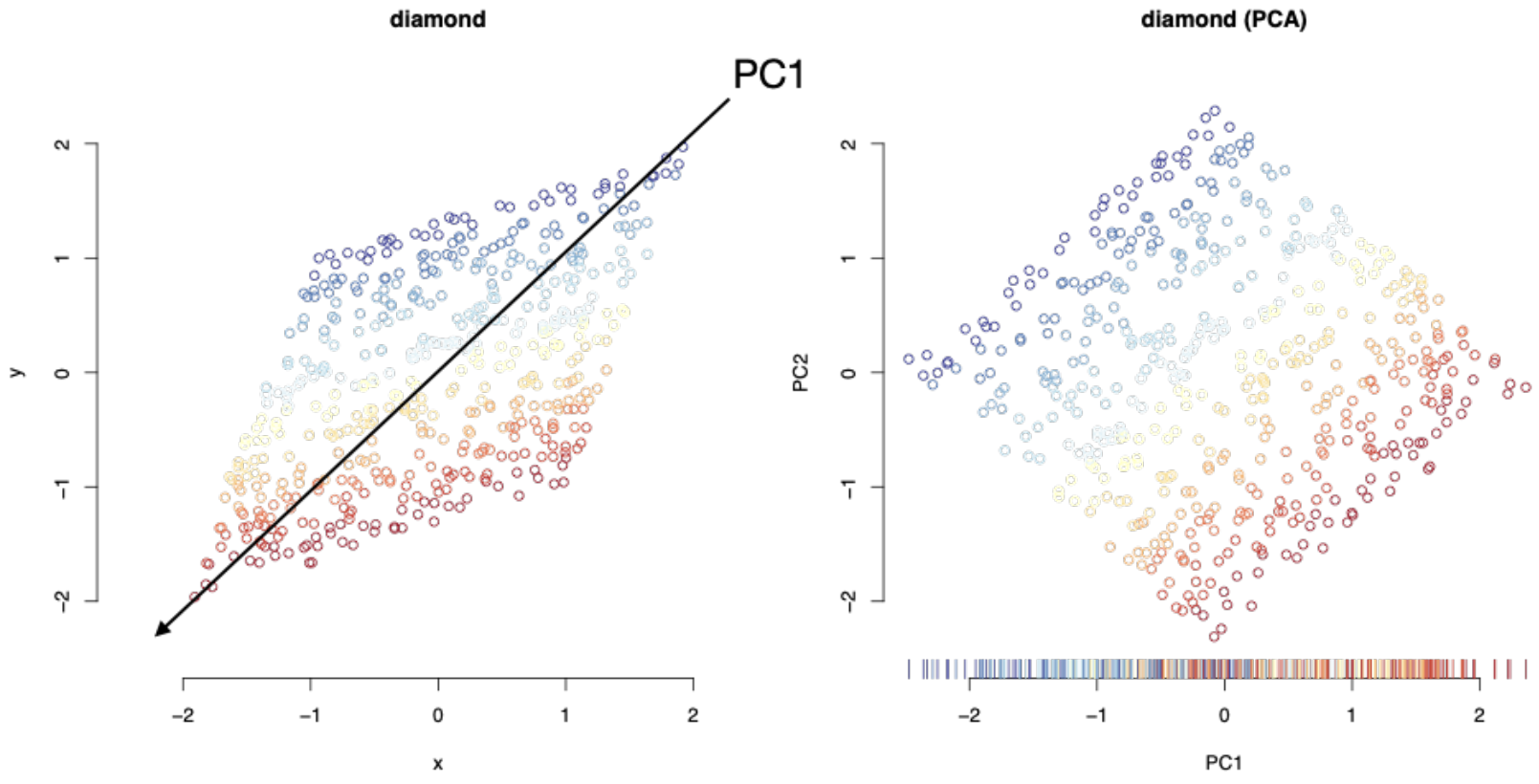
- Project simple 2D datasets into 1D
- Which of the features are preserved in 1D PCA projection?

Gaussian data



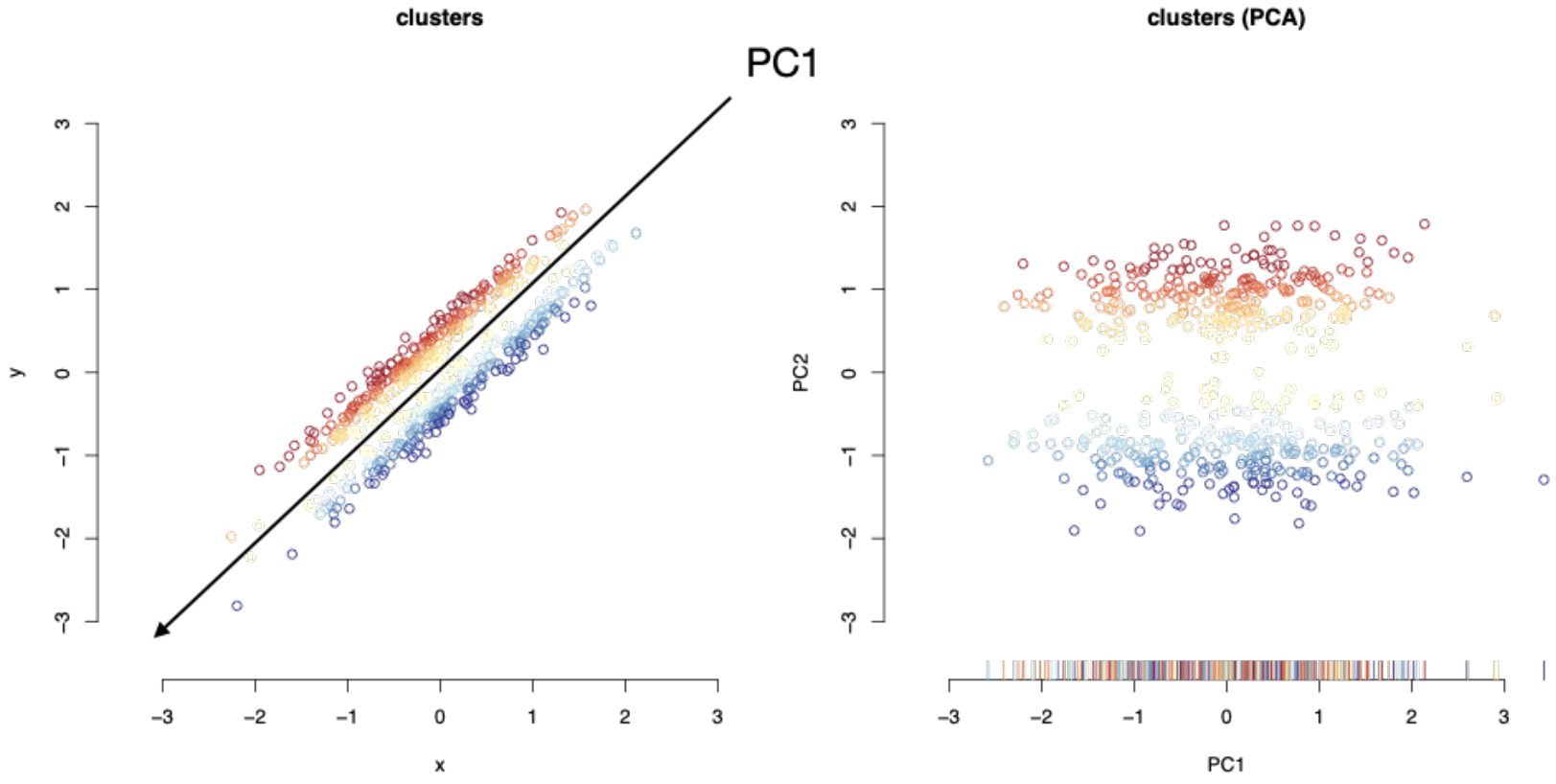
PCA finds the maximal variance direction.

Diamond data



PCA misses the square stucture.

Cluster data



PCA misses the cluster structure.

Implementation and practical issues

- Computing all PCA component takes(naively) $O(np^2)$.
- PCA can be computed by the *singular value decomposition* of the data matrix $X \in \mathbb{R}^{n \times p}$ as

$$X = UDV^T,$$

  where $D \in \mathbb{R}$ is a diagonal matrix containing the eigenvalues and the columns of V contain the eigenvectors.
- Data should almost always be **centered** first (may be done automatically by your PCA library)
- Scaling of columns (e.g., to unit variance) may or may not be necessary
- Efficient numerical libraries exist, use those!

