Unsupervised Learning: What is a Sports Car?

Simon Rentzmann* Mario V. Wüthrich†

Prepared for:
Fachgruppe "Data Science"
Swiss Association of Actuaries SAV

Version of October 9, 2019

Abstract

This tutorial studies unsupervised learning methods. Unsupervised learning methods are techniques that aim at reducing the dimension of data (covariables, features), cluster cases with similar features, and graphically illustrate high dimensional data. These techniques do not consider response variables, but they are solely based on the features themselves by studying incorporated similarities. For this reason, these methods belong to the field of unsupervised learning methods. The methods studied in this tutorial comprise principal components analysis (PCA) and bottleneck neural networks (BNNs) for dimension reduction, K-means clustering, K-medoids clustering, partitioning around medoids (PAM) algorithm and clustering with Gaussian mixture models (GMMs) for clustering, and variational autoencoder (VAE), t-distributed stochastic neighbor embedding (t-SNE), uniform manifold approximation and projection (UMAP), self-organizing maps (SOM) and Kohonen maps for visualizing high dimensional data.

Keywords. principal components analysis (PCA), biplot, autoencoder, bottleneck neural network (BNN), minimal spanning tree (MST), K-means clustering, K-medoids clustering, partitioning around medoids (PAM) algorithm, expectation-maximization (EM) algorithm, clustering with Gaussian mixture models (GMMs), variational autoencoder (VAE), t-distributed stochastic neighbor embedding (t-SNE), model-based clustering, uniform manifold approximation and projection (UMAP), self-organizing maps (SOM), Kohonen map, multi-dimensional scaling (MDS).

0 Introduction and overview

This data analytics tutorial has been written for the working group "Data Science" of the Swiss Association of Actuaries SAV, see

`https://www.actuarialdatascience.org`

We are going to discuss three different types of unsupervised learning methods in this tutorial. The first type of methods aims at reducing the dimension of the data; the main objective in this

*AXA Switzerland, simon.rentzmann@axa-winterthur.ch
†RiskLab, Department of Mathematics, ETH Zurich, mario.wuethrich@math.ethz.ch

Electronic copy available at: https://ssrn.com/abstract=3439358

dimension reduction exercise is to minimize the reconstruction error of the original data. The second type of methods is aiming at categorizing data into clusters of similar cases; the main objective in this part is the similarity between cases. The third type of methods mainly aims at visualizing high dimensional data; this is also done by dimension reduction, but the main objective is to keep the (local) topology of the data as far as possible. Figure 1 gives a schematic

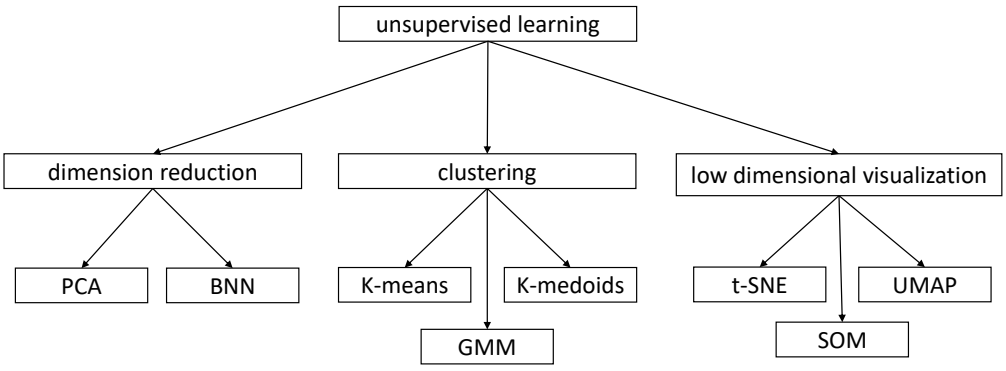

Figure 1: Classification of unsupervised learning methods studied in this tutorial.

overview of the structure of this tutorial.

Clustering or *cluster analysis* is concerned with grouping individual cases that are similar. In doing so, the portfolio of all cases is partitioned into (sub-)groups of similar cases which leads to a segmentation of a heterogeneous portfolio into homogeneous sub-portfolios (of similar cases). Clustering is *only* based on the covariates (features) of the cases without taking into consideration a potential response variable. For this reason, clustering methods are called unsupervised learning methods.
Clustering builds homogeneous (sub-)groups which results in a classification of all cases of a portfolio. Classification can be thought of as an unordered labeling. In insurance we may also be interested into a more continuous representation of these cases. Continuity is understood in the sense that we explicitly try to embed high dimensional cases into lower dimensional spaces. This can be achieved by *dimension reduction* techniques which reduce high dimensional features to lower dimensional objects. Such dimension reduction techniques also belong to the family of unsupervised learning methods if they do not use any information about response variables. We will present two different types of dimension reduction techniques. The first type aims at a representation which leads to a minimal reconstruction error w.r.t. the original features, the second type aims at preserving the original local topology as good as possible for *visualization*.

Overview. The methods discussed in this tutorial comprise principal components analysis (PCA), autoencoders and bottleneck neural networks (BNNs) for dimension reduction, K-means clustering, K-medoids clustering, partitioning around medoids (PAM) algorithm and clustering with Gaussian mixture models (GMMs) for clustering, and t-distributed stochastic neighbor embedding (t-SNE), uniform manifold approximation and projection (UMAP), self-organizing

Electronic copy available at: https://ssrn.com/abstract=3439358

maps (SOM) and Kohonen maps for visualization. These methods are applied to a small data set that we are going to discuss first in the next section.

1 "What is a sports car?", Ingenbleek–Lemaire (1988)

Our analysis of unsupervised learning methods starts from the article "What is a sports car" by Ingenbleek–Lemaire [14]. Unfortunately, only part of the original data set of [14] is still available. Therefore, we have extended this original part of the data with additional cars which have been compiled from the internet.[1] An excerpt of our data is illustrated in Listing 1.[2]

Listing 1: data of car models

```
'data.frame':   475 obs. of  13 variables:
 $ brand           : Factor w/ 43 levels "Alfa Romeo","Audi",..: 3 7 7 11 26 29 29 ...
 $ type            : Factor w/ 96 levels "100","1200","200",..: 68 93 8 55 27 63 64 ...
 $ model           : Factor w/ 113 levels "1 generation",..: 73 55 100 32 88 58 62 ...
 $ cubic_capacity  : int  998 652 602 850 1598 845 956 1588 1596 992 ...
 $ max_power       : int  31 25 21 25 41 21 31 40 40 37 ...
 $ max_torque      : num  67 49 39 60 96 56 65 100 100 98 ...
 $ seats           : int  4 5 4 5 5 4 5 5 5 5 ...
 $ weight          : int  620 755 585 680 1015 695 695 900 1030 920 ...
 $ max_engine_speed: int  5000 5500 5750 5250 4600 4500 5750 4500 4800 4250 ...
 $ seconds_to_100  : num  19.5 26.2 NA 32.3 21 NA 19.3 18.7 20 NA ...
 $ top_speed       : int  129 125 115 125 143 115 137 148 140 130 ...
 $ sports_car      : int  0 0 0 0 0 0 0 0 0 0 ...
 $ tau             : num  23.3 34.1 28.6 32.8 35 ...
```

The data comprises the `brand`, `type` and `model` of the cars considered. For each car we have information about the cubic capacity (in cm^3),[3] the maximal power of engine (in kW),[4] the maximal torque (in Nm), the number of seats, the weight of the car (in kg), the maximum engine speed (in rpm), the acceleration from 0 to 100km/h (in seconds), and the top speed (in km/h), see lines 5-12 of Listing 1. This data is illustrated in Figure 2. We consider the logarithmized data of the weight, the maximal power, the cubic capacity, the maximal torque, the maximal engine speed, the seconds to 100km/h and the top speed of the cars. The diagonal of Figure 2 shows the QQ plots (w.r.t. a Gaussian distribution). We observe that these variables look fairly Gaussian on the log-scale, only the maximal engine speed MES_l seems to be more skewed. The lower left part of Figure 2 gives the corresponding scatter plots with the resulting absolute values of correlations provided in the upper right part.

We focus on the continuous variables. The treatment of categorical variables is more difficult because they do not offer a canonical distance function, we also refer to Section 6, below.

The information of Listing 1 was used in the 1970s in Belgium to discriminate sports cars from ordinary cars. This discrimination had a direct impact on the prices of insurance policies.[5] The

[1] The additional cars compiled from the internet belong to the same time period as the ones selected from Ingenbleek–Lemaire [14]. This ensures consistency of the merged data set.

[2] The data set is available from https://github.com/JSchelldorfer/ActuarialDataScience

[3] 1 liter equals $1000cm^3$.

[4] 1 horse power equals 0.735499kW.

[5] Of course, today, one would directly use this original information in a GLM, a GAM or a neural network regression model for insurance pricing. However, we revisit this Belgium sports car example because it gives us a nice show case for illustrating different unsupervised learning methods.

Electronic copy available at: https://ssrn.com/abstract=3439358

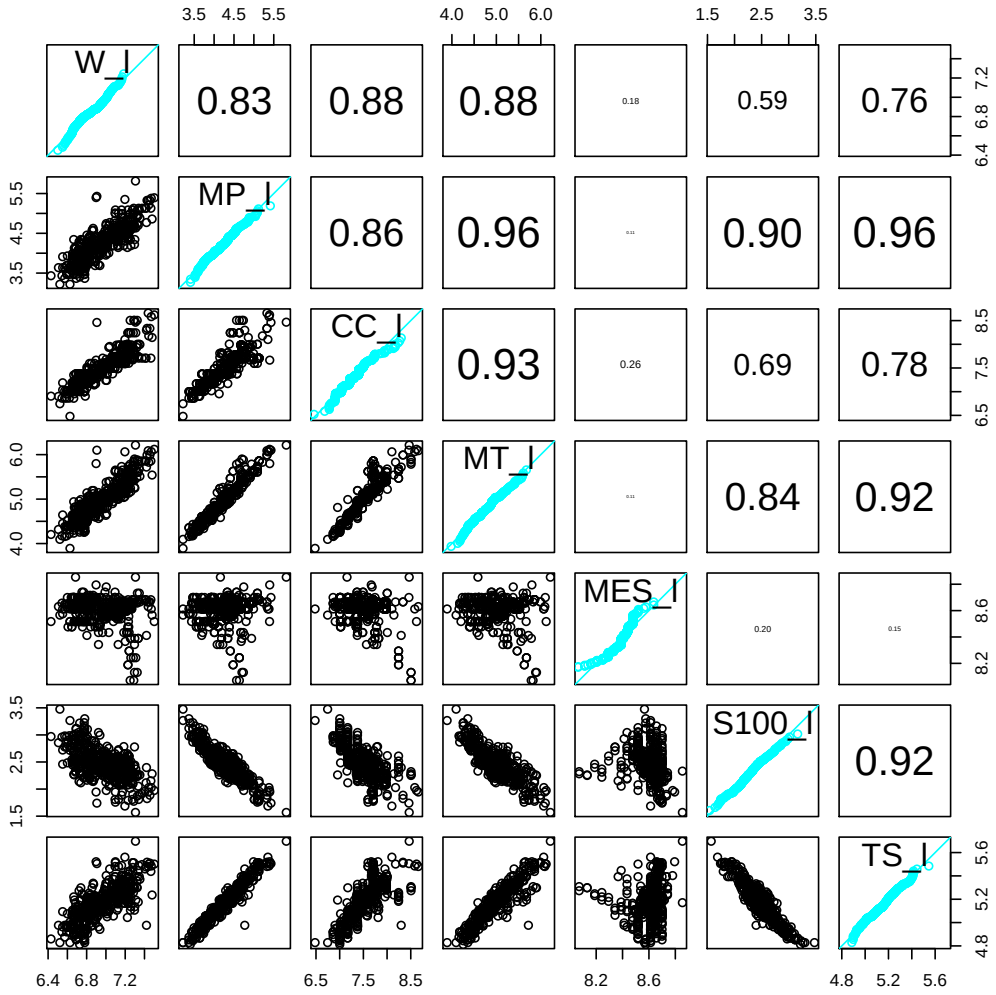

Figure 2: Scatter plots (lower left triangle), QQ plots (diagonal) and absolute values of correlations (upper right triangle) of the logarithmized data W_l = log(`weight`), MP_l = log(`max_power`), CC_l = log(`cubic_capacity`), MT_l = log(`max_torque`), MES_l = log(`max_engine_speed`), S100_l = log(`seconds_to_100`) and TS_l = log(`top_speed`).

following formula had been derived by car experts in Belgium for the classification of sports cars and ordinary cars, respectively, see Ingenbleek–Lemaire [14]. Define the variables

$$\tau = \frac{\texttt{weight}}{\left(\frac{\texttt{max\_power}}{0.735499}\right)} \sqrt[3]{\texttt{seats}} \sqrt[4]{\frac{\texttt{cubic\_capacity}}{1000}}, \qquad \text{and, respectively,}$$

$$\tau^{+} = \frac{1000^{1/4}}{0.735499}\,\tau = \frac{\texttt{weight}}{\texttt{max\_power}} \sqrt[3]{\texttt{seats}} \sqrt[4]{\texttt{cubic\_capacity}}.$$

The values of τ are provided on line 14 of Listing 1. A car is defined to be a sports car iff

$$\tau < 17 \qquad \text{or, equivalently,} \qquad \tau^{+} < 129.9773.$$

Electronic copy available at: https://ssrn.com/abstract=3439358

This results in the binary labels on line 13 of Listing 1. Thus, the Belgium sports car classification is done with a multiplicative formula. On the log-scale we receive a linear formula, namely,

$$\begin{aligned} y \;=\; \log(\tau^+) \;&=\; \log(\texttt{weight}) - \log(\texttt{max\_power}) + \frac{1}{3}\log(\texttt{seats}) \\ &\quad + \frac{1}{4}\log(\texttt{cubic\_capacity}) \;\overset{??}{<}\; \log(129.9773) = 4.86736. \end{aligned} \tag{1.1}$$

This latter formula corresponds to a linear regression equation with given slopes. In Figure 3

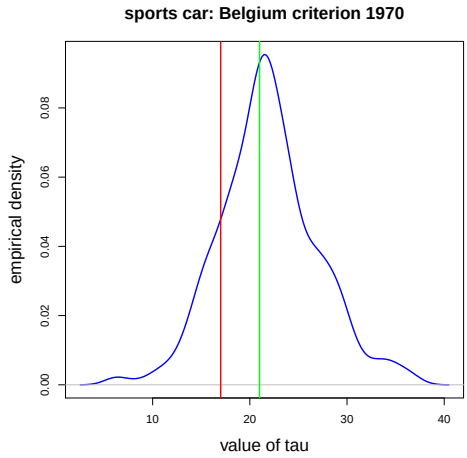

Figure 3: Empirical density of variable τ over all $n = 475$ cars (line 14 in Listing 1), the red vertical line shows the critical value of $\tau < 17$ for sports cars, and the green line $\tau < 21$ (which collects 42.7% of all cars considered).

we plot the empirical density of τ over all $n = 475$ cars in our data of Listing 1. The red vertical line is the critical value of $\tau < 17$ for sports cars. 15.2% of all cars have a value τ smaller than this critical value, and 42.7% of all cars have a value τ smaller than 21 (green line in Figure 3). The goal of Ingenbleek–Lemaire [14] has been to improve this expert choice using principal components analysis (PCA). First, Ingenbleek–Lemaire [14] defined the following features (covariates)

$$\begin{aligned} x_1^* &= \log(\texttt{weight/max\_power}), \\ x_2^* &= \log(\texttt{max\_power/cubic\_capacity}), \\ x_3^* &= \log(\texttt{max\_torque}), \\ x_4^* &= \log(\texttt{max\_engine\_speed}), \\ x_5^* &= \log(\texttt{cubic\_capacity}). \end{aligned} \tag{1.2}$$

These new features are illustrated in Figure 4. The diagonal shows the QQ plots of $x_1^*, \ldots, x_5^*$, the lower left triangle the scatter plots and the upper right triangle the resulting absolute values of the correlations.

Based on the features (1.2), the goal of Ingenbleek–Lemaire [14] is to find good regression parameters $\alpha_1, \ldots, \alpha_5$ such that the following dependent variable allows us to discriminate sports cars from ordinary cars

$$y^* = \alpha_1 x_1^* + \alpha_2 x_2^* + \alpha_3 x_3^* + \alpha_4 x_4^* + \alpha_5 x_5^*. \tag{1.3}$$

Electronic copy available at: https://ssrn.com/abstract=3439358

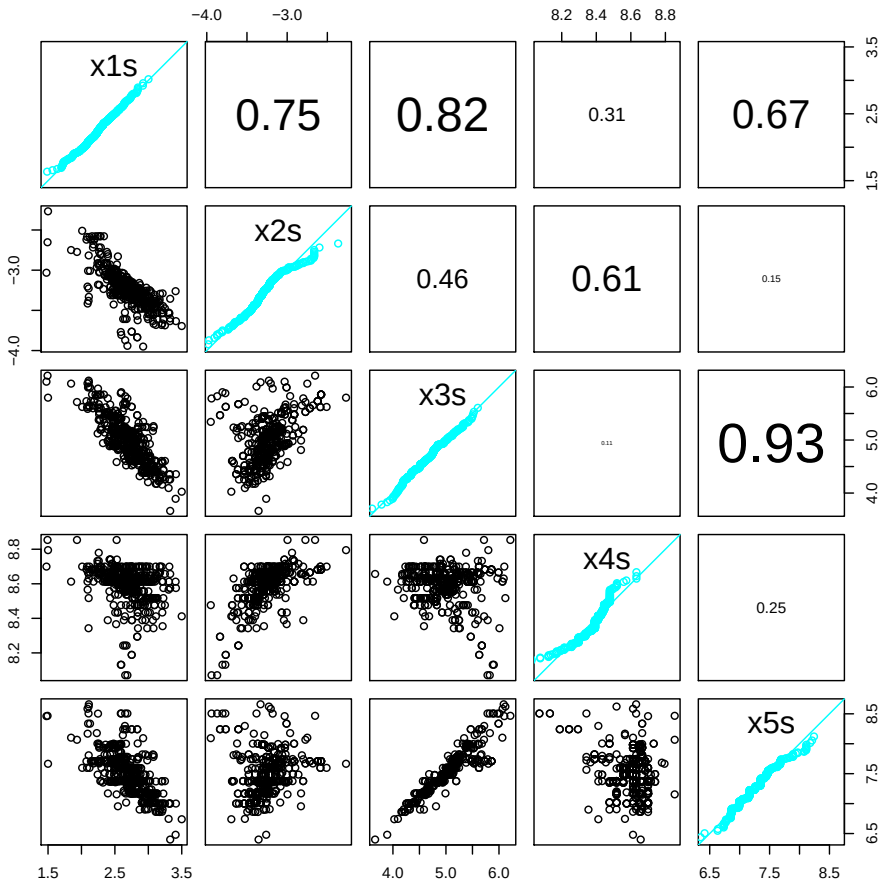

Figure 4: Scatter plots (lower left triangle), QQ plots (diagonal) and absolute values of correlations (upper right triangle) of the variables $x_1^*, \ldots, x_5^*$ (denoted by `x1s`, ..., `x5s` in the plots).