Example: iris data

Biplot

- Embedding of data points into first two principal components
- The principal vector coefficients are expressed as "arrows"

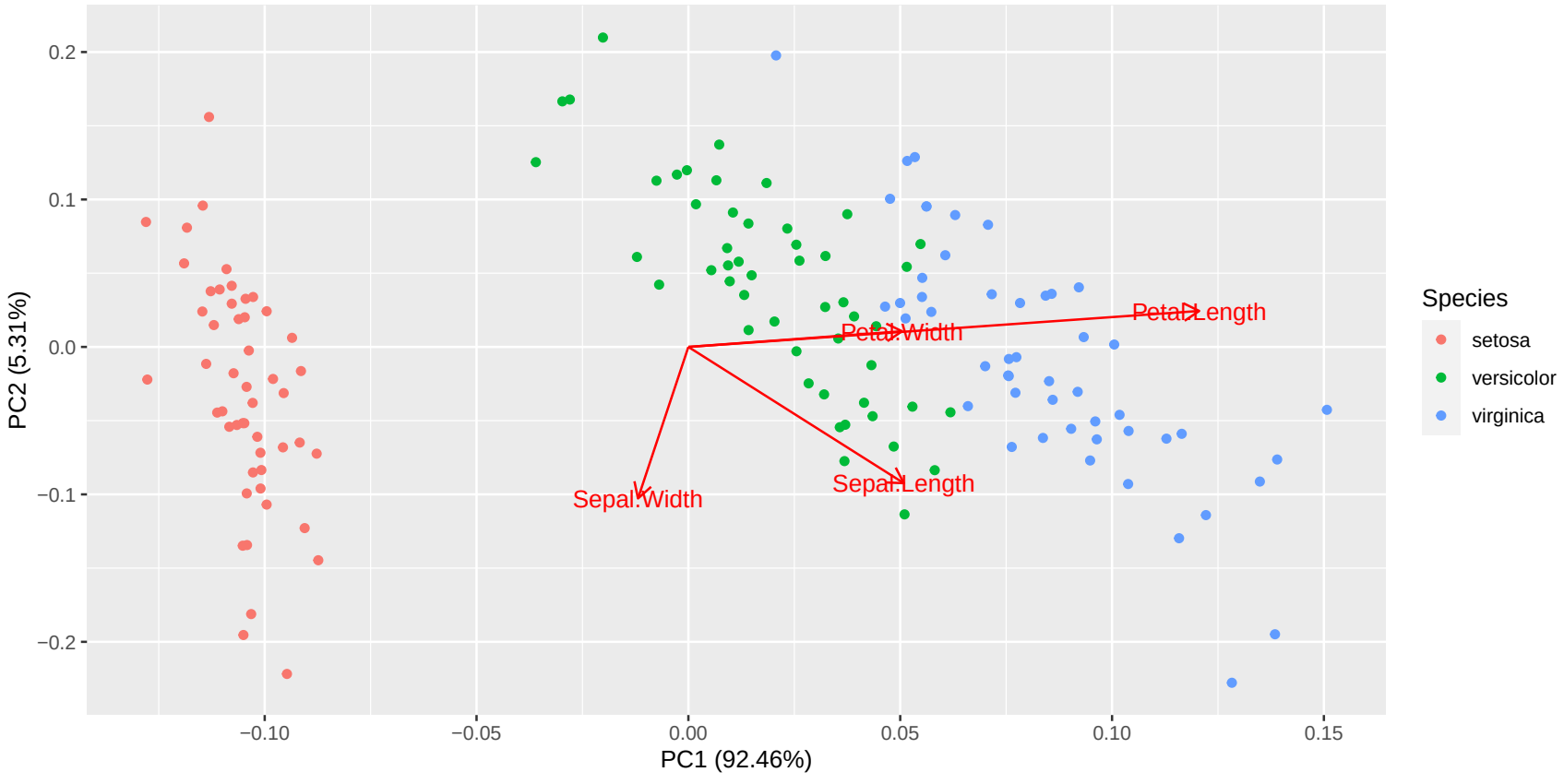
```
iris.pc <- prcomp(iris[,1:4])
```

	PC1	PC2	PC3	PC4
Sepal.Length	0.3613866	-0.6565888	0.5820299	0.3154872
Sepal.Width	-0.0845225	-0.7301614	-0.5979108	-0.3197231
Petal.Length	0.8566706	0.1733727	-0.0762361	-0.4798390
Petal.Width	0.3582892	0.0754810	-0.5458314	0.7536574

Biplot

- Embedding of data points into first two principal components
- The principal vector coefficients are expressed as "arrows"

```
library(ggfortify)
autoplot(iris.pc,data=iris,colour="Species",
         loadings=TRUE,loadings.label=TRUE)
```

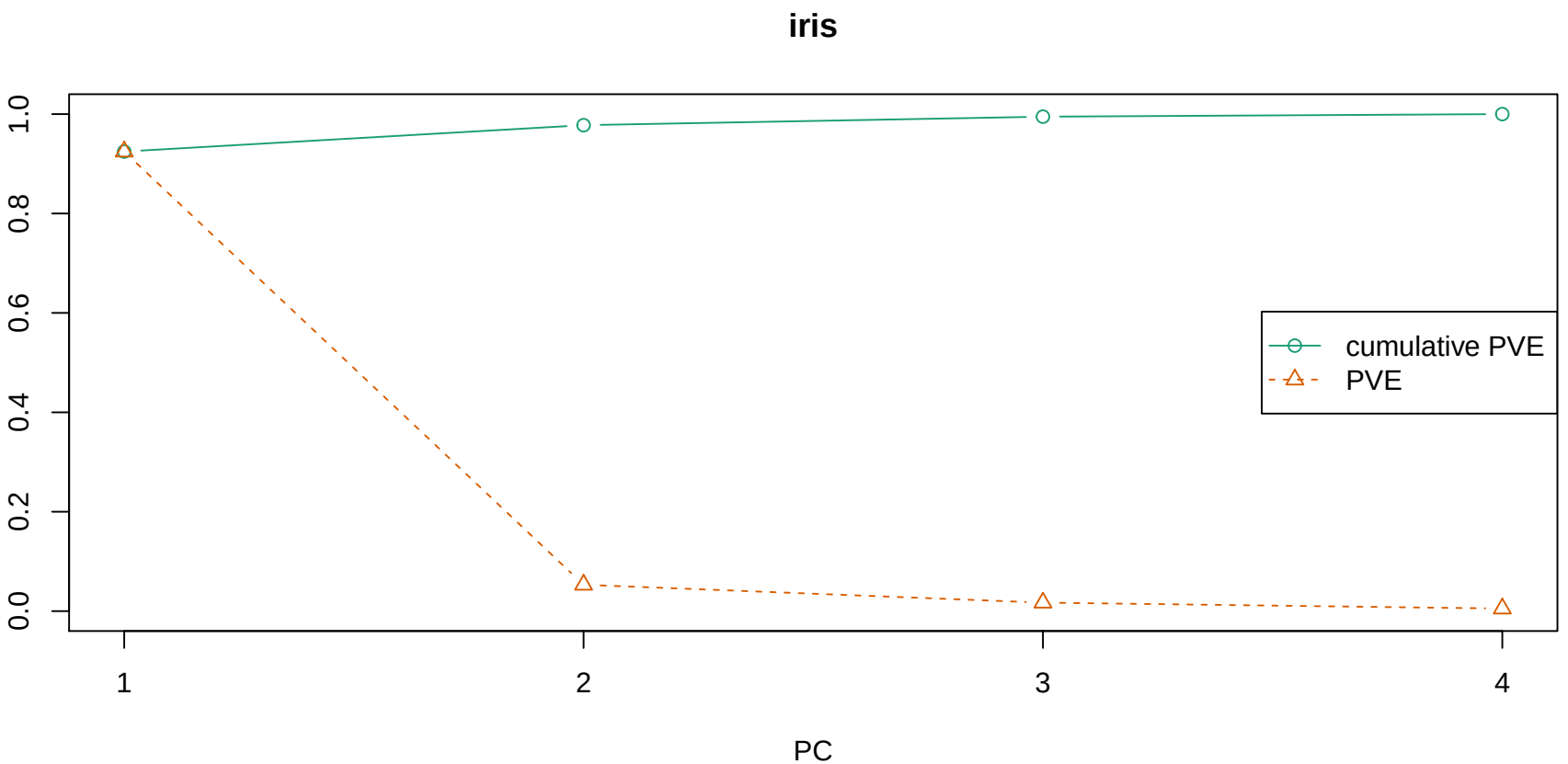


Proportion of variance explained (PVE)

- What is a good number of principal components (e.g., if we do something else than visualisation)?
- The *variance* of the ith principal component is the ith eigenvector of the covariance matrix λ_i.
- Total variance of the data is $\mathrm{var}(X) = \sum_{i=1}^{n} \sum_{j=1}^{p} x_{ij}^2 / n$ (recall: data zero centered!)
- Proportion of variance explained (PVE) by principal component i is then $\lambda_i / \mathrm{var}(X)$.
- Good number of principal components could be, e.g., such that cumulatively explain 90% of the variance.