Note that we can re-express this dependent variable in terms of the original features

$$\begin{aligned} y^* \quad = \quad & \alpha_1 \log(\texttt{weight}) - (\alpha_1 - \alpha_2) \log(\texttt{max\_power}) + \alpha_3 \log(\texttt{max\_torque}) \\ & + \alpha_4 \log(\texttt{max\_engine\_speed}) - (\alpha_2 - \alpha_5) \log(\texttt{cubic\_capacity}). \end{aligned} \tag{1.4}$$

Remarks.

- From (1.4) we see that each considered log-variable receives its own (independent) parameter $\alpha_1^\star = \alpha_1$, $\alpha_2^\star = -\alpha_1 + \alpha_2$, $\alpha_3^\star = \alpha_3$, $\alpha_4^\star = \alpha_4$ and $\alpha_5^\star = -\alpha_2 + \alpha_5$. Therefore, transformation (1.2) to the x^* variables would not be necessary.
- Compared to the original formula (1.1) the number of seats is dropped in formula (1.4), and on the other hand log(`max_torque`) and log(`max_engine_speed`) are added.

To keep comparability with Ingenbleek–Lemaire [14] we use the variables defined in (1.2) and illustrated in Figure 4, and we set for our cases

$$\boldsymbol{x}^* = (x_1^*, \ldots, x_5^*)^\top \in \mathbb{R}^q \qquad \text{with } q = 5.$$

Electronic copy available at: https://ssrn.com/abstract=3439358

In the sequel of this tutorial we are going to explain several dimension reduction techniques and clustering methods on this data $\boldsymbol{x}_i^* \in \mathbb{R}^q$ over all available cars $i = 1, \ldots, n = 475$.

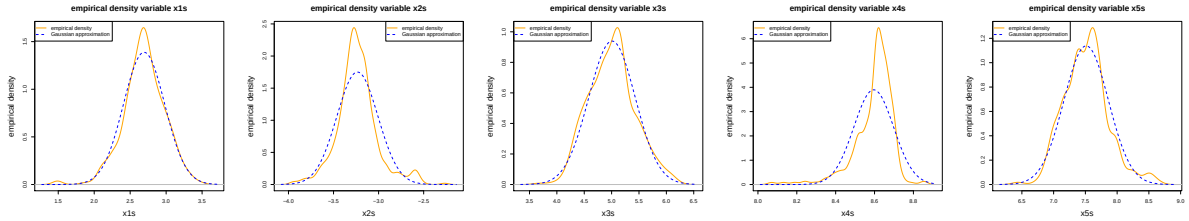

Figure 5: Empirical marginal densities of the features $x_1^*, \ldots, x_5^*$ (from left to right) over all available cars (orange color) and Gaussian approximation (blue color), compare with Figure 4.

In Figure 5 we provide the empirical marginal densities of the features $x_1^*, \ldots, x_5^*$ (from left to right) over all available cars and we compare it to a Gaussian approximation (this corresponds to the QQ plots on the diagonal of Figure 4). For components $x_1^*, x_2^*, x_3^*, x_5^*$ the Gaussian approximation works reasonably well, for component x_4^*, log(`max_engine_speed`), the Gaussian approximation does not look reasonable. We are mentioning this because in most subsequent analysis we are using the (squared) Euclidean distance in the objective function which is the canonical choice for (multivariate) Gaussian random variables (vectors). Moreover, for studying multivariate Gaussian distributions we should have Gaussian copulas (dependence structure) which is not fully justified by the scatter plots in the lower left triangle of Figure 4, because not all of these scatter plots have an elliptic shape.

2 Principal components analysis

In a nutshell, a PCA aims at reducing the dimension of high dimensional data such that the reconstruction error relative to the original data is minimized. If applied successfully, it reduces the dimension of the feature space, and it is particularly useful for (actuarial) regression modeling because it provides a small number of uncorrelated explanatory variables.

2.1 Standardized design matrix

Assume we have $n \in \mathbb{N}$ cases in our portfolio described by features (covariates) $\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^* \in \mathbb{R}^q$. These cases provide a *raw design matrix* $\mathfrak{X}^* = (\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^*)^\top \in \mathbb{R}^{n \times q}$. Thus, each row $(\boldsymbol{x}_i^*)^\top$ of $\mathfrak{X}^*$ describes a single case, and the columns $1 \le j \le q$ describe a given component $x_{i,j}^*$ over all cases $1 \le i \le n$, see (1.2) for the meaning of these components.
Typically, the columns $1 \le j \le q$ of the raw design matrix $\mathfrak{X}^* \in \mathbb{R}^{n \times q}$ live on different scales and they are correlated. We give an analogy in a multivariate Gaussian context: if we choose $\boldsymbol{x}_i^* \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, we may get an empirical distribution with level sets as in Figure 6.
From the observed cases $\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^* \in \mathbb{R}^q$ we can estimate the sample means, variances[6] and

[6] On purpose, we do not choose the unbiased versions of the variance estimates, here.

Electronic copy available at: https://ssrn.com/abstract=3439358

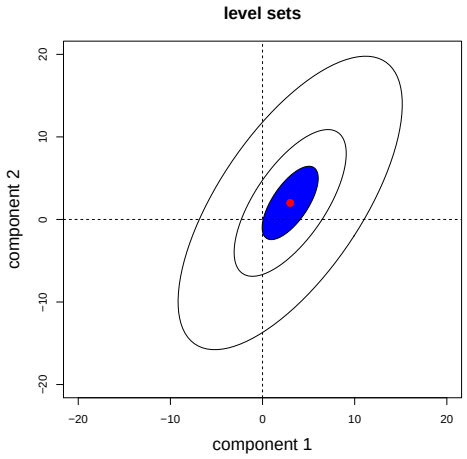

Figure 6: Level sets of a two dimensional Gaussian distribution with general mean vector $\boldsymbol{\mu} \in \mathbb{R}^q$ and symmetric and positive definite covariance matrix $\Sigma \in \mathbb{R}^{q \times q}$ illustrating a potential distribution of the cases $\boldsymbol{x}_i^*$ (here we set $q = 2$).

covariances of the columns of $\mathfrak{X}^*$. These are for $1 \leq j, l \leq q$ given by

$$\widehat{\mu}_j = \frac{1}{n} \sum_{i=1}^{n} x_{i,j}^*, \tag{2.1}$$

$$\widehat{\sigma}_j^2 = \frac{1}{n} \sum_{i=1}^{n} \left(x_{i,j}^* - \widehat{\mu}_j \right)^2, \tag{2.2}$$

$$\widehat{\sigma}_{j,l} = \frac{1}{n} \sum_{i=1}^{n} \left(x_{i,j}^* - \widehat{\mu}_j \right) \left(x_{i,l}^* - \widehat{\mu}_l \right). \tag{2.3}$$

Since all further derivations should not depend on translations and scales we standardize the raw design matrix $\mathfrak{X}^*$ (by column means and column standard deviations). We first center the columns of the raw design matrix $\mathfrak{X}^*$ by subtracting $(\widehat{\mu}_1, \ldots, \widehat{\mu}_q)$ on each row; in the multivariate Gaussian case $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ this corresponds to setting mean vector $\boldsymbol{\mu} = \mathbf{0}$.
After this centering the columns of the resulting design matrix may still live on different scales, since the components of the cases may be measured in different units. Different scales are reflected by sample variances $\widehat{\sigma}_j^2$ having different magnitudes; in the multivariate Gaussian case $\mathcal{N}(\mathbf{0}, \Sigma)$ this corresponds to a covariance matrix $\Sigma \in \mathbb{R}^{q \times q}$ which is not constant on its diagonal entries $(\Sigma_{j,j})_{1 \leq j \leq q}$.
Since all further derivations should not depend on the different units of the components we scale all centered columns of the raw design matrix by the sample standard deviations $(\widehat{\sigma}_1, \ldots, \widehat{\sigma}_q)$; in the multivariate Gaussian case this corresponds to transforming the covariance matrix Σ to its correlation matrix.
The resulting *standardized design matrix* is denoted by $\mathbf{X} \in \mathbb{R}^{n \times q}$. Its columns have sample mean zero and sample variance 1. We denote its (standardized) entries by $\mathbf{X} = (x_{i,j})_{1 \leq i \leq n, 1 \leq j \leq q} \in \mathbb{R}^{n \times q}$. In the standardized case, formula (2.3) relates to the sample correlations between the columns of $\mathbf{X}$. Our goal will be to explore these correlations. The following convention is used throughout.

Electronic copy available at: https://ssrn.com/abstract=3439358

Convention 2.1 *We always assume that $\mathbf{X} \in \mathbb{R}^{n \times q}$ is standardized and we simply call it design matrix. The (transposed) rows of $\mathbf{X}$ are denoted by $\boldsymbol{x}_i \in \mathbb{R}^q$, $1 \leq i \leq n$, and we assume that $\mathbf{X}$ has full rank $q \leq n$. The columns of $\mathbf{X}$ are labeled by $x_1, \ldots, x_q$ and they denote the components of $\boldsymbol{x} \in \mathbb{R}^q$, which are standardized versions of* (1.2).

The set (the portfolio) of all available cases is denoted by $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^q$.

	x_1	x_2	x_3	x_4	x_5
x_1	1.0000	−0.7484	−0.8173	−0.3074	−0.6690
x_2		1.0000	0.4552	0.6100	0.1531
x_3			1.0000	−0.1076	0.9317
x_4				1.0000	−0.2533
x_5					1.0000

Table 1: Sample correlations between the columns of $\mathbf{X}$, $x_1, \ldots, x_5$ denote the standardized components of (1.2).

We provide the sample correlations between the columns of the design matrix $\mathbf{X}$ (for our example) in Table 1; these are in line with the absolute values provided in the upper right triangle of Figure 4. We observe high (linear) correlations, and the PCA discussed in the next section tries to explain these correlations by finding an optimal coordinate system.

2.2 Principal components

By assumption, the design matrix $\mathbf{X}$ has full rank $q \leq n$. This means that we can find q linearly independent rows (cases $\boldsymbol{x}_i \in \mathcal{X}$) that may serve as a basis that spans the whole space $\mathbb{R}^q$. PCA determines a different basis, namely, it provides an orthonormal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_q \in \mathbb{R}^q$ such that $\boldsymbol{v}_1$ explains the direction of the biggest heterogeneity in $\mathbf{X}$, $\boldsymbol{v}_2$ the direction of the second biggest heterogeneity orthogonal to $\boldsymbol{v}_1$, etc.

2.2.1 Methodology

The first basis/weight vector $\boldsymbol{v}_1 \in \mathbb{R}^q$ is determined by

$$\boldsymbol{v}_1 \;=\; \underset{\|\boldsymbol{w}\|_2=1}{\arg\max} \, \|\mathbf{X}\boldsymbol{w}\|_2^2 \;=\; \underset{\boldsymbol{w}^\top \boldsymbol{w}=1}{\arg\max} \, \left(\boldsymbol{w}^\top \mathbf{X}^\top \mathbf{X} \boldsymbol{w}\right). \tag{2.4}$$

The second basis/weight vector $\boldsymbol{v}_2 \in \mathbb{R}^q$ is determined by

$$\boldsymbol{v}_2 \;=\; \underset{\|\boldsymbol{w}\|_2=1}{\arg\max} \, \|\mathbf{X}\boldsymbol{w}\|_2^2 \qquad \text{subject to } \langle \boldsymbol{v}_1, \boldsymbol{w} \rangle = 0, \tag{2.5}$$

and the j-th basis/weight vector $\boldsymbol{v}_j \in \mathbb{R}^q$ is determined by

$$\boldsymbol{v}_j \;=\; \underset{\|\boldsymbol{w}\|_2=1}{\arg\max} \, \|\mathbf{X}\boldsymbol{w}\|_2^2 \qquad \text{subject to } \langle \boldsymbol{v}_\ell, \boldsymbol{w} \rangle = 0 \text{ for all } 1 \leq \ell \leq j-1. \tag{2.6}$$

Electronic copy available at: https://ssrn.com/abstract=3439358

Optimization problems (2.4)-(2.6) are convex with convex constraints, i.e. they can be solved recursively using the method of Lagrange. By assumption, the symmetric matrix $A = \mathbf{X}^\top \mathbf{X}$ is positive definite, and the orthonormal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_q \in \mathbb{R}^q$ is simply given by the eigenvectors of A (appropriately ordered).
There is a second way of obtaining this orthonormal basis which is associated with dimension reduction, see Section 14.5.1 in Hastie et al. [11]. Namely, there exist an orthogonal matrix $U \in \mathbb{R}^{n\times q}$ (with $U^\top U = \mathbb{1}_q$), an orthogonal matrix $V \in \mathbb{R}^{q\times q}$ (with $V^\top V = \mathbb{1}_q$), and a diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_q) \in \mathbb{R}^{q\times q}$ with *singular values* $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_q \geq 0$ such that we have singular value decomposition (SVD)

$$\mathbf{X} = U\Lambda V^\top. \tag{2.7}$$

This SVD can be found by the algorithm of Golub–Van Loan [10]. The orthogonal matrix U is called *left singular matrix* of $\mathbf{X}$ and the orthogonal matrix V is called *right singular matrix* of $\mathbf{X}$. A simple calculation using decomposition (2.7) shows

$$V^\top \mathbf{X}^\top \mathbf{X} V = V^\top V \Lambda U^\top U \Lambda V^\top V = \Lambda^2 = \mathrm{diag}(\lambda_1^2, \ldots, \lambda_q^2).$$

Therefore, $(\lambda_j^2)_{1\leq j\leq q}$ are the eigenvalues of $A = \mathbf{X}^\top \mathbf{X}$, and the column vectors of V give the orthonormal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_q$ (eigenvectors of A).
The *principal components* are obtained by the column vectors of

$$\mathbf{X}V \;=\; U\Lambda \;=\; U\mathrm{diag}(\lambda_1, \ldots, \lambda_q) \;\in\; \mathbb{R}^{n\times q}. \tag{2.8}$$

This motivates us to define the following matrices of ranks $1 \leq p \leq q$, see (2.7),

$$\mathbf{X}_p \;=\; U\mathrm{diag}(\lambda_1, \ldots, \lambda_p, 0, \ldots, 0)V^\top \;\in\; \mathbb{R}^{n\times q}. \tag{2.9}$$

These rank p matrices $\mathbf{X}_p$ have the property that they minimize the total squared *reconstruction error* (Frobenius norm) relative to $\mathbf{X}$ among all rank p matrices, that is,

$$\mathbf{X}_p \;=\; \underset{B\in\mathbb{R}^{n\times q}}{\arg\min} \|\mathbf{X} - B\|_{\mathrm{F}} \qquad \text{subject to } \mathrm{rank}(B) \leq p, \tag{2.10}$$

where $\|\cdot\|_{\mathrm{F}}$ is the Frobenius norm given by $\|A\|_{\mathrm{F}}^2 = \sum_{i,j} a_{i,j}^2$ for a matrix $A = (a_{i,j})_{i,j}$. In other words, $\mathbf{X}_p$ is the best rank p approximation to $\mathbf{X}$ keeping as much variability as possible of $\mathbf{X}$ through an optimal choice of $p \leq q$ orthonormal basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p$. This also means that we have replaced a q dimensional representation $\boldsymbol{x}_i \in \mathcal{X}$ of the cases by a p dimensional one such that we receive a minimal reconstruction error. This is illustrated in the example in the next section in more detail. We close with remarks.

Remarks.

- We always use q for the dimension of the original features $\boldsymbol{x}_i \in \mathcal{X} \subset \mathbb{R}^q$ and $p \leq q$ for the lower dimensional approximations denoted by $\boldsymbol{y}_i$. We denote $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\} \subset \mathbb{R}^p$; this will be described in detail below, see for example (2.12) for $p = 2$.

- The PCA minimizes a square loss function which is most appropriate in a Gaussian context. If feature components are obviously non-Gaussian they need pre-processing by applying a transformation. For instance, if $x_1^* > 0$ has outliers, we may consider the logarithm of this component before standardizing to x_1, or more generally we may use "Tukey's first aid transformations", see Chapter 4 in Stahel [29].

Electronic copy available at: https://ssrn.com/abstract=3439358

2.2.2 Example, revisited

We come back to the data presented in Listing 1. In Listing 2 we provide the relevant code to perform the PCA in R.

Listing 2: R code for PCA using `svd`

```
dat2 <- dat
dat2$x1 <- log(dat2$weight/dat2$max_power)
dat2$x2 <- log(dat2$max_power/dat2$cubic_capacity)
dat2$x3 <- log(dat2$max_torque)
dat2$x4 <- log(dat2$max_engine_speed)
dat2$x5 <- log(dat2$cubic_capacity)
dat2 <- dat2[, c("x1","x2","x3","x4","x5")]      # raw design matrix

# normalization of raw design matrix
X <- dat2-colMeans(dat2)[col(dat2)]    # centering of columns
X <- X/sqrt(colMeans(X^2))[col(X)]     # normalization of columns

# singular value decomposition
SVD <- svd(as.matrix(X))
SVD$d                                  # singular values
dat2$v1 <- X %*% SVD$v[,1]             # 1st principal components
dat2$v2 <- X %*% SVD$v[,2]             # 2nd principal components
```

Alternatively, we can also use the R command `princomp` instead of `svd`, see Listing 3.[7]

Listing 3: R code for PCA using `princomp`

```
pca <- princomp(as.matrix(X), cor=TRUE)
pca$loadings
summary(pca)
```

principal component p	singular values λ_p	scaled eigenvalues λ_p^2/n	proportion λ_p^2/n of total increment	cumulative
1	37.53	2.966	59%	59%
2	28.07	1.659	33%	92%
3	11.48	0.277	6%	98%
4	6.78	0.088	2%	100%
5	2.12	0.009	0%	100%

Table 2: PCA results from `svd` and `princomp`, respectively.

The singular values λ_p and the scaled eigenvalues λ_p^2/n, $1 \leq p \leq q$, are presented in Table 2. We observe that the first two principal components explain 92% of the total variance,[8] thus, most of the variability in the columns can be explained by the first two basis vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. This is in line with Table 2 of Ingenbleek-Lemaire [14], though we use slightly different data, here. The principal component values of cases $i = 1, \ldots, n$ are obtained by

$$\boldsymbol{x}_i \in \mathbb{R}^q \;\mapsto\; y_{i,j} = \langle \boldsymbol{v}_j, \boldsymbol{x}_i \rangle = v_{j,1} x_{i,1} + \ldots + v_{j,q} x_{i,q}, \tag{2.11}$$

[7] Remark that the R functions `svd` and `princomp` may provide opposite directions for the weight vectors.
[8] In fact, principal components explain correlations and not variances here because we have standardized.