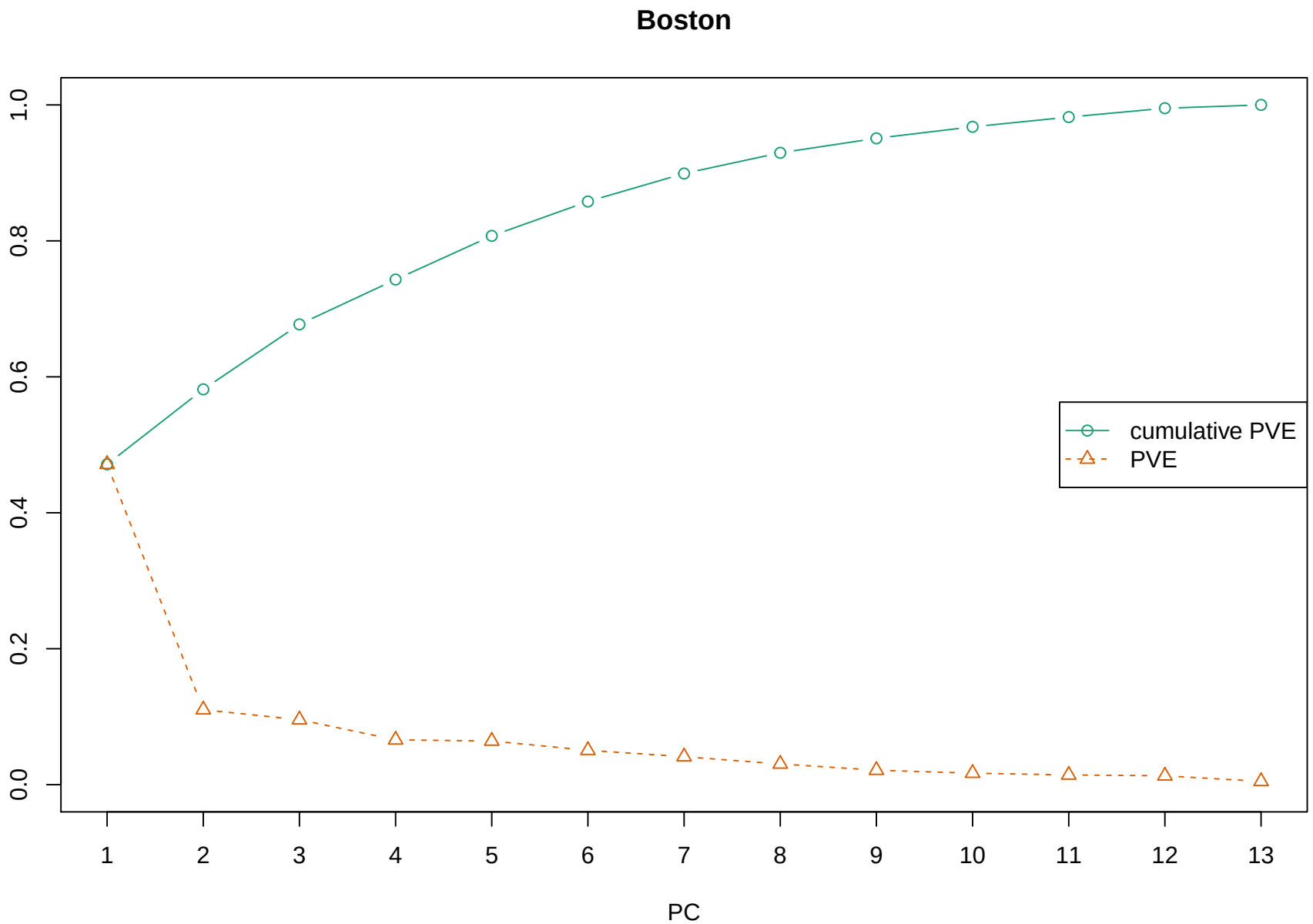
Proportion of variance explained (PVE)

```r
plot_PVE <- function(p,n=length(p$sdev),...) {
  ## https://colorbrewer2.org/?type=qualitative&scheme=Dark2&n=3
  cols3 <- c("#1b9e77","#d95f02","#7570b3")
  PVE <- p$sdev[1:n]^2/sum(p$sdev^2)
  plot(c(1,length(PVE)),c(0,1),type="n",xlab="PC",ylab="",xaxt="n",...)
  axis(1,at=1:length(PVE))
  lines(cumsum(PVE),type="b",col=cols3[1])
  lines(PVE,type="b",col=cols3[2],lty="dashed",pch=2)
  legend("right",c("cumulative PVE","PVE"),col=cols3[1:2],lty=c("solid","dashed"),pch=c(1,2))
}
plot_PVE(iris.pc,main="iris")
```



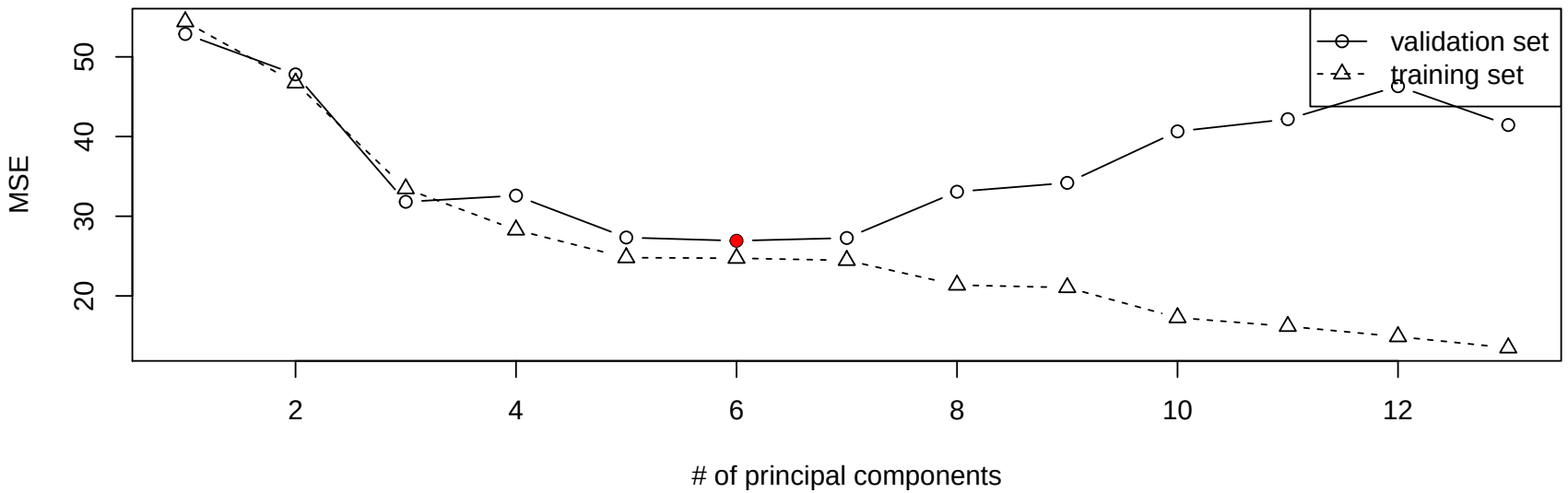
Proportion of variance explained (PVE)

```
Boston.pc <- prcomp(scale(Boston[,1:13]))
plot_PVE(Boston.pc,main="Boston")
```