Electronic copy available at: https://ssrn.com/abstract=3439358

for $j = 1, \ldots, q = 5$. We plot these values $(y_{i,j})_{1 \le i \le n, 1 \le j \le q}$ in Figure 7. The diagonal gives

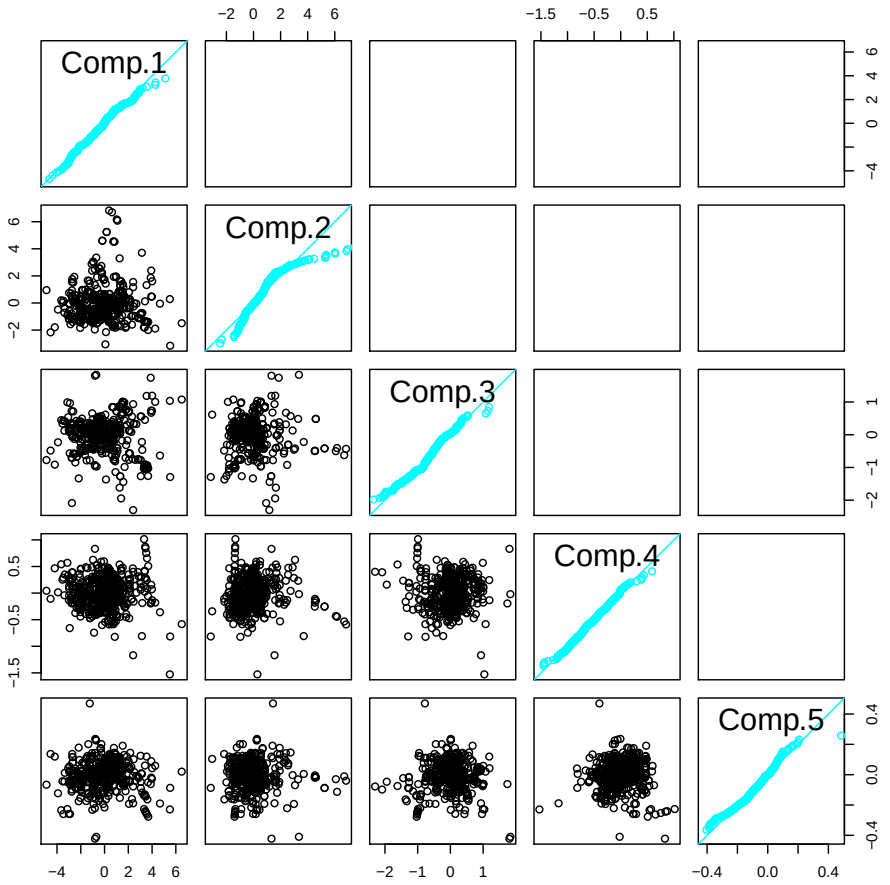

Figure 7: Scatter plots (lower left triangle) and QQ plots (diagonal) of the principal component values of all $n = 475$ cases; the upper right triangle is empty because weight vectors are orthogonal by construction.

the QQ plots against the Gaussian distribution. In particular, the second component values $(y_{i,2})_{1 \le i \le n}$ question a Gaussian assumption. Since these values are uncorrelated (by construction) between components $1 \le j \le q$ we expect a copula similar to independence, see scatter plots in the lower left triangle of Figure 7.

Using the first basis vector $\boldsymbol{v}_1$ we define the rank 1 approximation $\mathbf{X}_1$ to $\mathbf{X}$. This first basis vector defines the regression equation (still in the standardized version without *), see also (2.11),

$$\boldsymbol{x} \in \mathbb{R}^q \;\mapsto\; y_1 = \langle \boldsymbol{v}_1, \boldsymbol{x} \rangle = -0.558x_1 + 0.412x_2 + 0.539x_3 + 0.126x_4 + 0.461x_5.$$

If we transform this back to the original scale (1.3), we need to consider for $1 \le l \le q$

$$\alpha_l x_l^* = \alpha_l \left(\widehat{\sigma}_l \frac{x_l^* - \widehat{\mu}_l}{\widehat{\sigma}_l} + \widehat{\mu}_l \right) = \alpha_l \widehat{\sigma}_l x_l + \alpha_l \widehat{\mu}_l \;\stackrel{!}{=}\; v_{1,l} x_l + \frac{v_{1,l}}{\widehat{\sigma}_l} \widehat{\mu}_l.$$

This provides for $\alpha_l = v_{1,l} / \widehat{\sigma}_l$, see also (1.3),

$$(\alpha_1, \ldots, \alpha_5) = (-1.9423, 1.8107, 1.2703, 1.2341, 1.3165).$$

Electronic copy available at: https://ssrn.com/abstract=3439358

In view of (1.4) this provides (we scale with $1/\alpha_1$)

$$\begin{aligned}\frac{y^*}{\alpha_1} &= \log(\texttt{weight}) - 1.93 \cdot \log(\texttt{max\_power}) - 0.65 \cdot \log(\texttt{max\_torque}) \\ &\quad -0.64 \cdot \log(\texttt{max\_engine\_speed}) + 0.25 \cdot \log(\texttt{cubic\_capacity}).\end{aligned}$$

Comparing this to the expert choice (1.1), the first principal component provides a heavier punishment to `max_power`, exactly the same relief to `cubic_capacity`, and some punishment terms are added on `max_torque` and `max_engine_speed`.
The second principal component value is determined by, see also (2.11),

$$\boldsymbol{x} \in \mathbb{R}^q \;\mapsto\; y_2 = \langle \boldsymbol{v}_2, \boldsymbol{x} \rangle = 0.103x_1 - 0.482x_2 + 0.268x_3 - 0.705x_4 + 0.434x_5.$$

This allows us to illustrate all cars along the first two principal component axis

$$\mathbf{X} \;\mapsto\; (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)^\top \overset{\text{def.}}{=} (\mathbf{X}\boldsymbol{v}_1, \mathbf{X}\boldsymbol{v}_2) \;\in\; \mathbb{R}^{n \times 2}, \tag{2.12}$$

which exactly corresponds to the first two columns of $\mathbf{X}V$, see (2.8), and where we set $\boldsymbol{y}_i = (y_{i,1}, y_{i,2})^\top = (\langle \boldsymbol{v}_1, \boldsymbol{x}_i \rangle, \langle \boldsymbol{v}_2, \boldsymbol{x}_i \rangle)^\top \in \mathcal{Y} \subset \mathbb{R}^{p=2}$, for $i = 1, \ldots, n$. In fact, this exactly corresponds to the first scatter plot in the lower left triangle of Figure 7. In Figure 8 (lhs) we again plot these

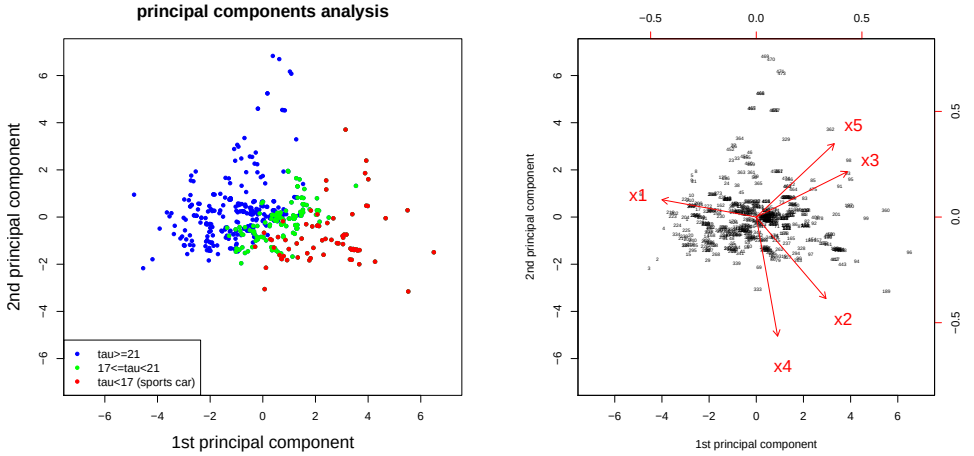

Figure 8: (lhs) First two principal components with red color for sports car $\tau < 17$, green color for cars with $\tau \in [17, 21)$, the remaining cars $\tau \geq 21$ are in blue color; (rhs) biplot of the first two principal component values $\boldsymbol{y}_i$ (on the primary axis) and the loading vectors $(v_{1,\ell}, v_{2,\ell})^\top$ (on the secondary axis).

first two principal components $\boldsymbol{y}_i \in \mathcal{Y} \subset \mathbb{R}^2$ of all cars $i = 1, \ldots, n = 475$, but in different colors. In red color we illustrate the sports car with $\tau < 17$, in green color the cars with $\tau \in [17, 21)$, and in blue color the cars with $\tau \geq 21$. We observe that the first two principal components explain the expert choice of a sports car fairly well because there seems to be a hyperplane separation between the two types of cars.
Importantly (and interestingly), Figure 8 (lhs) provides a dimension reduction from $\mathbb{R}^q$ to $\mathbb{R}^2$ because each car $\boldsymbol{x}_i \in \mathcal{X}$ is represented by a point $\boldsymbol{y}_i = (\boldsymbol{v}_1, \boldsymbol{v}_2)^\top \boldsymbol{x}_i \in \mathcal{Y} \subset \mathbb{R}^{p=2}$ in Figure 8, see (2.12). This is the core topic of this tutorial which is going to be discussed in more detail below.

Electronic copy available at: https://ssrn.com/abstract=3439358

Finally, Figure 8 (rhs) provides a biplot which simultaneously shows the first two principal component values $\boldsymbol{y}_i = (\boldsymbol{v}_1, \boldsymbol{v}_2)^\top \boldsymbol{x}_i \in \mathcal{Y} \subset \mathbb{R}^{p=2}$, $i = 1, \ldots, n$, and the two dimensional loading plot illustrating the vectors

$$(v_{1,\ell}, v_{2,\ell})^\top, \qquad \text{for } \ell = 1, \ldots, q = 5.$$

The low dimensional representations $\boldsymbol{y}_i$ of the cases $i = 1, \ldots, n$ are illustrated by black dots in Figure 8 (rhs), and proximity between black dots means that the corresponding cases are similar. The loading vectors $(v_{1,\ell}, v_{2,\ell})^\top$, $\ell = 1, \ldots, q$, are illustrated by red vectors in Figure 8 (rhs). Note that these are the components of the first two orthonormal weight vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. The lengths of these red vectors reflect standard deviations along the variables, and the cosines of the angles between the red vectors give the corresponding correlations. The projections of the cases $\boldsymbol{y}_i$ on the axes of the loading vectors $(v_{1,\ell}, v_{2,\ell})^\top$ approximate the original cases $\boldsymbol{x}_i$. In fact, this is exact for $q = 2$ and an approximation for $q > 2$, for more interpretation we refer to Section 7.2 of Stahel [29].

Remarks.

- PCA is sensitive to outliers, and there also exist robust PCA versions. For instance, Croux et al. [3] give an algorithm that is based on median absolute deviations (MADs), we also refer to the R package `pcaPP`. For general autoencoders we refer to the next section.
- Standardization considers translation and scaling of cases to the same origin and the same units. SVD then tries to find an optimal rotation of the standardized data to a new orthonormal basis.

3 Autoencoders

The PCA introduced above can be seen as an autoencoder. In this section we introduce autoencoders in more generality, and we provide a bottleneck neural network (BNN) which is a second example of an autoencoder.

3.1 Introduction to autoencoders

3.1.1 Methodology

An autoencoder consist of two mappings

$$\varphi : \mathbb{R}^q \to \mathbb{R}^p \qquad \text{and} \qquad \psi : \mathbb{R}^p \to \mathbb{R}^q, \tag{3.1}$$

with dimensions $p \leq q$, therefore, an autoencoder typically leads to a loss of information. Choose a *dissimilarity function* $d(\cdot, \cdot) : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}_+$ being zero if and only if the two arguments in $d(\cdot, \cdot)$ are identical, that is, $d(\boldsymbol{x}', \boldsymbol{x}) = 0$ if and only if $\boldsymbol{x}' = \boldsymbol{x}$. An *autoencoder* is a pair (φ, ψ) of mappings (3.1) such that their composition $\pi = \psi \circ \varphi$ leads to a *small reconstruction error* w.r.t. the dissimilarity function $d(\cdot, \cdot)$, that is,

$$\boldsymbol{x} \mapsto d(\pi(\boldsymbol{x}), \boldsymbol{x}) \quad \text{is small.} \tag{3.2}$$

Electronic copy available at: https://ssrn.com/abstract=3439358

The mapping φ is then called *encoder*, the mapping ψ is called *decoder*, and $\boldsymbol{y} = \varphi(\boldsymbol{x}) \in \mathbb{R}^p$ is a p dimensional representation of $\boldsymbol{x} \in \mathbb{R}^q$ which contains all information of $\boldsymbol{x}$ up to some reconstruction error that is small (3.2).

Remark. In (3.2) we try to find mappings such that we have a small reconstruction error between the data $\boldsymbol{x}_i$ and its representation $\pi(\boldsymbol{x}_i)$ for $i = 1, \ldots, n$. We can think of $(\boldsymbol{x}_i)_{1 \le i \le n}$ and $(\pi(\boldsymbol{x}_i))_{1 \le i \le n}$ describing two similar shapes. If we restrict ourselves to finding optimal translations, scalings and rotations such that the shapes of two objects can be superimposed, then we are in the field of procrustes analysis (we will encounter this again later). This is implicitly done by an autoencoder (possibly in a non-linear fashion) if we minimize the reconstruction error.

The PCA of the previous section provides a first example of an autoencoder. Assume we would like to have a small reconstruction error for all covariates $\boldsymbol{x}_i \in \mathcal{X}$. We choose as dissimilarity function the squared Euclidean distance on $\mathbb{R}^q$, i.e. $d(\boldsymbol{x}', \boldsymbol{x}) = \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2 = \sum_{j=1}^q (x'_j - x_j)^2$. This results in the squared Frobenius norm for matrix differences if we consider all cases simultaneously, that is, $\|\mathbf{X}' - \mathbf{X}\|_{\mathrm{F}}^2 = \sum_{i=1}^n \|\boldsymbol{x}'_i - \boldsymbol{x}_i\|_2^2$.
The encoder φ is received from the orthonormal basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_q$ by setting, see (2.11),

$$\varphi : \mathbb{R}^q \to \mathbb{R}^p, \qquad \boldsymbol{x} \mapsto \boldsymbol{y} = \varphi(\boldsymbol{x}) = (\langle \boldsymbol{v}_1, \boldsymbol{x} \rangle, \ldots, \langle \boldsymbol{v}_p, \boldsymbol{x} \rangle)^\top = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)^\top \boldsymbol{x}.$$

This corresponds to the first p columns in (2.8) if we insert $\mathbf{X}^\top \in \mathbb{R}^{q \times n}$ for $\boldsymbol{x} \in \mathbb{R}^q$. For $p = 2$ this exactly corresponds to the 2 dimensional representation of Figure 8 (lhs) in our example.
The decoder is given by

$$\psi : \mathbb{R}^p \to \mathbb{R}^q, \qquad \boldsymbol{y} \mapsto \psi(\boldsymbol{y}) = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)\boldsymbol{y}.$$

This implies column-wise in $\mathbf{X}^\top$

$$\pi(\mathbf{X}^\top) \;=\; \psi \circ \varphi(\mathbf{X}^\top) \;=\; (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p)^\top \mathbf{X}^\top \;=\; \mathbf{X}_p^\top.$$

Thus, we have for the squared Euclidean distance on $\mathbb{R}^q$ a total reconstruction error of

$$\sum_{i=1}^n d(\pi(\boldsymbol{x}_i), \boldsymbol{x}_i) = \sum_{i=1}^n \|\pi(\boldsymbol{x}_i) - \boldsymbol{x}_i\|_2^2 = \|\mathbf{X}_p - \mathbf{X}\|_{\mathrm{F}}^2. \tag{3.3}$$

3.1.2 Example, revisited: PCA as autoencoder

In Table 3 we provide the scaled reconstruction errors of the PCA for an increasing number p of considered principal components. For $p = 2$ principal components we receive a reconstruction error of 0.6124.

p	1	2	3	4	5
scaled reconstruction error $\|\mathbf{X}_p - \mathbf{X}\|_{\mathrm{F}}/\sqrt{n}$	1.4263	0.6124	0.3128	0.0974	0.0000

Table 3: Scaled Frobenius norm reconstruction error of the PCA for $1 \le p \le 5$.

In Figure 9 we illustrate all reconstructed values $\pi_j(\boldsymbol{x}_i)$ on the y-axis against the original values $x_{i,j}$ on the x-axis (for components $j = 1, \ldots, 5$ from left to right); these plots can also be obtained

Electronic copy available at: https://ssrn.com/abstract=3439358

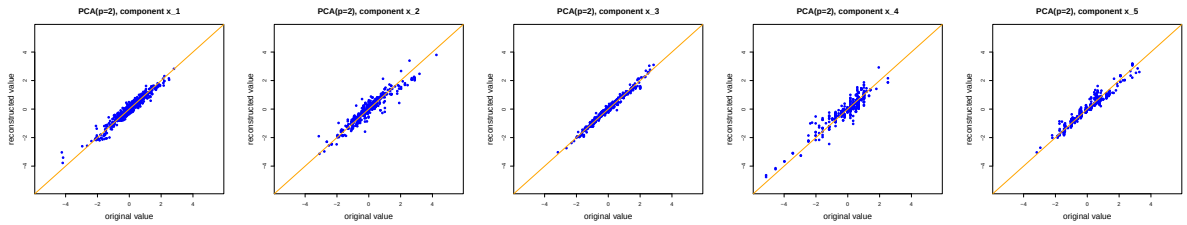

Figure 9: Reconstruction of original variables using a PCA with $p = 2$ principal components for the variables $x_1, \ldots, x_5$ (from left to right).

from the biplot on the right-hand side of Figure 8 by projecting $\boldsymbol{y}_i \in \mathbb{R}^2$ onto the corresponding loading vectors $(v_{1,\ell}, v_{2,\ell})^\top$, $1 \le \ell \le q = 5$. If the reconstruction would be perfect, then all these points would lie on the orange diagonal line. We observe that the reconstruction with PCA $p = 2$ works rather well, the most difficult component seems x_4 which is the most non-Gaussian one, see Figure 5. This finishes this example.

The PCA is optimal if we consider linear approximations under the squared Euclidean distance reconstruction error (3.3). Of course, this is not appropriate in any circumstances as seen above, therefore we present other methods below.

3.2 Bottleneck neural network

3.2.1 Methodology

As an example of a non-linear autoencoder we consider a bottleneck neural network (BNN). BNNs have been introduced and studied by Kramer [22], Hinton–Salakhutdinov [12], and other researchers. BNNs are a special type of feed-forward neural networks. For a general discussion of feed-forward neural networks we refer to our previous tutorials [7, 27], as well as to Chapter 5 in the lecture notes of Wüthrich–Buser [32].

In the following we use the notation of our previous tutorials. To successfully calibrate a BNN it should have an odd number d of hidden layers (d is called the depth of the neural network). The central hidden layer should be low dimensional, having $q_{(d+1)/2} = p < q$ hidden neurons, and all remaining hidden layers should be symmetric around this central hidden layer in the sense that their numbers of hidden neurons satisfy $q_{(d+1)/2-k} = q_{(d+1)/2+k}$ for all $k = 1, \ldots, (d+1)/2 - 1$. The output dimension should be equal to the input dimension, i.e. $q_{d+1} = q_0 = q$. Thus, for a BNN of depth $d = 3$ we may choose the following numbers of neurons

$$(q_0, q_1, q_2, q_3, q_4) = (5, 7, 2, 7, 5), \tag{3.4}$$

such a BNN is illustrated in Figure 10. We describe below why we prefer a neural network architecture that is symmetric around the central (bottleneck) layer which has $q_{(d+1)/2} = p = 2$ hidden neurons in example (3.4). Note that the input and output layers in (3.4) have dimension $q_0 = q_4 = q = 5$ which is naturally given in our example of Section 2.2.2 because $\boldsymbol{x}_i \in \mathbb{R}^{q=5}$. The bottleneck has dimension $q_2 = p = 2$, and the symmetric hidden layers around the bottleneck have dimension $q_1 = q_3 = 7$, see Figure 10.

Electronic copy available at: https://ssrn.com/abstract=3439358

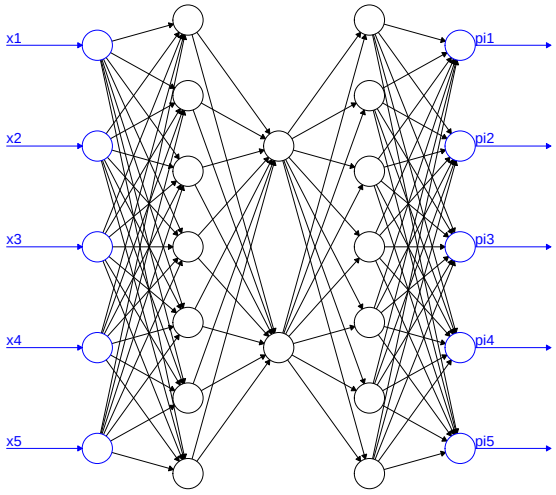

Figure 10: BNN with $(q_0, q_1, q_2, q_3, q_4) = (5, 7, 2, 7, 5)$, input $\boldsymbol{x} \in \mathbb{R}^5$ and output $\pi = \pi(\boldsymbol{x}) \in \mathbb{R}^5$ (both in blue color), and the black circles are the hidden neurons.

More formally, we choose a neural network architecture that has the following structure

$$\pi : \mathbb{R}^{q_0} \to \mathbb{R}^{q_{d+1}=q_0}, \qquad \boldsymbol{x} \mapsto \pi(\boldsymbol{x}) = \left(\boldsymbol{z}^{(d+1)} \circ \boldsymbol{z}^{(d)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}),$$

with hidden neural network layers for $1 \leq m \leq d$ given by

$$\boldsymbol{z}^{(m)} : \mathbb{R}^{q_{m-1}} \to \mathbb{R}^{q_m}, \qquad \boldsymbol{z} \mapsto \boldsymbol{z}^{(m)}(\boldsymbol{z}) = \left(\phi(\langle \boldsymbol{w}_1^{(m)}, \boldsymbol{z}\rangle), \ldots, \phi(\langle \boldsymbol{w}_{q_m}^{(m)}, \boldsymbol{z}\rangle)\right)^\top,$$

with neural network weights $\boldsymbol{w}_l^{(m)} \in \mathbb{R}^{q_{m-1}}$, $1 \leq l \leq q_m$, and with activation function $\phi : \mathbb{R} \to \mathbb{R}$. Finally, the output layer is chosen as

$$\boldsymbol{z}^{(d+1)} : \mathbb{R}^{q_d} \to \mathbb{R}^{q_{d+1}}, \qquad \boldsymbol{z} \mapsto \boldsymbol{z}^{(d+1)}(\boldsymbol{z}) = \left(\langle \boldsymbol{w}_1^{(d+1)}, \boldsymbol{z}\rangle, \ldots, \langle \boldsymbol{w}_{q_{d+1}}^{(d+1)}, \boldsymbol{z}\rangle\right)^\top,$$

with neural network weights $\boldsymbol{w}_l^{(d+1)} \in \mathbb{R}^{q_d}$, $1 \leq l \leq q_{d+1}$, and with linear output activation function. This BNN has a network parameter $\theta = (w_1^{(1)}, \ldots, w_{q_{d+1}}^{(d+1)}) \in \mathbb{R}^r$ of dimension $r = \sum_{m=1}^{d+1} q_m q_{m-1}$. The BNN in Figure 10 has a network parameter of dimension $r = 98$.

Remarks.

- In contrast to classical feed-forward neural networks we do not include an intercept here, because the features $\boldsymbol{x}_i \in \mathcal{X}$ have been standardized. This slightly reduces the dimension of the network parameter compared to classical feed-forward neural networks.
- The output activation has been chosen to be the linear one because all components of $\boldsymbol{x}$ live in $\mathbb{R}$. Below, we will also meet other output activation functions.

Electronic copy available at: https://ssrn.com/abstract=3439358

- As activation function ϕ for the hidden layers we typically choose the hyperbolic tangent function. If we would choose the linear activation function for ϕ, we would be back in the PCA case, i.e., a BNN having only linear activation functions gives a PCA.

The *BNN encoder* is given by (neuron activation at the central hidden layer)

$$\varphi : \mathbb{R}^{q_0=q} \to \mathbb{R}^{q_{(d+1)/2}=p}, \qquad \boldsymbol{x} \mapsto \boldsymbol{y} = \varphi(\boldsymbol{x}) = \left(\boldsymbol{z}^{((d+1)/2)} \circ \cdots \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}),$$

and the *BNN decoder* is given by

$$\psi : \mathbb{R}^{q_{(d+1)/2}=p} \to \mathbb{R}^{q_{d+1}=q}, \qquad \boldsymbol{y} \mapsto \psi(\boldsymbol{y}) = \left(\boldsymbol{z}^{(d+1)} \circ \cdots \circ \boldsymbol{z}^{((d+1)/2+1)}\right)(\boldsymbol{y}).$$

We implement this BNN using the `keras` library in R. In Listing 4 we illustrate the corresponding

Listing 4: BNN implementation using the `keras` library of R

```
library(keras)

Input <- layer_input(shape = c(5), dtype = 'float32', name = 'Input')

Encoder = Input %>%
          layer_dense(units=7, activation='tanh',   use_bias=FALSE, name='Layer1') %>%
          layer_dense(units=2, activation='tanh',   use_bias=FALSE, name='Bottleneck')

Decoder = Encoder %>%
          layer_dense(units=7, activation='tanh',   use_bias=FALSE, name='Layer3') %>%
          layer_dense(units=5, activation='linear', use_bias=FALSE, name='Output')

model <- keras_model(inputs = Input, outputs = Decoder)
model %>% compile(optimizer = optimizer_nadam(), loss = 'mean_squared_error')
```

R code using the architecture illustrated in Figure 10, see also (3.4), with hyperbolic tangent activation function for ϕ. This results in a BNN having $r = 98$ parameters, see Listing 5.

Listing 5: BNN architecture of Figure 10

```
Layer (type)                 Output Shape              Param #
=================================================================
Input (InputLayer)           (None, 5)                 0
_________________________________________________________________
Layer1 (Dense)               (None, 7)                 35
_________________________________________________________________
Bottleneck (Dense)           (None, 2)                 14
_________________________________________________________________
Layer3 (Dense)               (None, 7)                 14
_________________________________________________________________
Output (Dense)               (None, 5)                 35
=================================================================
Total params: 98
Trainable params: 98
Non-trainable params: 0
```

This BNN could now be trained brute force by applying the gradient descent algorithm. In our problem this would work well because it is a low dimensional problem, however, in high dimensional situations it often does not work properly. Therefore, we follow the proposal of Hinton–Salakhutdinov [12] for model training.

Electronic copy available at: https://ssrn.com/abstract=3439358

3.2.2 Bottleneck neural network calibration

Hinton–Salakhutdinov [12] have suggested to use symmetric feed-forward neural network architectures around the bottleneck layer because these architectures can nicely be pre-trained. Since $q_{(d+1)/2-k} = q_{(d+1)/2+k}$ for all $k = 1, \ldots, (d+1)/2 - 1$, we can collapse all hidden layers in between the two layers $(d+1)/2 - k$ and $(d+1)/2 + k$ by merging these two layers to a new central layer. This is illustrated in Figure 11. The graph on the left-hand side shows the full

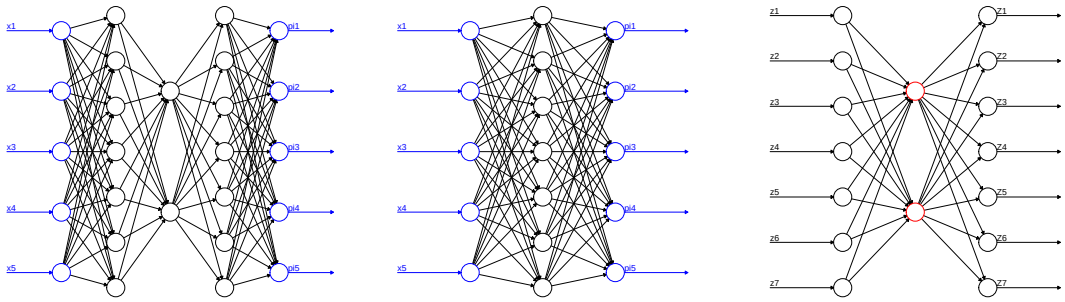

Figure 11: (lhs) full BNN, (middle) hidden layers 1 and 3 merged to the new central layer, (rhs) remaining inner BNN.

BNN architecture of depth $d = 3$ with $(q_0, q_1, q_2, q_3, q_4) = (5, 7, 2, 7, 5)$. The middle graph shows this BNN where we have merged layers 1 and 3 with $q_1 = q_3 = 7$ to the new central layer, i.e. we have a network of depth 1 with $(q_0, q_1, q_4) = (5, 7, 5)$. The graph on the right-hand side shows the collapsed part of the BNN. Training then includes the following steps:

- In a first step we train the new neural network (middle graph of Figure 11). This provides pre-trained weights $\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_{q_1=7}^{(1)} \in \mathbb{R}^{q_0=5}$ and $\boldsymbol{w}_1^{(d+1)}, \ldots, \boldsymbol{w}_{q_{d+1}=5}^{(d+1)} \in \mathbb{R}^{q_d=7}$ for the full BNN. This corresponds to "pre-training 1: outer part" on lines 13-14 of Listing 6.

- From this merged and trained neural network (middle graph of Figure 11) we read off the neuron activations $\boldsymbol{z}_i = \boldsymbol{z}^{(1)}(\boldsymbol{x}_i) = (z_1^{(1)}(\boldsymbol{x}_i), \ldots, z_{q_1}^{(1)}(\boldsymbol{x}_i))^\top \in \mathbb{R}^{q_1=q_d}$ in the central layer for all cases $i = 1, \ldots, n$. This corresponds to lines 17-18 of Listing 6.

- These neuron activations $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n \in \mathbb{R}^{q_1=q_d}$ are used to pre-train the inner part of the BNN which is illustrated on the right-hand side of Figure 11. Therefore, we again try to minimize the reconstruction error between the inputs $\boldsymbol{z}_i$ and the resulting outputs. This provides pre-trained weights $\boldsymbol{w}_1^{(2)}, \ldots, \boldsymbol{w}_{q_2=2}^{(2)} \in \mathbb{R}^{q_1=7}$ and $\boldsymbol{w}_1^{(3)}, \ldots, \boldsymbol{w}_{q_3=7}^{(3)} \in \mathbb{R}^{q_2=2}$ for the full BNN. This corresponds to "pre-training 2: inner part" on lines 21-22 of Listing 6.

- All these pre-trained weights are then used as starting weights for the calibration of the full BNN using the gradient descent algorithm given in Listing 4.

3.2.3 Example, revisited

We come back to the data presented in Listing 1. We train the BNN illustrated in Listings 4 and 5. The neuron activations at the bottleneck (using the encoder) $\boldsymbol{y}_i = \varphi(\boldsymbol{x}_i) = \left(\boldsymbol{z}^{(2)} \circ \boldsymbol{z}^{(1)}\right)(\boldsymbol{x}_i)$ then provide the $q_2 = p = 2$ dimensional representation of $\boldsymbol{x}_i \in \mathbb{R}^{q=5}$.

Electronic copy available at: https://ssrn.com/abstract=3439358

Listing 6: pre-training of weights in BNNs

```
1  # definition of a 1 hidden layer BNN
2  bottleneck.1 <- function(q0, q1){
3     Input <- layer_input(shape = c(q0), dtype = 'float32', name = 'Input')
4     Output = Input %>%
5         layer_dense(units=q1, activation='tanh',   use_bias=FALSE, name='Bottleneck') %>%
6         layer_dense(units=q0, activation='linear', use_bias=FALSE, name='Output')
7     model <- keras_model(inputs = Input, outputs = Output)
8     model %>% compile(optimizer = optimizer_nadam(), loss = 'mean_squared_error')
9     model
10    }
11
12 # pre-training 1: outer part
13 model.1 <- bottleneck.1(5,7)
14 fit <- model.1 %>% fit(as.matrix(X), as.matrix(X), epochs=2000, batch_size=nrow(X))
15
16 # neuron activations in the central layer
17 zz <- keras_model(inputs=model.1$input, outputs=get_layer(model.1,'Bottleneck')$output)
18 yy <- zz %>% predict(as.matrix(X))
19
20 # pre-training 2: inner part
21 model.2 <- bottleneck.1(7,2)
22 fit <- model.2 %>% fit(as.matrix(yy), as.matrix(yy), epochs=2000, batch_size=nrow(yy))
23
24 # get pre-trained weights
25 get_weights(model.1)
26 get_weights(model.2)
```

For training the BNN of Listing 4 we pre-train the weights using the methodology described above by collapsing some of the hidden layers. The R code used for this pre-training is provided in Listing 6. On lines 12-14 the outer part is pre-trained, and on lines 20-22 the inner part is pre-trained. On lines 24-26 we extract the pre-trained weights. Using these pre-trained weights for the full BNN we receive a reconstruction error $\|(\psi \circ \varphi)(\mathbf{X}) - \mathbf{X}\|_{\mathrm{F}}/\sqrt{n}$ of 0.7968, which is worse than the one of the PCA with $p = 2$ principal components, see Table 3.

Using these pre-trained weights as initialization, the gradient descent algorithm is applied to the full BNN of Listing 4 to obtain a BNN dimension reduction. In Figure 12 we illustrate the decrease of the Frobenius loss function over 10'000 training epochs.[9] We observe that after roughly 2'000 epochs the loss falls below the one of the PCA with $p = 2$ principal components (orange line in Figure 12 at level 0.6124), thus, we receive a smaller reconstruction error in this BNN with $q_2 = p = 2$ bottleneck neurons. The final reconstruction error after 10'000 epochs is 0.5611.[10]

In Figure 13 we illustrate the PCA and the BNN dimension reductions to $p = q_2 = 2$ dimensions. We see very similar results, the BNN dimension reduction seems to be a slightly rotated and scaled version of the PCA solution. We could compare the two shapes of the low dimensional representations in more depth, as mentioned this may include a translation, a scaling and a rotation of, say, the bottleneck representation such that the two shapes can be superimposed. Again, procrustes analysis maybe useful for this task, but we refrain here from going into more detail.[11]

[9] The total run-time of 10'000 training epochs is 25 seconds.

[10] Note that we do not track over-fitting here.

[11] The R package `MCMCpack` has a `procrustes` function.

Electronic copy available at: https://ssrn.com/abstract=3439358

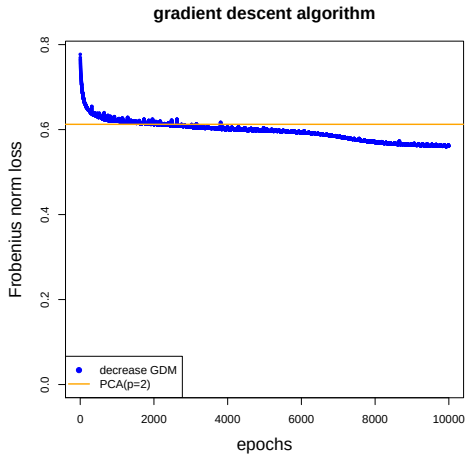

Figure 12: Decrease of Frobenius loss function in the gradient descent algorithm over 10'000 epochs, and pre-trained weights as starting value.

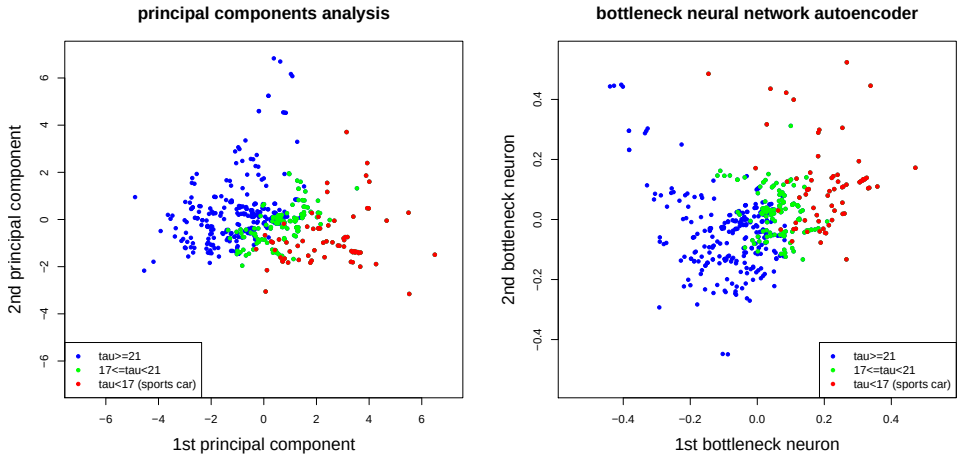

Figure 13: PCA dimension reduction, see Figure 8, and BNN autoencoder dimension reduction for $p = 2$ illustrating the resulting $\boldsymbol{y}_i \in \mathcal{Y}$.