Using PCA components as features: Boston data set

```
set.seed(1)
i.tr <- sample.int(nrow(Boston),50); i.va <- setdiff(1:nrow(Boston),i.tr)
MSE <- function(j) {
  sapply(1:13,function(i) {
    mean((predict(lm(medv ~ .,data.frame(Boston.pc$x[i.tr,1:i,drop=FALSE],
                                         medv=Boston[i.tr,"medv"])),
                  data.frame(Boston.pc$x[j,1:i,drop=FALSE]))-Boston[j,"medv"])^2)})
}
MSE_va <- MSE(i.va); MSE_tr <- MSE(i.tr)
plot(c(1,13),range(c(MSE_tr,MSE_va)),type="n",xlab="# of principal components",ylab="MSE")
lines(MSE_va,type="b")
lines(MSE_tr,type="b",pch=2,lty="dashed")
i <- which.min(MSE_va); points(i,MSE_va[i],pch=16,col="red")
legend("topright",c("validation set","training set"),lty=c("solid","dashed"),pch=c(1,2))
```



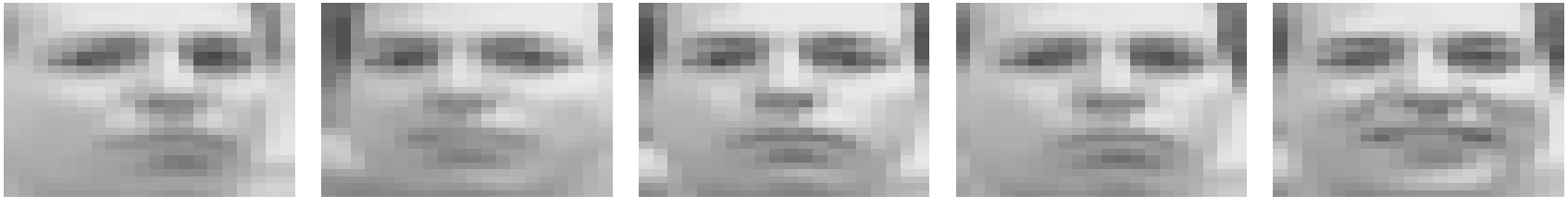
PCA is closely related to k-means clustering

- Principal components are the continuous solutions to the discrete cluster membership indicators for K-means clustering
- Practical implementation: e.g., $k = 3$ cluster centroids should separate well in 2-dimensional PCA projection
- Ding et al. (2004) K-means clustering via principal component analysis. In Proc ICML 2004.

Frey faces

30 (random but ordered) Frey faces from a short video:

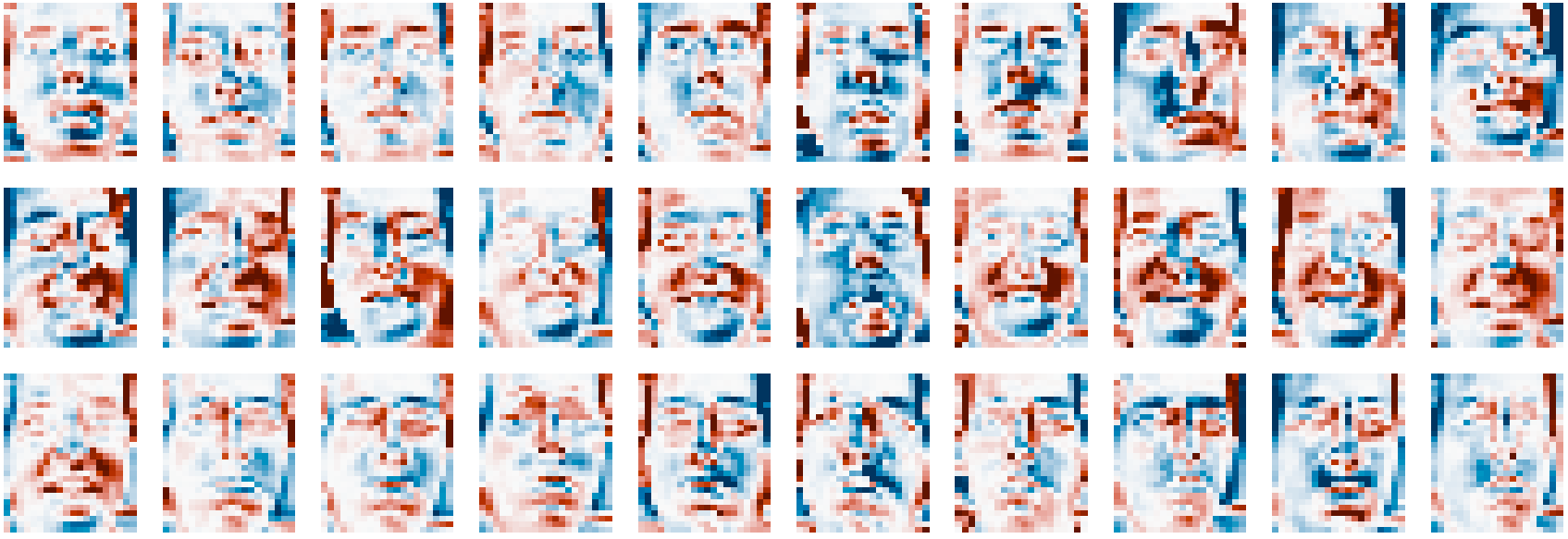
Notice that each frame is a 560-dimensional vector (20x28 grayscale image!) and we can cluster the images or do PCA (the vectors can be visualized as images)! K-means cluster prototype vectors ($k = 5$).



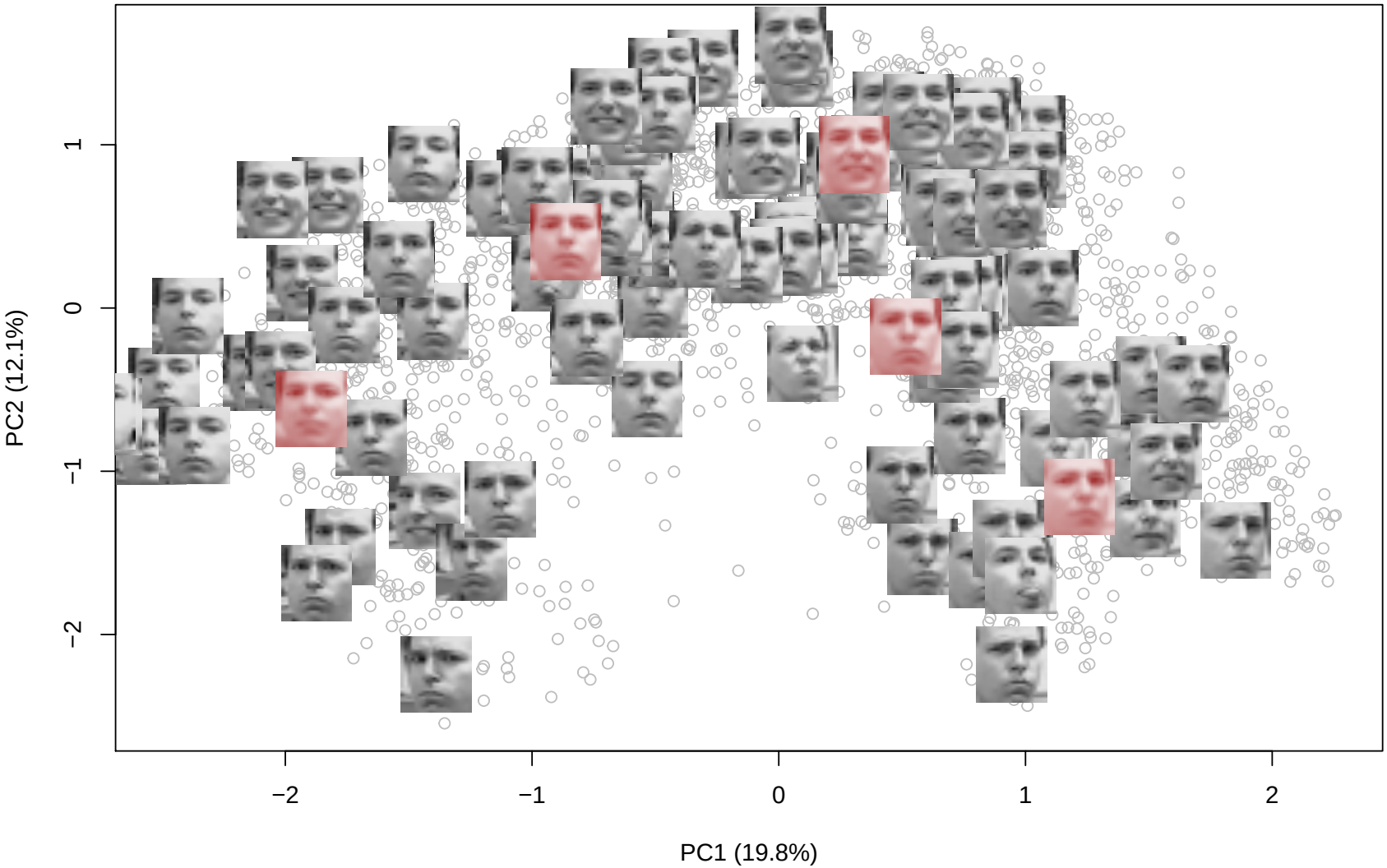
Frey faces: PCA taking away the mean face