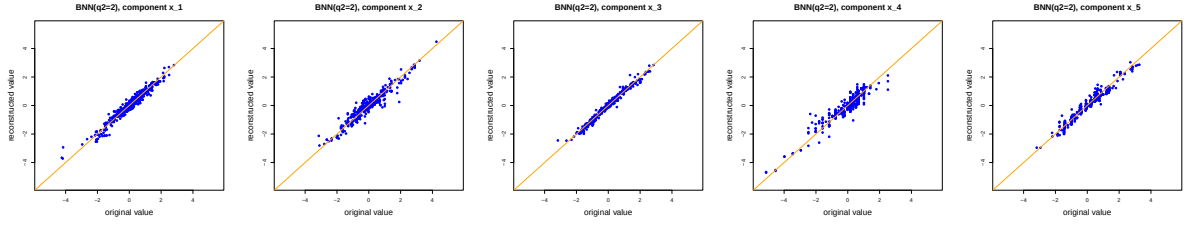

Figure 14: Reconstruction of original variables using a BNN with $q_2 = 2$ bottleneck neurons for the variables $x_1, \ldots, x_5$ (from left to right).

Electronic copy available at: https://ssrn.com/abstract=3439358

In Figure 14 we illustrate the BNN reconstructions over all components $1 \leq j \leq q$. This should be compared to the PCA reconstructions given in Figure 9. The BNN results are slightly better, though only hardly visible.

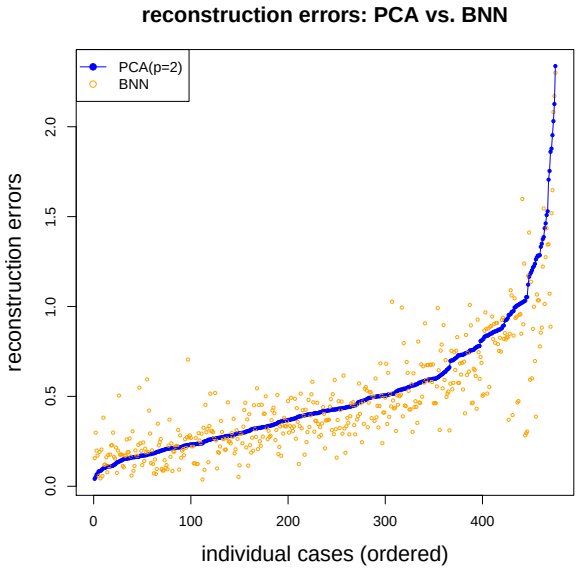

Figure 15: Reconstruction errors on individual cases: PCA($p = 2$) in blue color vs. BNN reduction in orange color, individual cases are ordered w.r.t. PCA reconstruction errors.

In Figure 15 we compare the reconstruction errors on individual cases for the PCA dimension reduction technique and the BNN dimension reduction technique to two dimensional representations $\boldsymbol{y}_i \in \mathcal{Y} \subset \mathbb{R}^2$. From this graph we conclude that both methods come to similar results, also on individual cases, but in general, the reconstruction error is smaller for the BNN. On few cases the BNN technique leads to clearly better reconstruction results (orange dots in the lower right corner). The main conclusion of this example is that for low dimensional problems the PCA is (not surprisingly) often sufficient because non-linearities do not play a crucial role.

3.3 Loss functions

For the PCA the natural loss function to calculate the reconstruction error is the Frobenius norm

$$\|\mathbf{X}_p - \mathbf{X}\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{n} \|\pi(\boldsymbol{x}_i) - \boldsymbol{x}_i\|_2^2}.$$

For comparability reasons we choose the mean squared error loss function for the BNN calibration, see line 14 in Listing 4. This loss function scales the Frobenius norm with $(nq)^{-1}$ which is a constant for a given number n of cases, i.e. the same objective function is minimized in the PCA and in the BNN of Listing 4.
However, the BNN technique allows for much more flexibility in the choice of the loss function. We may, for instance, replace the Euclidean dissimilarity function by any other norm, that is,

Electronic copy available at: https://ssrn.com/abstract=3439358

we may choose, say, the L^1-norm (Manhattan distance)

$$\boldsymbol{x} \;\mapsto\; d(\pi(\boldsymbol{x}), \boldsymbol{x}) = \|\pi(\boldsymbol{x}) - \boldsymbol{x}\|_1.$$

In the implementation, this only requires that the `mean_squared_error` on line 14 of Listing 4 is replaced by the corresponding dissimilarity measure.[12]

Another point that is worth mentioning is that we perform unsupervised learning here, and implicitly we give the same importance to all components of $\boldsymbol{x}$. However, if dimension reduction is done as data pre-processing for a subsequent regression analysis for insurance pricing, it might be that more attention should be given to specific components of $\boldsymbol{x}$. In this case we may consider a dissimilarity function

$$\boldsymbol{x} \;\mapsto\; d(\pi(\boldsymbol{x}), \boldsymbol{x}) = \sum_{j=1}^{q} \omega_j \left|\pi_j(\boldsymbol{x}) - x_j\right|^2,$$

for non-negative weights $\omega_j > 0$.

A different important example is obtained if $\boldsymbol{x}$ is a discrete probability measure belonging the $(q-1)$-unit simplex

$$\mathcal{S}_q = \left\{ \boldsymbol{x} \in \mathbb{R}^q;\; x_j \geq 0 \text{ and } \sum_{j=1}^{q} x_j = 1 \right\} \;\subset\; \mathbb{R}^q. \tag{3.5}$$

In this case we would design a BNN autoencoder

$$\varphi : \mathcal{S}_q \to \mathbb{R}^p \qquad \text{and} \qquad \psi : \mathbb{R}^p \to \mathcal{S}_q.$$

The dissimilarity function at hand for this kind of problem is the Kullback–Leibler (KL) divergence given by

$$\boldsymbol{x} \;\mapsto\; d(\pi(\boldsymbol{x}), \boldsymbol{x}) = D_{\mathrm{KL}}(\pi(\boldsymbol{x}) || \boldsymbol{x}) = \sum_{j=1}^{q} \pi_j(\boldsymbol{x}) \log\left(\frac{\pi_j(\boldsymbol{x})}{x_j}\right). \tag{3.6}$$

Note that the KL divergence is not a metric because it is neither symmetric nor does it satisfy the triangle inequality. In the probability measure case one should also replace the linear output function, see line 11 of Listing 4, by the softmax output function (generalized logistic function) to guarantee that the output $\pi(\boldsymbol{x})$ lies in the unit simplex $\mathcal{S}_q$. We will come back to this example in the subsequent chapters.

4 Clustering

The previous methods have aimed at reducing the dimension of the data by projecting the feature space to a lower dimensional space such that the original data can be reconstructed sufficiently well. These projections have led to continuous low dimensional representations $\mathcal{Y} \subset \mathbb{R}^p$ of the

[12] We refer to the `Keras` documentation for further available dissimilarity measures such as the `mean_absolute_error` (Manhattan distance).

Electronic copy available at: https://ssrn.com/abstract=3439358

data $\mathcal{X} \subset \mathbb{R}^q$. In the present section we do not focus on having good reconstruction properties, but we rather aim at partitioning the data $\mathcal{X}$ into K clusters such that the resulting clusters are homogeneous. The latter is measured using again a dissimilarity measure which we try to minimize simultaneously on all clusters.

4.1 Types of clusterings

There are different types of clustering methods. One distinguishes between (i) hierarchical clustering, (ii) centroid-based clustering and (iii) distribution-based clustering. There are two types of hierarchical clusterings. The bottom-up clustering[13] algorithm merges recursively similar clusters to bigger clusters until, eventually, the algorithm is stopped. That is, this algorithm groups clusters tree-like into a next higher level. A top-down clustering algorithm acts differently in that it splits recursively (and tree-like) the entire portfolio into smaller (more homogeneous) sub-groups. We will not study hierarchical clustering methods in this tutorial, but we refer to Section 14.3.12 in Hastie et al. [11], and we mention that an advantage of hierarchical clustering is that it does not require pre-specification of the numbers of clusters wanted.
Centroid-based and distribution-based clustering methods need to pre-specify the number of clusters wanted. We describe these two types of clustering methods in the following sections.

4.2 K-means clustering

4.2.1 Methodology and algorithm

K-means clustering is a centroid-based clustering that partitions the n cases $\boldsymbol{x}_i \in \mathcal{X} \subset \mathbb{R}^q$ into K disjoint clusters based on a hyperparameter K. This clustering is described by a classifier

$$\mathcal{C}_K : \mathbb{R}^q \to \mathcal{K} = \{1, \ldots, K\}, \qquad \boldsymbol{x} \mapsto \mathcal{C}_K(\boldsymbol{x}),$$

that gives us a partition $(C_1, \ldots, C_K)$ of $\mathbb{R}^q$ by defining for all $k \in \mathcal{K}$ the clusters

$$C_k = \{\boldsymbol{x} \in \mathbb{R}^q;\ \mathcal{C}_K(\boldsymbol{x}) = k\}.$$

For an illustration see Figure 16. Note that a partition $(C_1, \ldots, C_K)$ of $\mathbb{R}^q$ satisfies

$$\bigcup_{k=1}^{K} C_k = \mathbb{R}^q \qquad \text{and} \qquad C_k \cap C_l = \emptyset \quad \text{for all } k \neq l.$$

The classifier $\mathcal{C}_K$ is chosen such that we have minimal dissimilarity within all clusters C_k. As dissimilarity function we choose the squared Euclidean distance in $\mathbb{R}^q$, that is, for $\boldsymbol{x}', \boldsymbol{x} \in \mathbb{R}^q$ we set

$$d(\boldsymbol{x}', \boldsymbol{x}) = \|\boldsymbol{x}' - \boldsymbol{x}\|_2^2.$$

The reason for this choice will become clear later. The K-means clustering is then obtained by minimizing the following objective function

$$\underset{(C_1,\ldots,C_K)}{\arg\min} \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in C_k \cap \mathcal{X}} d(\boldsymbol{\mu}_k, \boldsymbol{x}_i) \;=\; \underset{(C_1,\ldots,C_K)}{\arg\min} \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i \in C_k \cap \mathcal{X}} \|\boldsymbol{\mu}_k - \boldsymbol{x}_i\|_2^2, \tag{4.1}$$

[13] Bottom-up clustering is also known as agglomerative clustering and single-linkage clustering.

Electronic copy available at: https://ssrn.com/abstract=3439358

where $C = (C_1, \ldots, C_K)$ is a partition of $\mathbb{R}^q$, where $\boldsymbol{x}_i \in C_k \cap \mathcal{X}$ runs over all data $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ that lie in C_k, and where $\boldsymbol{\mu}_k$ is the sample mean over all $\boldsymbol{x}_i \in C_k \cap \mathcal{X}$, i.e.

$$\boldsymbol{\mu}_k = \frac{1}{|\{\boldsymbol{x}_i \in C_k \cap \mathcal{X}\}|} \sum_{\boldsymbol{x}_i \in C_k \cap \mathcal{X}} \boldsymbol{x}_i \ \in \ \mathbb{R}^q. \tag{4.2}$$

We give some interpretations. The last term in (4.1), given by

$$D(C_k, \boldsymbol{\mu}_k) = \sum_{\boldsymbol{x}_i \in C_k \cap \mathcal{X}} \|\boldsymbol{\mu}_k - \boldsymbol{x}_i\|_2^2,$$

is the *within-cluster dissimilarity* on cluster C_k. Thereby, we consider all cases $\boldsymbol{x}_i \in C_k \cap \mathcal{X}$, and $\boldsymbol{\mu}_k \in \mathbb{R}^q$ is the sample mean on C_k. This sample mean minimizes the within-cluster dissimilarity on C_k relative to that *cluster center* (centroid), that is,

$$\boldsymbol{\mu}_k \ = \ \arg\min_{\boldsymbol{\mu} \in \mathbb{R}^q} \sum_{\boldsymbol{x}_i \in C_k \cap \mathcal{X}} \|\boldsymbol{\mu} - \boldsymbol{x}_i\|_2^2 \ = \ \arg\min_{\boldsymbol{\mu} \in \mathbb{R}^q} D(C_k, \boldsymbol{\mu}). \tag{4.3}$$

This is the reason for choosing the squared Euclidean distance as dissimilarity measure, and it gives the name *K-means* to this method. Optimization (4.1) then aims at minimizing the *total within-cluster dissimilarity*, after already having performed (4.3), thus, we determine

$$\arg\min_{(C_1, \ldots, C_K)} \sum_{k=1}^{K} D(C_k, \boldsymbol{\mu}_k).$$

The resulting optimal partition $(C_1, \ldots, C_K)$ of $\mathbb{R}^q$ decomposes the total space $\mathbb{R}^q$ into so-called Voronoi cells. This is illustrated in Figure 16 (lhs) for a synthetic data set for $q = 2$ dimensional

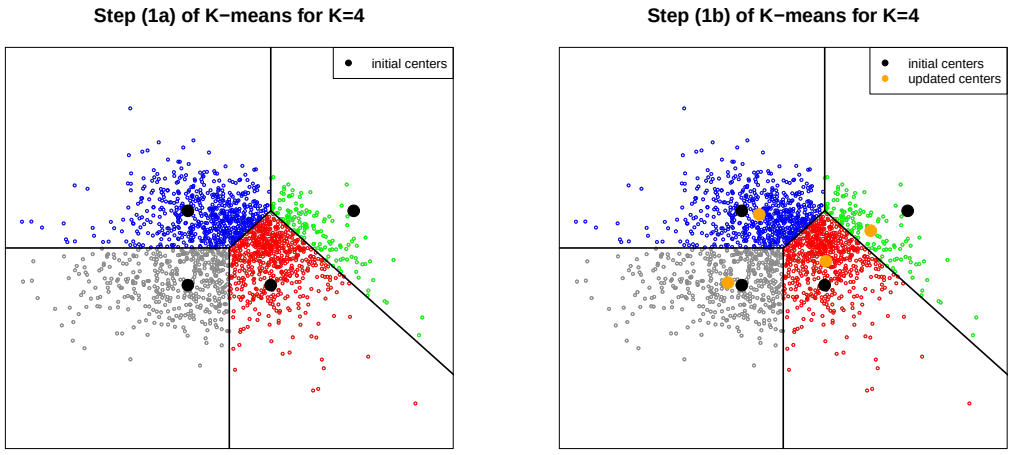

Figure 16: Illustration of K-means clustering for $\mathcal{C}_K : \mathbb{R}^2 \to \mathcal{K} = \{1, \ldots, K = 4\}$: black dots illustrate the cluster centers of step (1a) of the K-means clustering algorithm, and orange dots step (1b).

features $\boldsymbol{x}_i \in \mathbb{R}^2$ and $K = 4$ clusters $C_1, \ldots, C_4$ (gray, blue, green and red). The cluster centers

Electronic copy available at: https://ssrn.com/abstract=3439358

$\boldsymbol{\mu}_k \in \mathbb{R}^2$ are illustrated by the black dots. The clusters then provides us with the classifier

$$\boldsymbol{x} \;\mapsto\; \mathcal{C}_K(\boldsymbol{x}) = \sum_{k=1}^{K} k \mathbb{1}_{\{\boldsymbol{x} \in C_k\}} \;\in\; \mathcal{K}.$$

The remaining difficulty is to find the optimal partition $(C_1, \ldots, C_K)$ of $\mathbb{R}^q$.

In general, the global minimum of (4.1) cannot easily be determined. However, using additional step (4.3) we can provide an algorithm that converges to a local minimum. Consider the set of available features by $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^q$.

K-Means Clustering Algorithm.

---

(0) Choose an initial classifier $\mathcal{C}_K^{(0)} : \mathcal{X} \to \mathcal{K}$ with corresponding sample means $(\boldsymbol{\mu}_k^{(0)})_{k \in \mathcal{K}}$ on this initial partition, see (4.2).

(1) Repeat for $t \geq 1$ until no changes are observed:

(a) given the present sample means $(\boldsymbol{\mu}_k^{(t-1)})_{k \in \mathcal{K}}$ choose the updated classifier $\mathcal{C}_K^{(t)} : \mathcal{X} \to \mathcal{K}$ such that for each $\boldsymbol{x}_i \in \mathcal{X}$ we have

$$\mathcal{C}_K^{(t)}(\boldsymbol{x}_i) \;=\; \operatorname*{argmin}_{k \in \mathcal{K}} \|\boldsymbol{\mu}_k^{(t-1)} - \boldsymbol{x}_i\|_2^2 \;\in\; \mathcal{K};$$

(b) calculate the sample means $(\boldsymbol{\mu}_k^{(t)})_{k \in \mathcal{K}}$ on the new partition induced by classifier $\mathcal{C}_K^{(t)} : \mathcal{X} \to \mathcal{K}$.

- The K-means clustering algorithm converges: Note that due to the minimization in step (1a) and due to (4.3) for step (1b) each iteration in (1) reduces the total within-cluster dissimilarity. These two steps are illustrated in Figure 16: the left-hand side illustrates step (1a) of the algorithm, which aims at finding the best matching cluster center $\boldsymbol{\mu}_k^{(t-1)}$; the right-hand side illustrates step (1b) which updates the cluster centers (from black to orange dots). Thus, we receive a sequence with decreasing value for the objective function that is bounded from below by zero, and, henceforth, we have convergence. However, we may end up in a local minimum of the objective function. Therefore, one may use different (random) initial classifiers (seeds) $\mathcal{C}_K^{(0)}$ in step (0) of the algorithm to back-test the solution.

- Another issue is the choice of the hyperparameter K for the number of clusters considered. We may start with running the algorithm for $K = 2$ which leads to a binary partition $(C_k)_{k=1,2}$ with sample means $(\boldsymbol{\mu}_k)_{k=1,2}$. For $K = 3$, we may then use these sample means $(\boldsymbol{\mu}_k)_{k=1,2}$ together with an arbitrary value $\boldsymbol{\mu} \in \mathbb{R}^q$ as initial values for the K-means clustering algorithm with $K = 3$. This choice ensures that the resulting total within-cluster dissimilarity is decreasing in K.

Electronic copy available at: https://ssrn.com/abstract=3439358

- For an arbitrary feature $\boldsymbol{x} \in \mathbb{R}^q$ we extend classifier $\mathcal{C}_K^{(t)}$ of the above algorithm by finding the best matching cluster center to

$$\boldsymbol{x} \;\mapsto\; \mathcal{C}_K^{(t)}(\boldsymbol{x}) \;=\; \underset{k\in\mathcal{K}}{\operatorname{argmin}} \; \|\boldsymbol{\mu}_k^{(t-1)} - \boldsymbol{x}\|_2^2.$$

- For further information: The Voronoi partition (tessellation) is dual to the Delaunay triangulation, see Fisher [8]. Consider the dual graph to the Voronoi partition by connecting the cluster centers. The Voronoi partition is then obtained such that the Voronoi segments split the cluster connections orthogonally at half distance between the cluster centers.