The average term μ:

The same 30 faces without the mean term (red=positive, blue=negative), $x_i \leftarrow x_i - \mu$:



Frey faces: PCA and cluster centroids

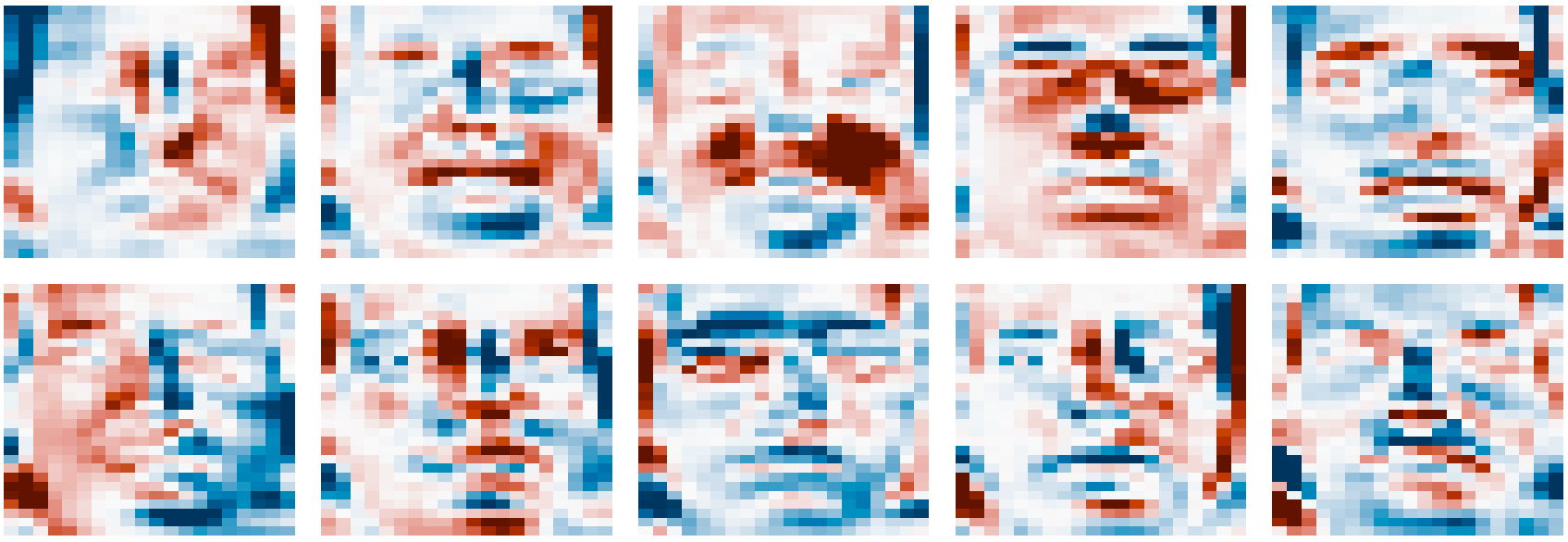


PCA projection of the faces (grayscale images). The cluster centroids are projected to the PCA coordinates and they are shown in red.

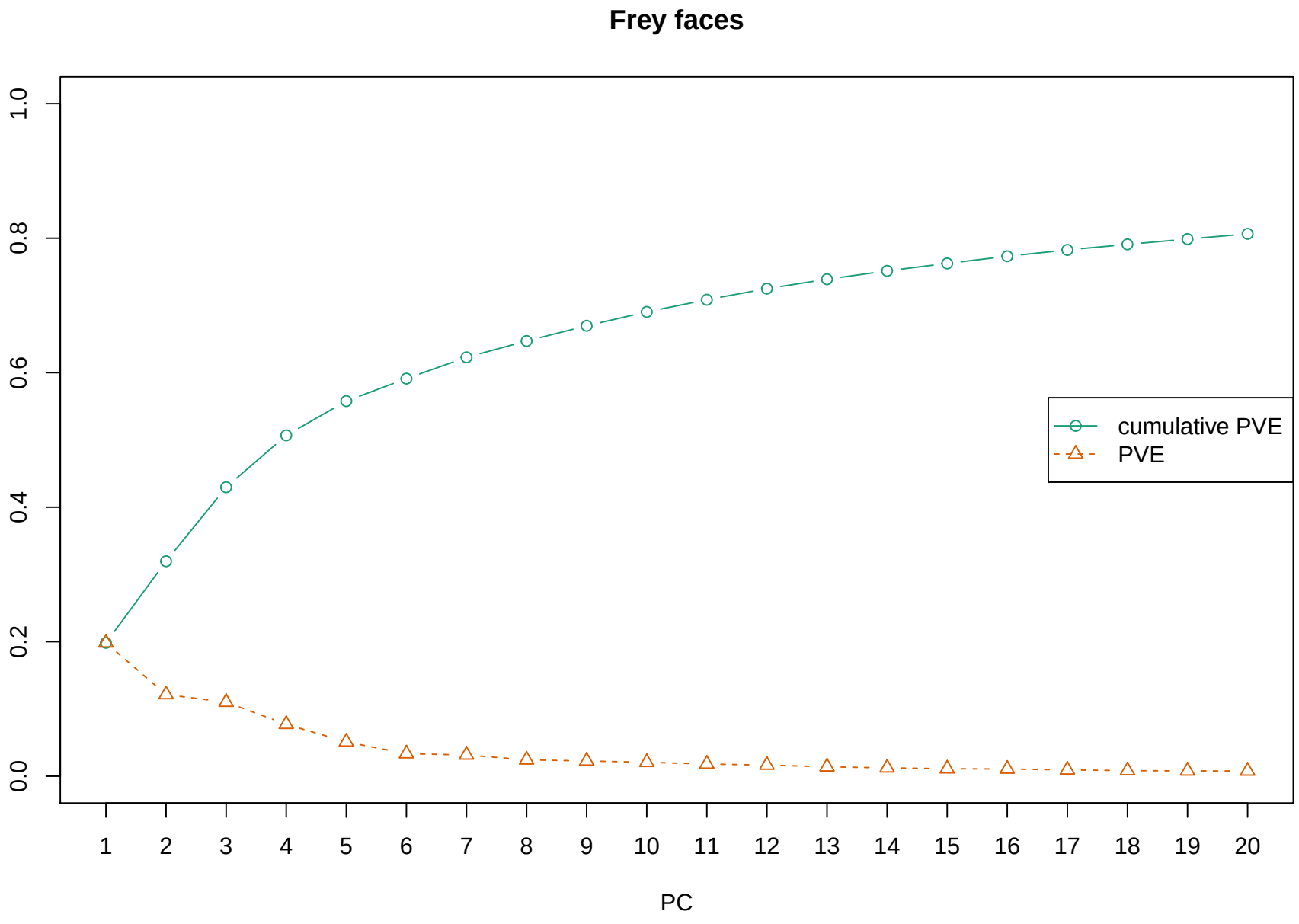
Frey faces: eigenfaces

The average term:

10 first principal components (red=positive, blue=negative):



Frey faces: PVE



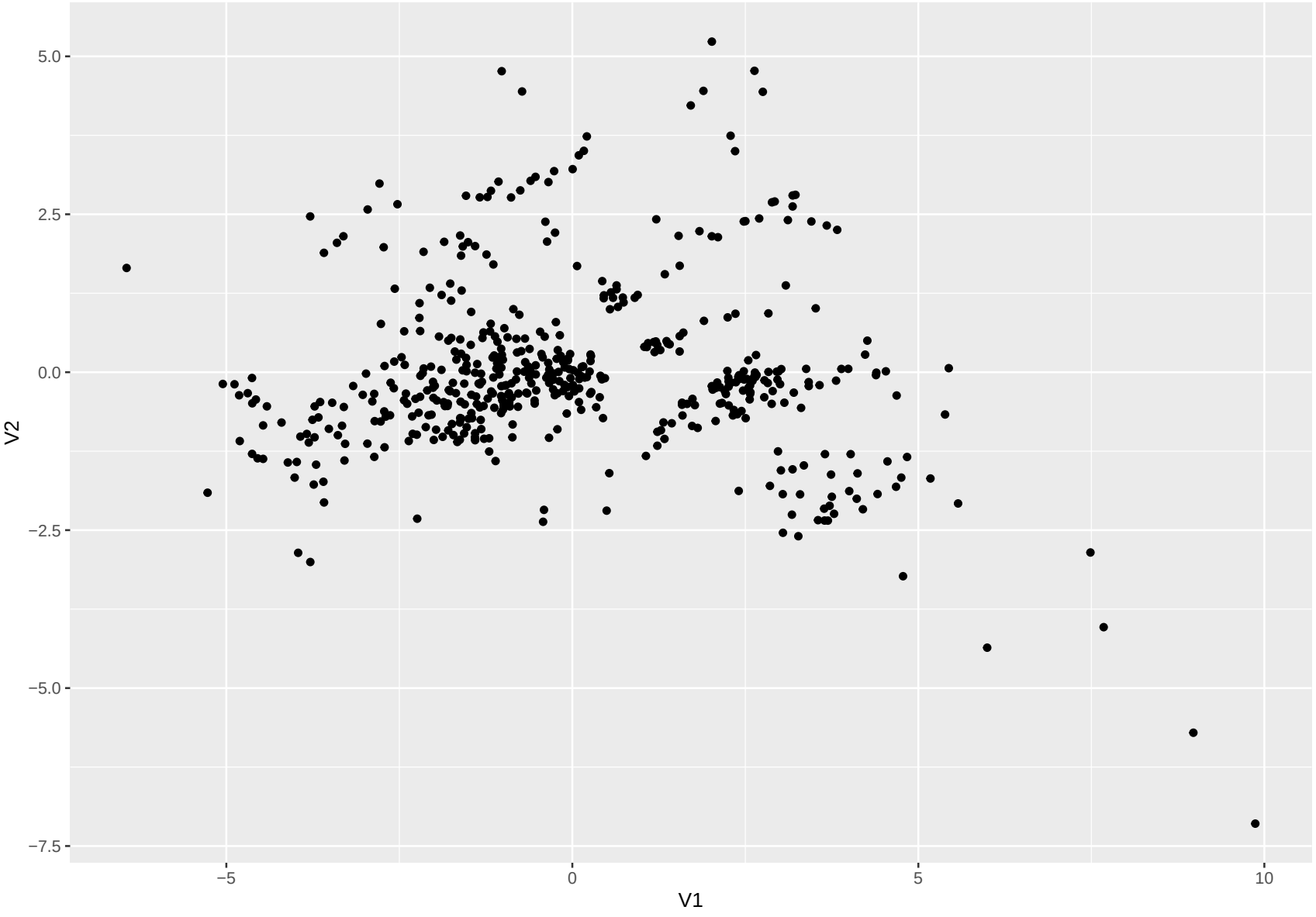
Multidimensional scaling (MDS)

- "Nonlinear PCA", global distance preserving method
- Requires only distances between data points, no vectors needed
- Optimization problem: find the coordinates of embedding $(y_1, \ldots, y_n)$ and a function $f : \mathbb{R} \to \mathbb{R}$ that minimizes *stress* (MDS word for "loss") given by

$$\text{stress} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(\hat{d}(i,j) - f(d(i,j)) \right)^2.$$

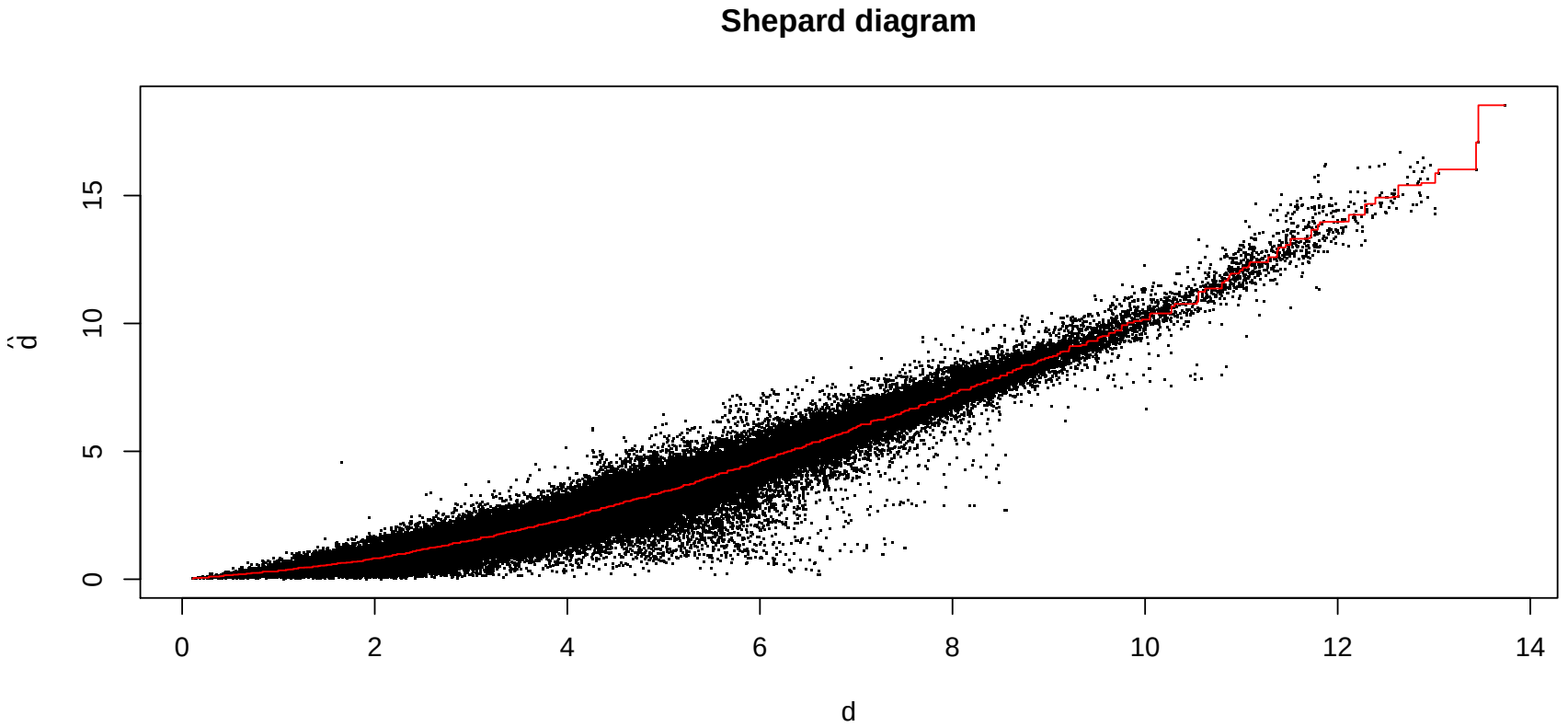
- I.e., find low-dimensional embedding that approximately preserves especially large distances.
- Different types of f define different MDS families. An useful one is *non-metric MDS* in which f can be any monotone function.