4.2.2 Example, revisited

We revisit the car models example of Listing 1. The goal is to cluster these car models using the standardized features $\boldsymbol{x}_i$ of each car $i = 1, \ldots, n$, pre-processed according to Conventions 2.1. In

Listing 7: K-means clustering in R using `kmeans`

```
Kaverage <- colMeans(X)                    # should be zero because X is normalized

K0 <- 10                                   # maximal number of K-means clusters
TWCD <- array(NA, c(K0))                   # total within-cluster dissimilarity (TWCD)
Classifier <- array(1, c(K0, nrow(X)))     # classification


TWCD[1] <- sum(colSums(as.matrix(X^2)))    # total dissimilarity for X normalized

set.seed(100)                              # set seed
for (K in 2:K0){                           # determine K-means for K=2,...,K0
   if (K==2){K_res <- kmeans(X,K)}
   if (K>2) {K_res <- kmeans(X,K_centers)}
   TWCD[K]          <- sum(K_res$withins)
   Classifier[K,]   <- K_res$cluster
   K_centers        <- array(NA, c(K+1, ncol(X)))
   K_centers[K+1,] <- Kaverage
   K_centers[1:K,] <- K_res$centers
                }
```

Listing 7 we provide the R code to perform the K-means clustering. The first line of this listing should provide zero, because the design matrix $\mathbf{X}$ has been standardized under Convention 2.1. We perform K-means clustering for $K = 2, \ldots, 10$, and we always use the previous cluster centers $(\boldsymbol{\mu}_k)_{k\in\mathcal{K}}$ as initial clusters for the K-means clustering with $K+1$ clusters in step (0) of the K-means clustering algorithm above. This way we receive a decreasing total within-cluster dissimilarity for increasing hyperparameter K. This is illustrated in Figure 17 (lhs).
For further analysis of our results we choose hyperparameter $K = 4$,[14] i.e. we consider partitioning into four clusters. In Figure 17 (middle, rhs) we illustrate the resulting clustering of the K-means algorithm outlined in Listing 7. The graph in the middle illustrates the four clusters (in red, orange, magenta and blue colors) on the first two principal component axes obtained from the PCA of Listing 2. The locations of the dots are identical to Figure 8, but the coloring is different. In Figure 8 (lhs) the dots are colored according to the Belgium expert criterion for

[14]The hyperparameter choice $K = 4$ corresponds to the elbow method in Figure 17 (lhs), which tries to find the kink in the decreasing dissimilarities (as a function of K).

Electronic copy available at: https://ssrn.com/abstract=3439358

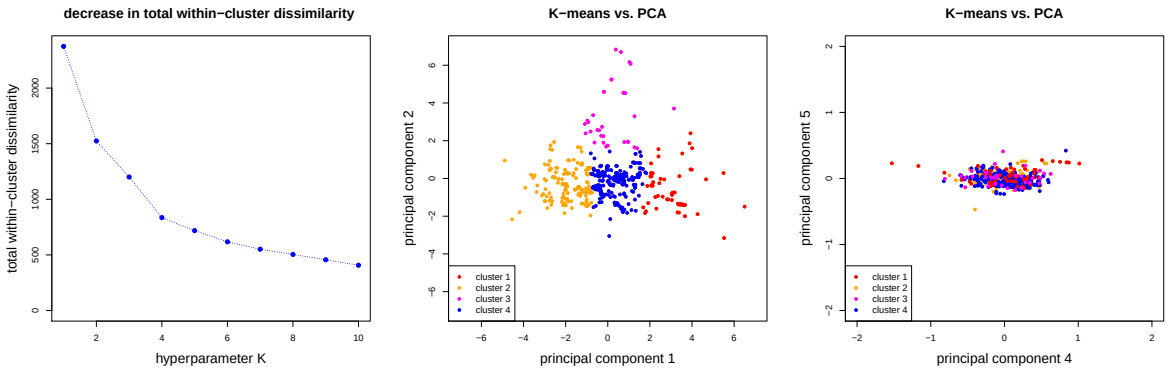

Figure 17: (lhs) total within-cluster dissimilarities for increasing $K = 1, \ldots, 10$, (middle) 4-means clustering w.r.t. the first two principal components, (rhs) 4-means clustering w.r.t. the last two principal components.

sports cars, in Figure 17 (middle) they are colored according to the 4 clusters from 4-means clustering. Remarkable is that in 4-means clustering we obtain sharp color borders w.r.t. the first two principal components of the PCA. This expresses that 4-means clustering essentially uses the first two principal components for clustering. This is further supported by Figure 17 (rhs) which shows the clusters w.r.t. the last two principal components. In this case the colored dots are completely mixed, which means that small singular values are less important for K-means clustering.

	cluster 1	cluster 2	cluster 3	cluster 4
number of cars	59	145	33	238
sports cars	50	0	1	21
in %	85%	0%	3%	9%

Table 4: K-means clustering of sports cars for $K = 4$.

Table 4 summarizes the K-means clustering results w.r.t. sports cars (expert judgment) for $K = 4$. Interestingly, most of the sports cars belong to cluster 1 and a couple of sports cars fall into cluster 4 (which contains roughly 50% of all cars). Thus, cluster 1 could be called the "sports car cluster", and cluster 4 may need further analysis.

4.3 K-medoids clustering

4.3.1 Methodology and algorithm

K-medoids clustering is a centroid-based method that is closely related to K-means clustering. The main difference between these two methods is that the K-medoids clustering uses explicit data points $\boldsymbol{x}_i$ as cluster centers, whereas the sample means $\boldsymbol{\mu}_k$ of the K-means clustering typically do not belong to the observed data points $\mathcal{X}$. Moreover, the K-medoids clustering may consider dissimilarity measures different from squared Euclidean distances. The K-medoids method is more robust than the K-means clustering if we choose dissimilarity measures that can deal with outliers, for instance, an absolute value distance gives less weight to outliers than

Electronic copy available at: https://ssrn.com/abstract=3439358

a squared Euclidean distance. The resulting cluster centers are called *medoids* because they are located most centrally within the cluster. We minimize the following objective function

$$\underset{(\boldsymbol{c}_1,\ldots,\boldsymbol{c}_K)\subset\mathcal{X}}{\arg\min} \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i\in C_k\cap\mathcal{X}} d(\boldsymbol{c}_k,\boldsymbol{x}_i), \tag{4.4}$$

where the medoids $\boldsymbol{c}_k \in \mathcal{X} = \{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n\}$ belong to the data points, where $d(\cdot,\cdot)$ is a dissimilarity function on $\mathbb{R}^q$, and where the clusters around the medoids $\boldsymbol{c}_k$ are given by

$$C_k = \{\boldsymbol{x}\in\mathcal{X};\ d(\boldsymbol{c}_k,\boldsymbol{x}) < d(\boldsymbol{c}_l,\boldsymbol{x}) \text{ for all } l\neq k\}, \tag{4.5}$$

with a deterministic rule if we do not have a unique best matching cluster center $\boldsymbol{c}_k$.
Again the global minimum is difficult to find, therefore, we are typically satisfied by a local minimum. This can be found by the partitioning around medoids (PAM) algorithm which goes back to Kaufman–Rousseeuw [16, 17].

Partitioning Around Methods Algorithm.

---

(0) Choose initial medoids $\boldsymbol{c}_1,\ldots,\boldsymbol{c}_K \in \mathcal{X}$, allocate each data point $\boldsymbol{x}_i \in \mathcal{X}$ to its closest medoid, see (4.5), and calculate the resulting total within-cluster dissimilarity (TWCD)

$$\text{TWCD} = \sum_{k=1}^{K} \sum_{\boldsymbol{x}_i\in C_k\cap\mathcal{X}} d(\boldsymbol{c}_k,\boldsymbol{x}_i).$$

(1) Repeat until no decrease in total within-cluster dissimilarity TWCD is observed: for each medoid $\boldsymbol{c}_k$ and for each non-medoid $\boldsymbol{x}_i$:

   (a) swap the role of $\boldsymbol{c}_k$ and $\boldsymbol{x}_i$, and allocate each data point to the closest medoid under this new configuration;

   (b) calculate the new total within-cluster dissimilarity;

   (c) if the total within-cluster dissimilarity increases then reject the swap, otherwise keep the swap.

Remarks.

- The are many variants of the swap step (1a). In the algorithm below we use the original version of Kaufman–Rousseeuw [16, 17] which is described in Algorithm 2 of Schubert–Rousseeuw [28]. This latter reference also provides several (computational) improvements.

- Note that in the PAM algorithm we only work on the data $\mathcal{X}$ because the medoids are also part of this data set. Therefore, all dissimilarities $d(\boldsymbol{x}_i,\boldsymbol{x}_l)$ only need to be calculated once (resulting in a matrix), and then the PAM algorithm can entirely be calculated based on this dissimilarity matrix. For this reason, the algorithm works with any dissimilarity function which can directly be provided to the algorithm in terms of the resulting dissimilarity matrix.

Electronic copy available at: https://ssrn.com/abstract=3439358

Listing 8: K-medoids clustering in R using `pam`

```
library(cluster)

# pamonce=FALSE is the original version of Kaufman-Rousseeuw (1987,1990)
set.seed(100)
pam(X, k=4, metric="manhattan", diss=FALSE, pamonce=FALSE)
```

4.3.2 Example, revisited

We revisit the car models example of Listing 1. In Listing 8 we provide the R code to perform the K-medoids clustering with the `pam` algorithm of the R library `cluster`. We choose $K = 4$ clusters, as dissimilarity function we use the Manhattan metric which is the sum of the absolute values of the differences, and we set `diss` to `FALSE` which means that we provide the data matrix $\mathbf{X}$. Instead, as described above, we could also provide a dissimilarity matrix which would allow us to choose an arbitrary dissimilarity function $d(\cdot, \cdot)$.

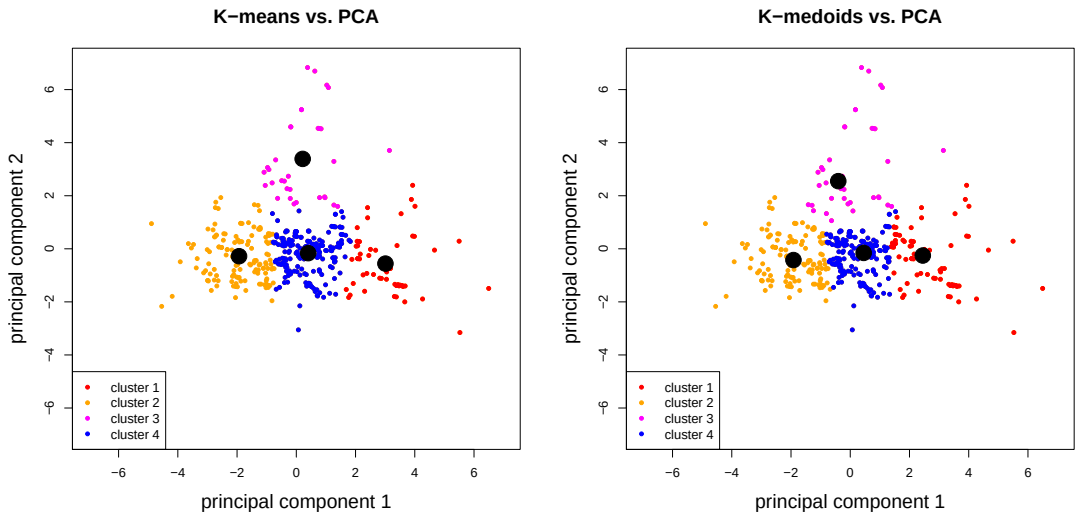

Figure 18: (lhs) 4-means clustering w.r.t. the first two principal components, (rhs) 4-medoids clustering w.r.t. the first two principal components and using the Manhattan metric as dissimilarity function.

The results are provided in Figure 18. The left-hand side gives the K-means result with the squared Euclidean distance, and the right-hand side gives the K-medoids results for the Manhattan distance, both with $K = 4$. We observe that for the Manhattan distance the cluster centers (black dots) are closer together, this comes from the fact that the L^1-norm punishes outliers less heavily than the squared Euclidean norm. Besides this, the clustering takes a rather similar form in the two methods.

4.4 Clustering using Gaussian mixture models

The K-means algorithm is based on the implicit assumption that dissimilarity is roundish. Gaussian mixture models (GMMs) are distribution-based clustering methods that provide more

Electronic copy available at: https://ssrn.com/abstract=3439358

flexibility with respect to this assumption.

4.4.1 Methodology

Observe that all methods studied above have not been based on any assumptions on how the features $\boldsymbol{x}_i \in \mathcal{X}$ could have been generated. In this section we underpin a probabilistic model on how these features could have been generated.

Model 4.1 *Choose hyperparameter $K \in \mathbb{N}$. Assume that the features $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are i.i.d. realizations from the density of a weighted sum of normal distributions*

$$f(\boldsymbol{x}) = \sum_{k=1}^{K} \frac{1}{(2\pi|\Sigma_k|)^{q/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right\} p_k, \tag{4.6}$$

with mean vectors $\boldsymbol{\mu}_k \in \mathbb{R}^q$, positive definite covariance matrices $\Sigma_k \in \mathbb{R}^{q\times q}$, and having probabilities (weights) $\boldsymbol{p} = (p_1, \ldots, p_K) \in \mathcal{S}_K$ from the $(K-1)$-unit simplex

$$\mathcal{S}_K = \left\{ \boldsymbol{p} \in \mathbb{R}^K;\ p_k \geq 0 \text{ and } \sum_{k=1}^{K} p_k = 1 \right\}.$$

Density (4.6) describes a multivariate GMM with parameter $\theta = (\boldsymbol{\mu}_k, \Sigma_k, p_k)_{k\in\mathcal{K}}$, that is, K multivariate Gaussian distributions are mixed with mixing probability $\boldsymbol{p} \in \mathcal{S}_K$. The mean vectors $\boldsymbol{\mu}_k \in \mathbb{R}^q$ play the role of the K cluster centers that we try to find. Based on i.i.d. observations $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ one is tempted to directly estimate the parameter θ with maximum likelihood estimation (MLE) methods, which provide estimates for the cluster centers. The log-likelihood function is given by

$$\ell_{(\boldsymbol{x}_i)_i}(\theta) = \sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} \frac{1}{(2\pi|\Sigma_k|)^{q/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)\right\} p_k \right). \tag{4.7}$$

Unfortunately, this MLE problem leads to a non-trivial optimization problem, therefore, we are going to study a modified problem which can be solved more easily.

Instead of representing the density of the cases as a weighted sum of Gaussian distributions (4.6), we introduce a latent variable $\boldsymbol{Z} = (Z^1, \ldots, Z^K)$ indicating from which particular Gaussian distribution a selected observation $\boldsymbol{x}$ has been sampled from. Assume that $\boldsymbol{Z}$ takes values in

$$\mathcal{S}_K^\circ = \left\{ \{0,1\}^K;\ \sum_{k=1}^{K} Z^k = 1 \right\} \ \subset\ \mathcal{S}_K. \tag{4.8}$$

That is, $\boldsymbol{Z}$ takes values in the corners of the $(K-1)$-unit simplex $\mathcal{S}_K$. This is a categorical random variable represented in one-hot encoding, we refer to our tutorial [7] for one-hot encoding. Set

$$p_k = \mathbb{P}[Z^k = 1] > 0, \qquad \text{for } k \in \mathcal{K}.$$

Note that we exclude the boundaries $p_k = 0$ because this is equivalent to a reduction of the number K of cluster centers. Then, we can re-express the multivariate GMM as follows

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{z}\in\mathcal{S}_K^\circ} f(\boldsymbol{x}, \boldsymbol{z}),$$

Electronic copy available at: https://ssrn.com/abstract=3439358

with joint density for $(\boldsymbol{x}, \boldsymbol{z}) \in \mathbb{R}^q \times \mathcal{S}_K^\circ$

$$f(\boldsymbol{x}, \boldsymbol{z}) = f_{\mathcal{N}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z}) = \sum_{k=1}^{K} z^k \frac{1}{(2\pi|\Sigma_k|)^{q/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top \Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right\} p_k. \tag{4.9}$$

Under the assumption of having i.i.d. data $(\boldsymbol{x}_i, \boldsymbol{z}_i)$, $i = 1, \ldots, n$, from the joint density (4.9) we receive log-likelihood function (in θ)

$$\begin{aligned}
\ell_{(\boldsymbol{x}_i,\boldsymbol{z}_i)_i}(\theta) &= \sum_{i=1}^{n}\sum_{k=1}^{K} z_i^k \left(-\frac{q}{2}\log(2\pi|\Sigma_k|) - \frac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^\top\Sigma_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)\right) + \sum_{i=1}^{n}\sum_{k=1}^{K} z_i^k \log(p_k) \\
&= \sum_{i=1}^{n}\sum_{k=1}^{K} z_i^k \log f_{\mathcal{N}}(\boldsymbol{x}_i|\boldsymbol{\mu}_k, \Sigma_k) + \sum_{i=1}^{n}\sum_{k=1}^{K} z_i^k \log(p_k).
\end{aligned} \tag{4.10}$$

Comparing this to (4.7) we observe that we get rid off the summation within the logarithm because we replace the probabilities p_k by observations z_i^k. The first term on the right-hand side only depends on the Gaussian parameters $(\boldsymbol{\mu}_k, \Sigma_k)_k$, and classical MLE on Gaussian densities can be performed to estimate these parameters. That is, we receive MLEs

$$\widehat{\boldsymbol{\mu}}_k = \widehat{\boldsymbol{\mu}}_k\left((\boldsymbol{x}_i, \boldsymbol{z}_i)_{1\le i\le n}\right) = \frac{\sum_{i=1}^n z_i^k \boldsymbol{x}_i}{\sum_{i=1}^n z_i^k}, \tag{4.11}$$

and

$$\widehat{\Sigma}_k = \widehat{\Sigma}_k\left((\boldsymbol{x}_i, \boldsymbol{z}_i)_{1\le i\le n}\right) = \frac{\sum_{i=1}^n z_i^k \left(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_k\right)\left(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_k\right)^\top}{\sum_{i=1}^n z_i^k}. \tag{4.12}$$

The second term describes a multinomial distribution depending on the parameter $\boldsymbol{p} = (p_k)_k \in \mathcal{S}_K$, which can also be estimated with MLE

$$\widehat{p}_k = \widehat{p}_k\left((\boldsymbol{z}_i)_{1\le i\le n}\right) = \frac{1}{n}\sum_{i=1}^{n} z_i^k. \tag{4.13}$$

At the first sight, this does not seem to be useful because the latent variables $\boldsymbol{Z}_i$ have not been observed. The expectation-maximization (EM) algorithm is an appropriate tool to estimate such models in the absence of observations for latent variables.