Non-metric MDS: Boston data

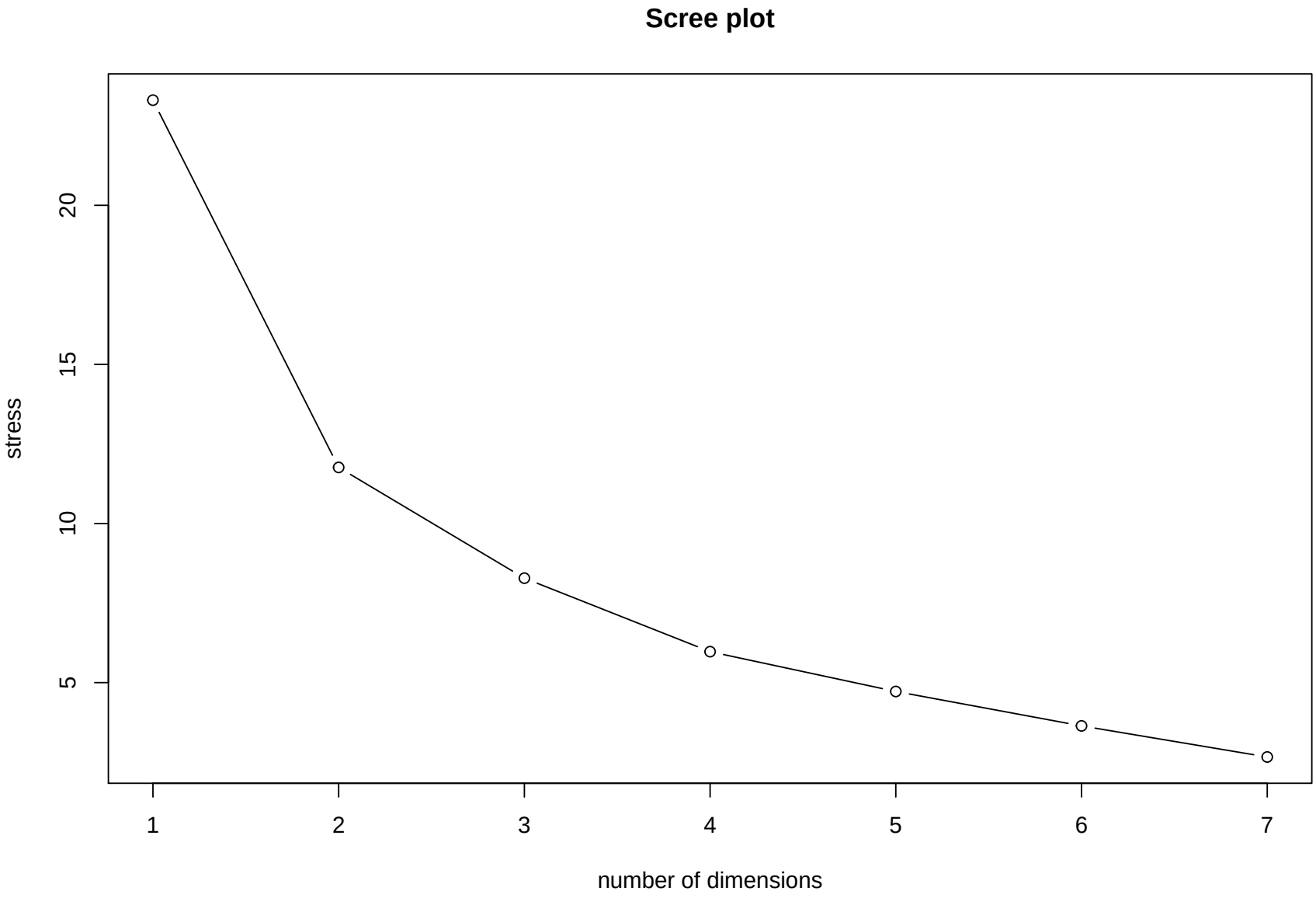


Non-metric MDS: Boston data

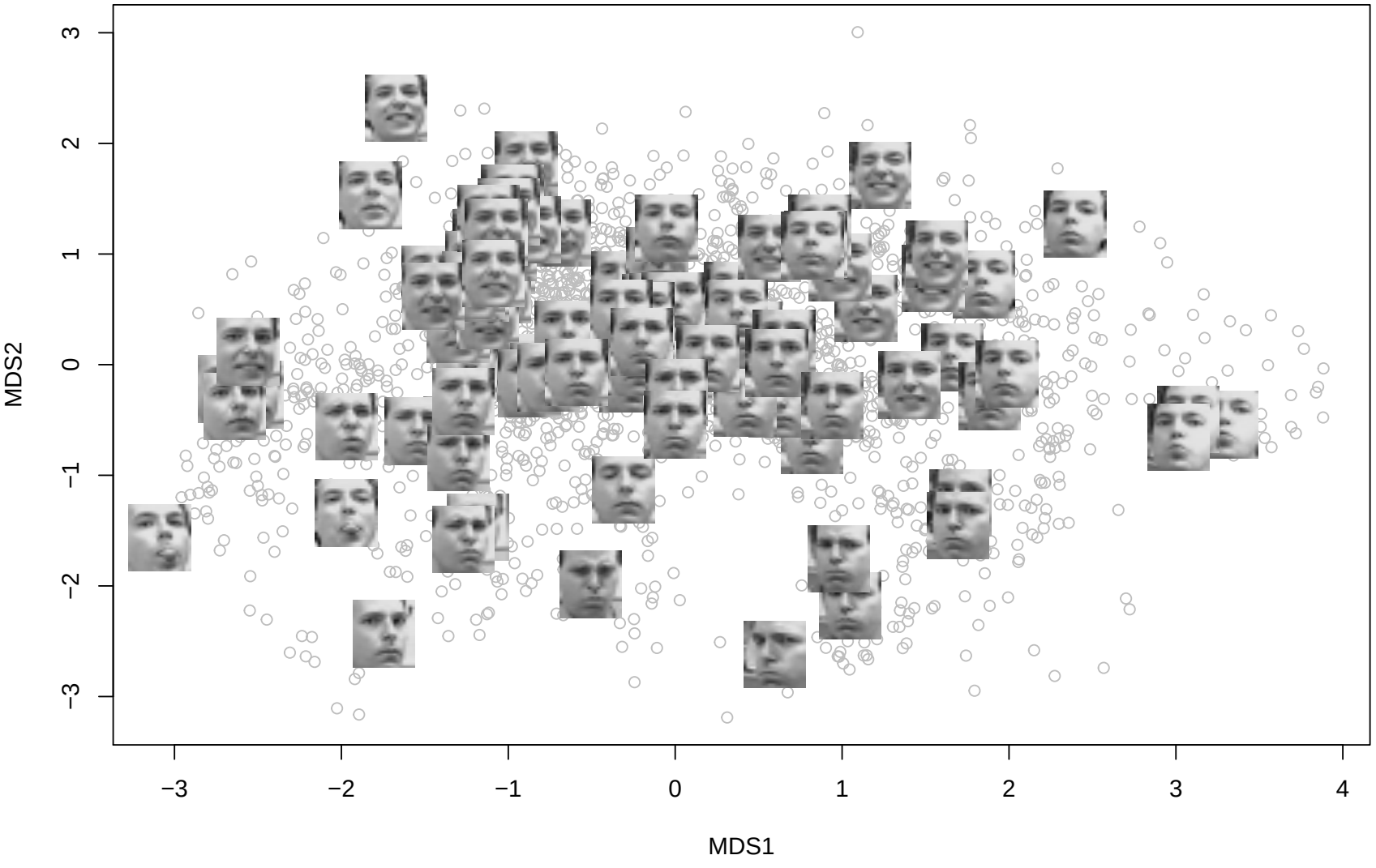
- *Shepard plot* shows the distance in the embedding $\hat{d}(i,j)$ as a function of the original distance $d(i,j)$, i.e., the function f that the non-metric MDS learned
- Notice that small distances are estimated inaccurately!