4.4.2 Expectation-maximization algorithm

The EM algorithm consists of two steps: (a) estimate the latent variables $(\boldsymbol{Z}_i)_i$ from $(\boldsymbol{x}_i)_i$ and $\widehat{\theta}$, where $\widehat{\theta}$ is an estimate for θ, and (b) estimate the model parameter θ from $(\boldsymbol{x}_i, \widehat{\boldsymbol{Z}}_i)_i$, where $\widehat{\boldsymbol{Z}}_i$ are estimates for $\boldsymbol{Z}_i$. This is similar to the K-means algorithm on page 26, where (a) we re-assess the best matching clusters using the estimated cluster centers, (b) we compute the cluster centers $\boldsymbol{\mu}_k$ based on the estimated clusters.

Step (a) is called *E-step* for expectation step. The posterior probability of $Z^k = 1$, given observation $\boldsymbol{x}$, is given by

$$p_k(\theta|\boldsymbol{x}) = \mathbb{P}[Z^k = 1|\boldsymbol{x}] = \frac{|\Sigma_k|^{-q/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^\top\Sigma_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right\}p_k}{\sum_{j=1}^K |\Sigma_j|^{-q/2}\exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^\top\Sigma_j^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)\right\}p_j}.$$

Electronic copy available at: https://ssrn.com/abstract=3439358

Therefore, the posterior estimate for Z^k, after having observed $\boldsymbol{x}$ for the Gaussian mixture random variable, is

$$\widehat{Z}^k(\theta|\boldsymbol{x}) = \mathbb{E}[Z^k|\boldsymbol{x}] = p_k(\theta|\boldsymbol{x}). \tag{4.14}$$

This posterior estimate is used as estimate for the hidden variables $\boldsymbol{Z}$.

Step (b) is called *M-step* for maximization step because we apply MLE.

EXPECTATION-MAXIMIZATION ALGORITHM.

---

(0) Choose an initial parameter $\theta^{(0)} = \left(\boldsymbol{\mu}_k^{(0)}, \Sigma_k^{(0)}, p_k^{(0)}\right)_{k\in\mathcal{K}}$.

(1) Repeat for $t \geq 1$:

(a) *E-step*: given parameter $\theta^{(t-1)} = \left(\boldsymbol{\mu}_k^{(t-1)}, \Sigma_k^{(t-1)}, p_k^{(t-1)}\right)_{k\in\mathcal{K}}$ we estimate the latent variables $\boldsymbol{Z}_i$, $i = 1, \ldots, n$, by, see (4.14),

$$\widehat{\boldsymbol{Z}}_i^{(t)} = \left(p_1(\theta^{(t-1)}|\boldsymbol{x}_i), \ldots, p_K(\theta^{(t-1)}|\boldsymbol{x}_i)\right) \in \mathcal{S}_K.$$

(b) *M-step*: calculate the MLE $\theta^{(t)} = \left(\boldsymbol{\mu}_k^{(t)}, \Sigma_k^{(t)}, p_k^{(t)}\right)_{k\in\mathcal{K}}$ based on the (estimated) observations $(\boldsymbol{x}_i, \widehat{\boldsymbol{Z}}_i^{(t)})_{1\leq i\leq n}$, see (4.11)-(4.13)

$$\begin{aligned}
\widehat{\boldsymbol{\mu}}_k^{(t)} &= \frac{\sum_{i=1}^n p_k(\theta^{(t-1)}|\boldsymbol{x}_i)\boldsymbol{x}_i}{\sum_{i=1}^n p_k(\theta^{(t-1)}|\boldsymbol{x}_i)}, \\
\widehat{\Sigma}_k^{(t)} &= \frac{\sum_{i=1}^n p_k(\theta^{(t-1)}|\boldsymbol{x}_i)\left(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_k^{(t)}\right)\left(\boldsymbol{x}_i - \widehat{\boldsymbol{\mu}}_k^{(t)}\right)^\top}{\sum_{i=1}^n p_k(\theta^{(t-1)}|\boldsymbol{x}_i)}, \\
\widehat{p}_k^{(t)} &= \frac{1}{n}\sum_{i=1}^n p_k(\theta^{(t-1)}|\boldsymbol{x}_i).
\end{aligned}$$

Comparison of the K-means algorithm and the EM algorithm:

(a) The E-step calculates a posterior expectation in the EM algorithm, whereas in the K-means algorithm we do a "hard assessment" by allocating each case to the best matching cluster center.

(b) The M-step is the same in both algorithms, recalculating the model parameters after the cases have been re-allocated to the new best matching cluster centers.

4.4.3 Justification of the EM algorithm (the fast reader can skip this section)

Since the latent variables $(\boldsymbol{Z}_i)_i$ are not observable, we replace them by their posterior expectation (4.14). In view of (4.10), we interpret this as considering the expected log-likelihood conditioned

Electronic copy available at: https://ssrn.com/abstract=3439358

on having observed $(\boldsymbol{x}_i)_i$, using the assumed i.i.d. property we have

$$\begin{aligned}\mathbb{E}\left[\ell_{(\boldsymbol{x}_i,\boldsymbol{Z}_i)_i}(\theta)\middle|(\boldsymbol{x}_i)_i\right] &= \sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{E}[Z_i^k|\boldsymbol{x}_i]\left(-\frac{q}{2}\log(2\pi|\Sigma_k|)-\frac{1}{2}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)^\top\Sigma_k^{-1}(\boldsymbol{x}_i-\boldsymbol{\mu}_k)\right)\\ &\quad+\sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{E}[Z_i^k|\boldsymbol{x}_i]\log(p_k).\end{aligned}$$

We analyze this expected log-likelihood.

(1) We have for any density $\pi\in\mathcal{S}_K$, using Jensen's inequality,

$$\begin{aligned}\log f_\theta(\boldsymbol{x}) &= \log\sum_{\boldsymbol{z}\in\mathcal{S}_K^\circ}f_\theta(\boldsymbol{x},\boldsymbol{z}) = \log\sum_{\boldsymbol{z}\in\mathcal{S}_K^\circ}\pi(\boldsymbol{z})\,\frac{f_\theta(\boldsymbol{x},\boldsymbol{z})}{\pi(\boldsymbol{z})} \qquad (4.15)\\ &\geq \sum_{\boldsymbol{z}\in\mathcal{S}_K^\circ}\pi(\boldsymbol{z})\log\left(\frac{f_\theta(\boldsymbol{x},\boldsymbol{z})}{\pi(\boldsymbol{z})}\right) = \mathbb{E}_{\boldsymbol{Z}\sim\pi}\left[\log\left(f_\theta(\boldsymbol{x},\boldsymbol{Z})\right)\right]+K=\mathcal{L}(\pi;\theta),\end{aligned}$$

where K is a constant not depending on θ. The right-hand side $\theta\mapsto\mathcal{L}(\pi;\theta)$ is a concave function in θ (for any $\pi\in\mathcal{S}_K$), and, thus, it has a unique maximum. This is exactly what we use in the EM algorithm in the M-step.

(2) Moreover, one can show that the posterior choice $\pi=\boldsymbol{p}(\theta|\boldsymbol{x})\in\mathcal{S}_K$ provides an equality in (4.15), i.e. $\log f_\theta(\boldsymbol{x})=\mathcal{L}(\boldsymbol{p}(\theta|\boldsymbol{x});\theta)$, this motivates the E-step.

Combining (1) and (2): We initialize $\theta^{(0)}$ and set $\pi=\boldsymbol{p}(\theta^{(0)}|\boldsymbol{x})$. The M-step implies that we find a maximum in the new parameter $\theta^{(1)}$. From (4.15) we derive

$$f_{\theta^{(1)}}(\boldsymbol{x}) \geq \mathcal{L}\left(\boldsymbol{p}(\theta^{(0)}|\boldsymbol{x});\theta^{(1)}\right).$$

Using (2) we update $\pi=\boldsymbol{p}(\theta^{(1)}|\boldsymbol{x})$, which implies

$$f_{\theta^{(1)}}(\boldsymbol{x})=\mathcal{L}\left(\boldsymbol{p}(\theta^{(1)}|\boldsymbol{x});\theta^{(1)}\right) \geq \mathcal{L}\left(\boldsymbol{p}(\theta^{(0)}|\boldsymbol{x});\theta^{(1)}\right).$$

Iterating this for $t\geq 1$ implies that we find a sequence of parameters $(\theta^{(t)})_{t\geq 0}$ with

$$\ldots\leq f_{\theta^{(t-1)}}(\boldsymbol{x})\leq f_{\theta^{(t)}}(\boldsymbol{x})\leq f_{\theta^{(t+1)}}(\boldsymbol{x})\leq\ldots,$$

and therefore the EM algorithm converges to a (local) maximum of the log-likelihood function.

4.4.4 Example, revisited

We revisit the car models example of Listing 1. In Listing 9 we provide the R code to perform the GMMs clustering with `GMM` of the R package `ClusterR`. Pay attention: this package only estimates diagonal covariance matrices Σ_k. For general covariance matrices and many more options we refer to the R package `mclust`. We choose $K=4$ clusters.
We observe quite some differences between the K-means results and the GMM results, see Figure 19. Interesting is that the multivariate Gaussian distribution with the magenta cluster center has the biggest variance. For this reason, this magenta cluster center is moved more towards the

Electronic copy available at: https://ssrn.com/abstract=3439358

Listing 9: GMM clustering in R using `GMM` from the library `ClusterR`

```
library(ClusterR)    # GMM only considers diagonal covariance matrices!

set.seed(100)
K_res <- GMM(X, gaussian_comps=4, dist_mode="eucl_dist",
                seed_mode="random_subset", em_iter=5, seed=100)

predict_GMM(X, K_res$centroids, K_res$covariance_matrices, K_res$weights)$cluster_labels
```

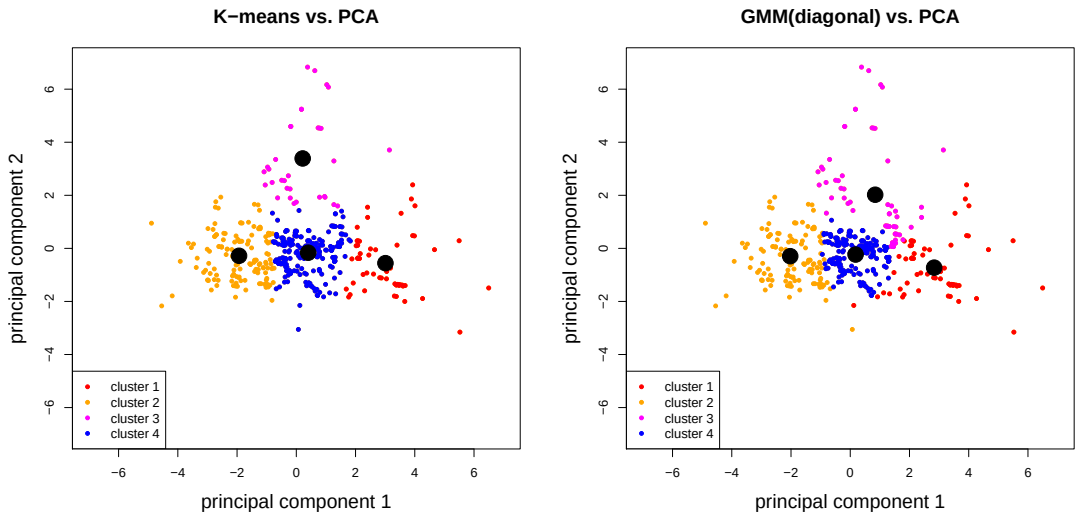

Figure 19: (lhs) 4-means clustering w.r.t. the first two principal components, (rhs) GMM clustering with diagonal covariance matrices and $K = 4$.

origin of the picture (compared to K-means) and it can still capture the outliers in the second principal component (because of its large variance parameter).

Remark. We have used the R package `ClusterR` which uses the assumption that the covariance matrices Σ_k are diagonal. Alternatively, we could use the R package `mclust` which allows for much more modeling flexibility. We can decouple the covariance matrices Σ_k as follows

$$\Sigma_k = \lambda_k D_k A_k D_k^\top,$$

where λ_k is a scalar, D_k is an orthogonal matrix containing the eigenvectors, and A_k is a diagonal matrix that is proportional to the eigenvalues of Σ_k. `mclust` then allows for different versions like EVI or VII. The first letter stands for the volume λ_k, the second letter for the shape A_k, and the third letter for the orientation D_k. Thus, EVI means Equal volumes $\lambda_k = \lambda$, Variable shapes A_k (ellipsoids), and the orientation is the Identity $D_k = \mathbb{1}$ (coordinate axes). For more details we refer to Table 1 in Fraley–Raftery [9].

4.4.5 Variational autoencoder: an outlook

There is an interesting connection between clustering with multivariate GMMs and autoencoders which have been introduced in Section 3, above. This connection is related to variational

Electronic copy available at: https://ssrn.com/abstract=3439358

autoencoders (VAEs) introduced and studied in Kingma–Welling [18].
The starting point of this connection is the joint density $f(\boldsymbol{x}, \boldsymbol{z})$ given in (4.9). In abstract terms, this joint density is given by

$$f(\boldsymbol{x}, \boldsymbol{z}) = f(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z}),$$

where $p(\boldsymbol{z})$ is a discrete distribution of the latent variable $\boldsymbol{Z}$, describing the choice of the cluster center, and where $f(\boldsymbol{x}|\boldsymbol{z})$ is the density of the observation, given cluster center $\boldsymbol{Z} = \boldsymbol{z}$. This is exactly the structural form as being used in VAEs, except that the discrete latent variable is replaced by an absolutely continuous latent variable $\boldsymbol{Z}$. Assume that there is an (unknown) model parameter θ such that the joint density of the observation and the absolutely continuous hidden variable is given by

$$f_\theta(\boldsymbol{x}, \boldsymbol{z}) = f_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z}).$$

Kingma–Welling [18] design an autoencoding variational Bayes (AEVB) algorithm to estimate the parameter θ and to infer the latent variable $\boldsymbol{Z}$ (doing approximate posterior inference).
Assume that $\boldsymbol{Z} = \boldsymbol{z} \in \mathbb{R}^p$ is a low dimensional latent variable that induces a high dimensional feature $\boldsymbol{x} \in \mathbb{R}^q$ via the law $f_\theta(\boldsymbol{x}|\boldsymbol{z})$. We may now define an encoder $\widehat{\boldsymbol{Z}}(\theta|\boldsymbol{x}) = \mathbb{E}_\theta[\boldsymbol{Z}|\boldsymbol{x}]$ which infers the latent variable $\boldsymbol{Z}$, having observed $\boldsymbol{x}$, this is the E-step from the EM algorithm above; and we get a probabilistic decoder that describes $f_\theta(\boldsymbol{x}|\boldsymbol{z})$, which relates to the M-step in the EM algorithm. Intuitively speaking, the AEVB algorithm tries to minimize a reconstruction error, and in this sense, $\widehat{\boldsymbol{Z}}(\theta|\boldsymbol{x})$ reflects a p dimensional approximation (description) of a high dimensional feature $\boldsymbol{x} \in \mathbb{R}^q$.

5 Topological approaches for dimension reductions

So far, we have been discussing two different types of methods. The first type of methods (PCA and BNNs) has been used to reduce the dimension of the data. The main objective in this dimension reduction has been to minimize the reconstruction error of the original data. The second type of methods (K-means, K-medoids and GMMs) has been aiming at categorizing data into clusters of similar cases. The methods that we are going to study in the remainder of this tutorial mainly aim at visualizing high dimensional data. This is also done by dimension reduction, but the main objective in this section is to keep the (local) topology of the data as far as possible. This is motivated by the idea that if $\mathcal{X}$ is a lower dimensional manifold in $\mathbb{R}^q$, then it can be described by lower dimensional object preserving the local topological relationships.

A simple way of motivating the following methods is multi-dimensional scaling (MDS). Assume that the cases $\boldsymbol{x}_i \in \mathcal{X}$ should be illustrated by two dimensional Euclidean objects $\boldsymbol{y}_i \in \mathbb{R}^2$ preserving as much as possible from the original topology. MDS tries to find these points as follows

$$\underset{\mathcal{Y}=(\boldsymbol{y}_1,\ldots,\boldsymbol{y}_n)}{\arg\min} \sum_{i,j} \left(d(\boldsymbol{x}_i, \boldsymbol{x}_j) - \|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2\right)^2.$$

Thus, we try to find points $\mathcal{Y}$ in the Euclidean plane $\mathbb{R}^2$ that preserve as much as possible from the original dissimilarity (adjacency) matrix $(d(\boldsymbol{x}_i, \boldsymbol{x}_j))_{1\leq i,j\leq n}$. This motivates the subsequent methods that are (slightly) more sophisticated than MDS.

Electronic copy available at: https://ssrn.com/abstract=3439358

Additional introductory remarks.

- We make a link to the clustering methods of the previous section. Assume that we have cases $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ illustrated by n vertices. Connect all vertices with undirected edges $[\boldsymbol{x}_i, \boldsymbol{x}_j]$, and attach to each edge a distance $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ that assesses the dissimilarity between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. A minimal spanning tree (MST) is a connected path $\mathfrak{T}$ that visits all vertices $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ (at least once) at minimal costs $\sum_{[\boldsymbol{x}_i, \boldsymbol{x}_j] \in \mathfrak{T}} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Note that $\mathfrak{T}$ is automatically a tree because any loop only creates extra costs. A MST $\mathfrak{T}$ naturally induces a neighborhood relationship along the tree (explained by an adjacency matrix). In this sense, a MST gives a topological representation of the cases $\mathcal{X}$.

- The K-medoids clustering can be understood as a spanning tree, not necessarily a minimal one. The K medoids $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K \in \mathcal{X}$ build the skeleton of the spanning tree. 1) Connect this skeleton by a MST $\mathfrak{T}_K$; and 2) connect each case $\boldsymbol{x}_i \in C_k$ to its cluster center $\boldsymbol{c}_k$. This provides the spanning tree of the K-medoids clustering which gives a natural proximity relationship along the constructed spanning tree. This spanning tree has the interpretation of possessing hubs (medoids) from which the fine distribution to all cases is done.

- A MST $\mathfrak{T}$ is rather close to UMAPs presented in Section 5.2, below, and SOMs presented in Section 5.3, below. A main difference is that the degree of each vertex $\boldsymbol{x}_i$ is variable in MSTs, whereas for the UMAP and the SOM we will choose a fixed degree for each vertex which will describe proximity more locally.