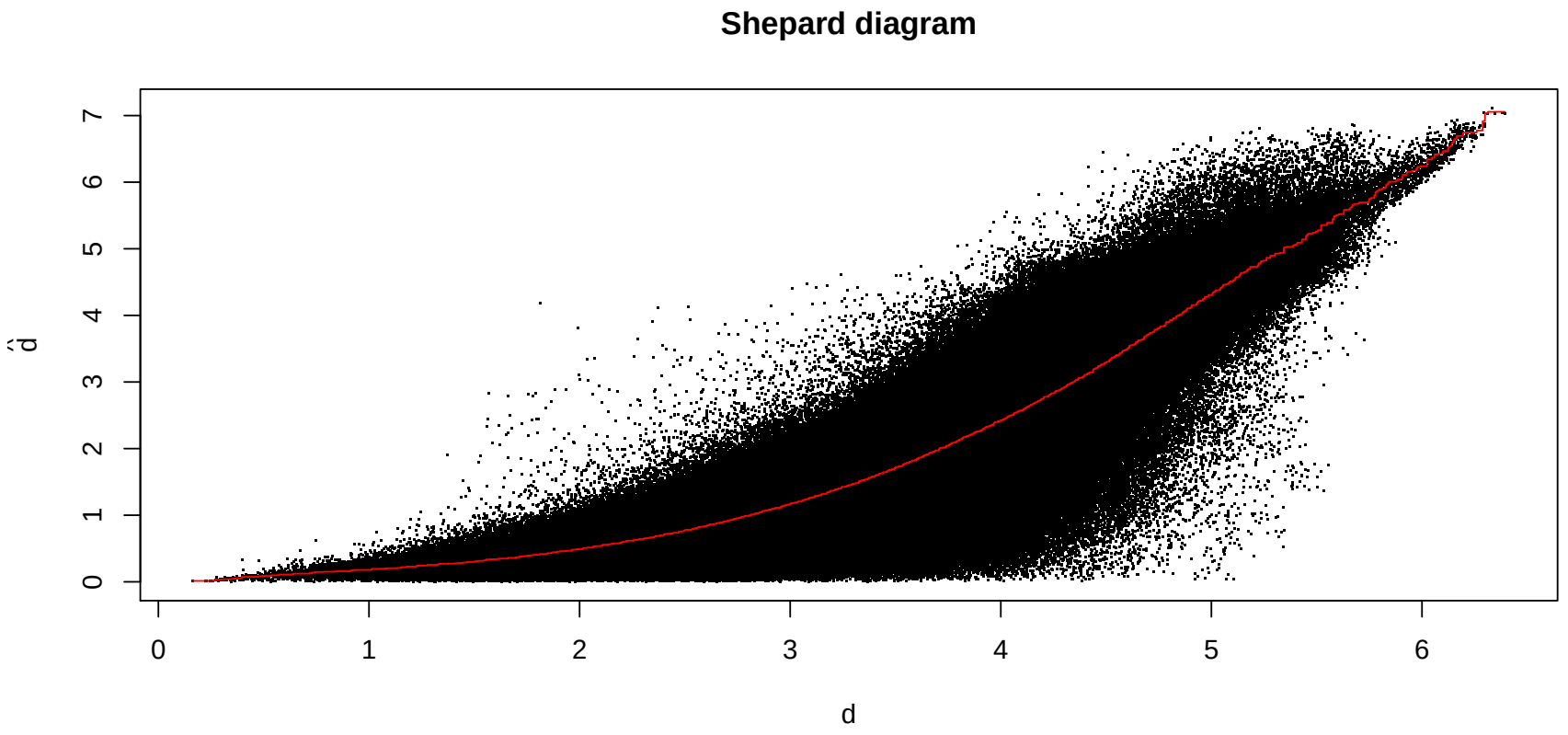
Non-metric MDS: Boston data



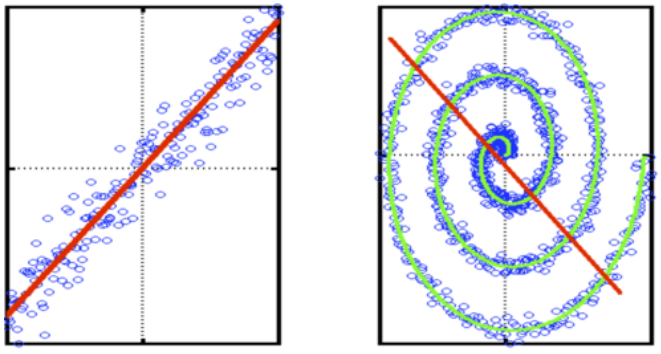
Non-metric MDS: Frey faces



Non-metric MDS: Frey faces



Finding manifolds (local structures)



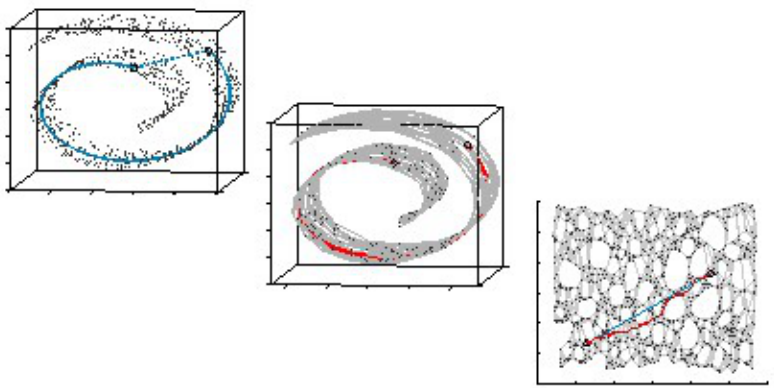
The first principal component found by PCA is given by the red line. The green line gives the "correct" non-linear embedding. PCA or MDS can't find the green component (when projecting the data into 1D).

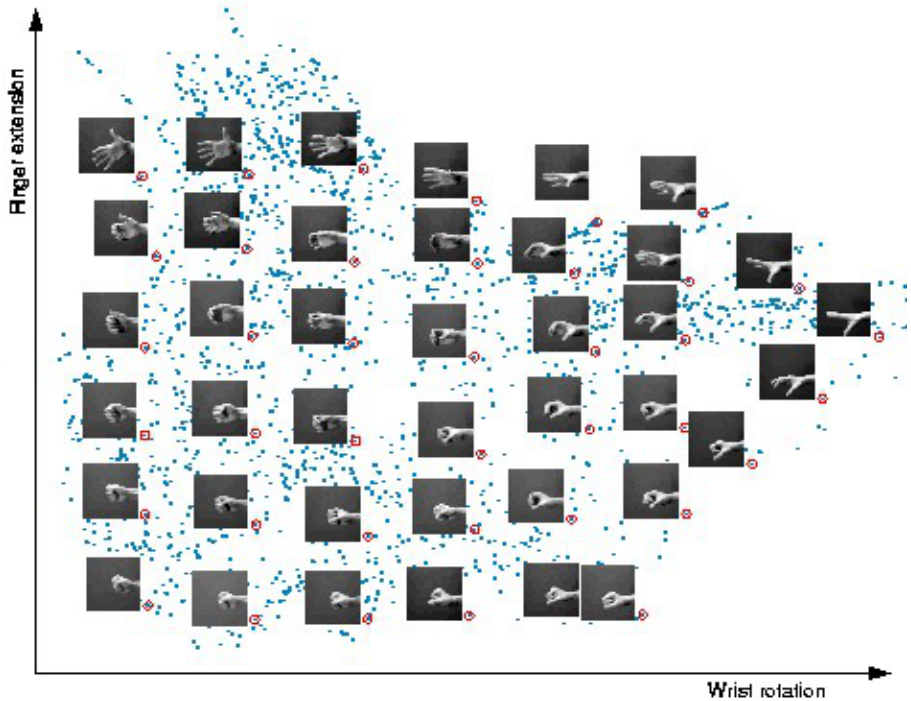
Finding manifolds (local structures)

- Often large-scale structures are not of a great interest
- Instead, we want to find *local* structures...
- ...enter the boom of manifold visualization methods in the early 00s.
- Underlying assumption: the data lies in a low-dimensional but non-linear subspace
- Especially relevant in bioinformatics
- The methods are typically more complex to use, but useful if the data contains "sub-manifold" which the method finds

Isometric mapping of data manifolds (ISOMAP)

- Early method, Tenenbaum et al. (2000) Science.
- Idea: replace the Euclidean distance by graph-theoretic distance, then use non-metric MDS to find the embedding. Makes it possible "flatten" data manifolds into 2D.





t-distributed stochastic neighbor embedding (t-SNE)

- t-SNE is (one of the) current state-of-the-art manifold visualization method, used extensively in many domains
- Flexible and can often find structure where other dimensionality-reduction algorithms cannot
  - . . . but sometimes tricky to interpret.
- van den Maaten et al. (2008) Visualizing Data using t-SNE. JMLR.
- For an educational demo see Wattenberg et al. (2016) How to Use t-SNE Effectively. Distill.

t-SNE: Frey faces