5.1 t-distributed stochastic neighboring embedding

The method that we are going to study in this section is t-distributed stochastic neighbor embedding (t-SNE) which has been developed by van der Maaten–Hinton [30].

5.1.1 Methodology

The idea behind t-SNE is to study proximity weights between all available features $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Dissimilar cases will receive small weights, and similar cases will attain high weights.[15] Based on these weights we will construct a low dimensional t-distribution which has a small KL divergence w.r.t. these proximity weights. This t-distribution then leads us to the low dimensional illustration of the original data. The KL divergence has been introduced in (3.6), and for two probabilities in the $(J-1)$-unit simplex, $\boldsymbol{q}, \boldsymbol{p} \in \mathcal{S}_J$, it is given by

$$D_{\mathrm{KL}}(\boldsymbol{q}||\boldsymbol{p}) = \sum_{j=1}^{J} q_j \log\left(\frac{q_j}{p_j}\right).$$

This reflects the gain of information if we walk from $\boldsymbol{p}$ to $\boldsymbol{q}$. Note that the KL-divergence is asymmetric: the role of the increased information $\boldsymbol{q}$ will be played by the original features (because they contain all available information), and the role of $\boldsymbol{p}$ will be played by the t-distributed approximation (because a projection leads to a loss of information).

[15] Proximity weights can be understood as inverse dissimilarity measures, for instance, used in minimal spanning trees (MSTs).

Electronic copy available at: https://ssrn.com/abstract=3439358

Step 1. We select two cases $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$, and we define the conditional probability weight

$$q_{j|i} = \frac{\exp\left\{-\frac{1}{2\sigma_i^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2\right\}}{\sum_{k\neq i}\exp\left\{-\frac{1}{2\sigma_i^2}\|\boldsymbol{x}_i - \boldsymbol{x}_k\|_2^2\right\}}, \qquad \text{for } i \neq j. \tag{5.1}$$

The meaning and the choice of $\sigma_i > 0$ is described in Remarks 5.1, below. $q_{j|i}$ quantifies the similarity of $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$. We make these weights symmetric by defining

$$q_{i,j} = \frac{1}{2n}\left(q_{j|i} + q_{i|j}\right), \qquad \text{for } i \neq j. \tag{5.2}$$

Observe that $\boldsymbol{q} = (q_{i,j})_{i\neq j} \in \mathcal{S}_J$ is a distribution from the $(J-1)$-unit simplex with $J = (n^2-n)/j$ (note that we exclude the "diagonal" $i = j$). This distribution $\boldsymbol{q}$ is used to explain the inner geometry of the features $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} = \mathcal{X} \subset \mathbb{R}^q$. The parameters $\sigma_i > 0$ determine the bandwidth considered in the weights $q_{j|i}$ (Gaussian kernels), and we typically choose smaller values for this bandwidth in areas where the cases $\boldsymbol{x}_i$ are more dense.

Step 2. We choose dimension $p < q$, and we aim at finding $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\} = \mathcal{Y} \subset \mathbb{R}^p$ such that the following probabilities are similar to $\boldsymbol{q}$. Define the Student-t probabilities (with 1 degree of freedom)

$$p_{i,j} = \frac{\left(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2^2\right)^{-1}}{\sum_{k\neq l}\left(1 + \|\boldsymbol{y}_k - \boldsymbol{y}_l\|_2^2\right)^{-1}}, \qquad \text{for } i \neq j, \tag{5.3}$$

and set $\boldsymbol{p} = (p_{i,j})_{i\neq j} \in \mathcal{S}_J$.

Goal. Find locations $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\}$ such that the KL divergence $D_{\mathrm{KL}}(\boldsymbol{q}||\boldsymbol{p})$ is minimized. These locations $\mathcal{Y}$ provide the low dimensional illustration of the original data $\mathcal{X}$.

Remarks 5.1

- The gradient descent algorithm is used to find the optimal locations $\mathcal{Y}$.
- In (5.3) we directly define a low dimensional symmetric distribution $\boldsymbol{p}$. In contrast to (5.3), in the high dimensional case we start from an asymmetric definition (5.1) given by $(q_{j|i})_{i\neq j}$, which is made symmetric in a second step in (5.2). The reason for starting with an asymmetric definition is that this approach provides us with more robustness towards individual outliers: if we have a symmetric definition similar to (5.3) and if $\boldsymbol{x}_i$ is far away from all other points $\boldsymbol{x}_j$, then $q_{i,j}$ would be small for any point $\boldsymbol{x}_j$. This would imply that the choice of $\boldsymbol{y}_i$ would influence to objective function only marginally. As a result, the position of $\boldsymbol{y}_i$ would not be well-specified. This problem is circumvented by (5.1)-(5.2) because the resulting $q_{i,j}$'s have the property $\sum_j q_{i,j} > 1/(2n)$ for all i.
- We choose the Student-t distribution with 1 degree of freedom because it has the nice property that $p_{i,j} \approx \|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2^{-2}$ for $\|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2 \to \infty$.
- The remaining parameter to be selected is the bandwidth given by $\sigma_i > 0$. Every choice σ_i provides a different conditional distribution $\boldsymbol{q}_{\bullet|i} = (q_{j|i})_{j\neq i}$. Usually, a smaller value for σ_i

Electronic copy available at: https://ssrn.com/abstract=3439358

is more appropriate in denser regions. A good choice for σ_i keeps the perplexity $\text{Perp}(\boldsymbol{q}_{\bullet|i})$ constant in i, where the perplexity is a measure for the effective number of neighbors. It is given by

$$\text{Perp}(\boldsymbol{q}_{\bullet|i}) = \exp\Big\{H(\boldsymbol{q}_{\bullet|i})\Big\} = \exp\left\{-\sum_{j\neq i} q_{j|i}\log_2(q_{j|i})\right\},$$

where $H(\boldsymbol{q}_{\bullet|i})$ is the Shannon entropy.

5.1.2 Example, revisited

We revisit the car models example of Listing 1. In Listing 10 we provide the R code to perform

Listing 10: t-SNE code in R using `tnse`

```
library(tsne)

set.seed(100)
tsne(X, k=2, initial_dim=ncol(X), perplexity=30)
```

the t-SNE dimension reduction using the `tsne` package of R. We choose a $p = 2$ dimensional illustration, and we minimize the KL divergence using the gradient descent algorithm. This algorithm needs a seed and correspondingly the solution will depend on the choice of this seed.

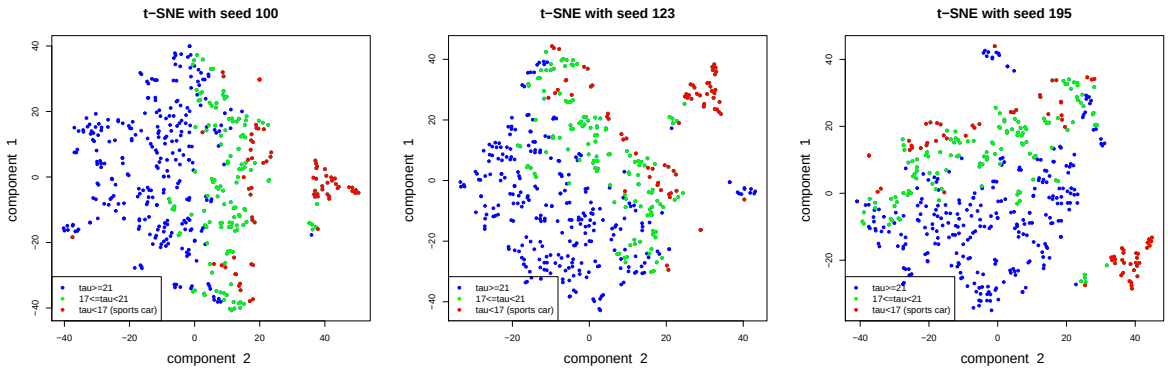

Figure 20: t-SNE illustration of the data $\mathcal{X}$ using three different seeds in the gradient descent algorithm.

In Figure 20 we provide the resulting illustrations. We observe that the solutions differ for different seeds, however, they have quite some similarities in terms of the red dots (sports cars with $\tau < 17$), green dots (cars with $17 \leq \tau < 21$) and blue dots (cars with $\tau \geq 21$). We find common clusterings in the three plots, there is a red cluster that appears in all three plots, and there is a small blue one which appears at least in the middle plot and the one on the right-hand side. The green dots build a lengthy cluster which has a similar structure in all three graphs. As described on page 20 we could use procrustes analysis to study the resulting differences in the plots of Figure 20, procrustes analysis would align the graphs so that they can be superimposed and compared. We refrain from doing so here.

Electronic copy available at: https://ssrn.com/abstract=3439358

If we interpret the results of Figure 20 w.r.t. sports cars we see that there are rather clear cases of sports cars, and cases which are less obvious. In some sense this is similar to the findings in K-means clustering, see Table 4. This concludes the t-SNE example.

5.2 Uniform manifold approximation and projection

Uniform manifold approximation and projection (UMAP) is a manifold learning technique for dimension reduction. It is based on Riemannian geometry and algebraic topology. In this tutorial, we are not going into the mathematical details, but we refer to work of McInnes et al. [24]. This work is important for the justification of the steps proposed below, the theoretical foundations of UMAP are outlined in Section 2 of McInnes et al. [24], and Section 3 of this reference is providing the computational part, which we are going to recall here briefly.

5.2.1 Methodology

UMAP is based on the assumption that the data $\mathcal{X} \subset \mathbb{R}^q$ is lying on a lower dimensional manifold, and it aims at learning the local structure to find the lower dimensional representation. The basic learning structure of the algorithm is similar to t-SNE, and has the following two steps.

Step 1. Assume that the dissimilarity measure $d : \mathbb{R}^q \times \mathbb{R}^q \to \mathbb{R}_+$ is a metric in $\mathbb{R}^q$. Choose a hyperparameter $k \in \mathbb{N}$ and calculate the k nearest neighbors of $\boldsymbol{x}_i \in \mathcal{X}$ w.r.t. $d(\cdot,\cdot)$; we also refer to the MST discussion on page 37. We denote these k nearest neighbors of $\boldsymbol{x}_i$ by $\mathcal{X}_i = \{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_k}\}$. Based on these k nearest neighbors we calculate the distance to the closest neighbor

$$\varrho_i = \min\left\{d(\boldsymbol{x}_i, \boldsymbol{x}_{i_j});\ 1 \le j \le k,\ d(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}) > 0\right\}, \tag{5.4}$$

and we choose a bandwidth $\sigma_i > 0$ such that

$$\sum_{j=1}^{k} \exp\left\{-\frac{\max\{0, d(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}) - \varrho_i\}}{\sigma_i}\right\} = \log_2(k).$$

This allows us to define proximity weights for a directed weighted graph. The vertices of the graph are the cases $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} = \mathcal{X}$ and the directed weighted edges between $\boldsymbol{x}_i$ and its k nearest neighbors $\mathcal{X}_i$ are obtained by the weights

$$q_{i_j|i} = \exp\left\{-\frac{\max\{0, d(\boldsymbol{x}_i, \boldsymbol{x}_{i_j}) - \varrho_i\}}{\sigma_i}\right\}.$$

This is similar to the construction in (5.1), but we only consider the k nearest neighbors here. Next, we are going to turn this asymmetric proximity relationship into a symmetric one, in analogy to step (5.2) in the t-SNE method. Define by $A \in \mathbb{R}_+^{n\times n}$ the adjacency matrix on $\mathcal{X}$ obtained from $(q_{i_j|i})_{i,j}$. A symmetric (undirected) version is defined by

$$Q = A + A^\top - A \circ A^\top,$$

where $\circ$ denotes the Hadamard product.[16] The matrix $Q = (q_{i,j})_{i,j}$ provides an undirected weighted graph on the vertices $\mathcal{X}$ describing the topology of the original data.

[16] The Hadamard product is the element wise product.

Electronic copy available at: https://ssrn.com/abstract=3439358

Step 2. We choose dimension $p < q$, and we aim to find a low dimensional representation $\mathcal{Y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n\} \subset \mathbb{R}^p$ of the original data $\mathcal{X}$. The idea behind UMAP in this second step is to design a force directed graph that is similar in topology to the original one. Since this step is technically challenging we will not provide the details here, but refer to Algorithms 4 and 5 in McInnes et al. [24]. We just mention that the fuzzy set cross entropy[17] is used between the representations of the original data $\mathcal{X}$ and the low dimensional illustration.

Remarks.

- The first hyperparameter involved is $k \in \mathbb{N}$ for the number of nearest neighbors to be considered. This defines the local scale at which we obtain a roughly flat manifold. Basically, individual information within the k nearest neighbor environment is lost as we will see below, and smaller values for k provide more pronounced and smaller clusters.
- `min_dist` is a second hyperparameter to be chosen. This hyperparameter determines how close the points in $\mathcal{Y}$ are. This hyperparameter is comparable to ϱ_i given in (5.4), and it will be illustrated in more detail in the example below.

5.2.2 Example, revisited

We revisit the car models example of Listing 1. In Listing 11 we provide the R code to perform

Listing 11: UMAP code in R using `umap`

```
library(umap)

umap.param <- umap.defaults
umap.param$n_components <- 2
umap.param$n_neighbors  <- 15
umap.param$random_state <- 100

umap(X, config=umap.param, method="naive")
```

the UMAP dimension reduction using the `umap` package of R. We choose a $p = 2$ dimensional illustration, $k = 15$ nearest neighbors, and `min_dist` $= 0.1$. This algorithm needs a seed and correspondingly the solution will depend on the choice of this seed.
The results are presented in Figure 21. We observe rather nice clusters for $k = 15$ and `min_dist` $= 0.1$ which are very similar for the different seeds. Indeed, the clustering in the UMAP method seems to depend much less on the initial seed than the one of t-SNE, compare Figures 20 and 21.
In Figure 22 we analyze the sensitivities of the UMAP method in the two hyperparameters k for the number of nearest neighbors, and `min_dist` for the separation between the points in the projection $\mathcal{Y}$. The first row in Figure 22 is based on `min_dist` $= 0.1$ and the second row on `min_dist` $= 0.5$. We observe that the bigger this value to more uniformly the points are spread in $\mathbb{R}^p$ because they become more repulsive with increasing `min_dist`. The second observation

[17] The cross entropy of two fuzzy sets (A, μ) and (A, ν) is given by $\sum_{a \in A} \mu(a) \log\left(\frac{\mu(a)}{\nu(a)}\right) + (1-\mu(a)) \log\left(\frac{1-\mu(a)}{1-\nu(a)}\right)$, which is closely related to the corresponding KL divergences.

Electronic copy available at: https://ssrn.com/abstract=3439358

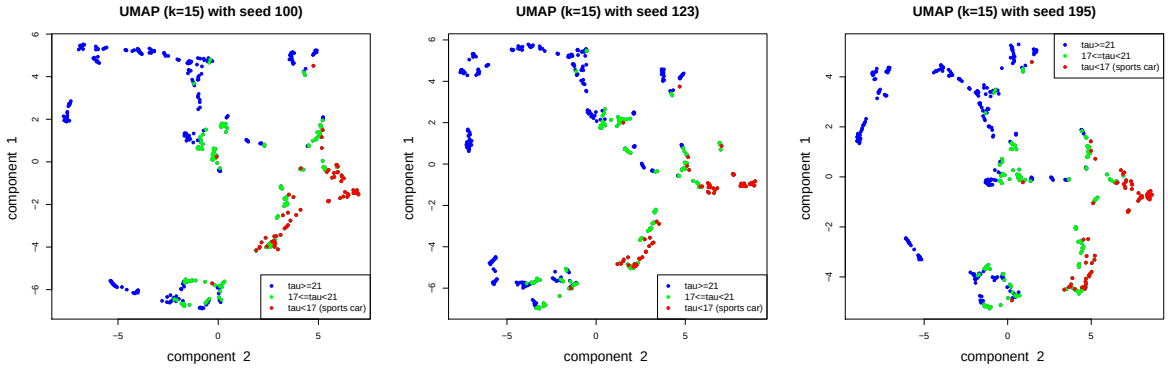

Figure 21: UMAP illustration of the data $\mathcal{X}$ using three different seeds with $k = 15$ nearest neighbors and `min_dist` $= 0.1$.

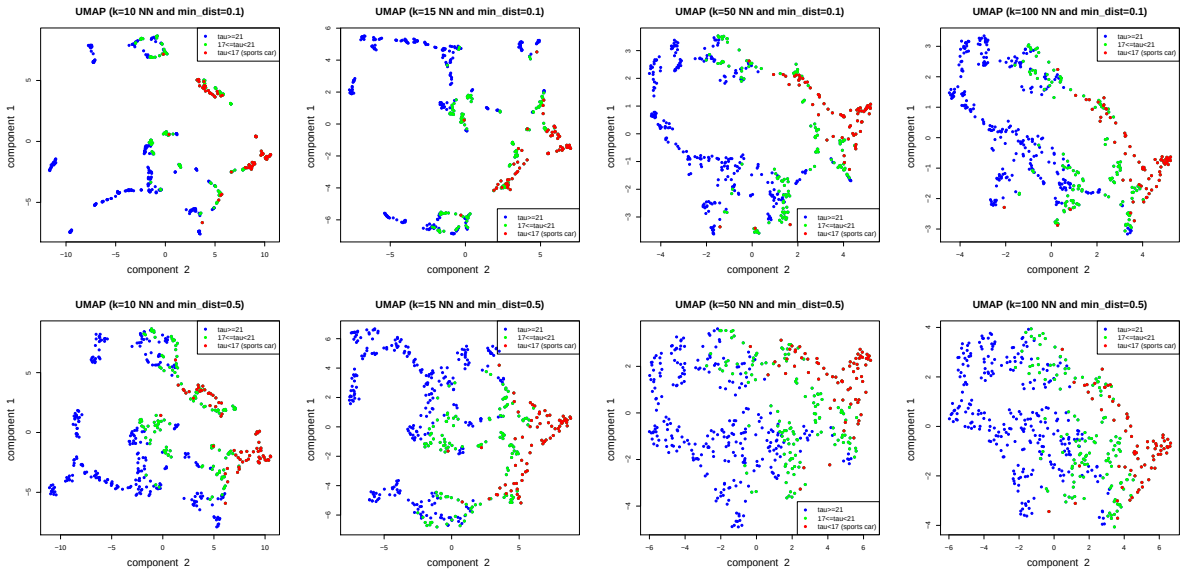

Figure 22: UMAP illustration of the data $\mathcal{X}$ using $k = 10, 15, 50, 100$ nearest neighbors: on the first row we set `min_dist` $= 0.1$ and on the second row we set `min_dist` $= 0.5$.

is that the clustering becomes more pronounced with smaller values k for the nearest neighbors considered. Smaller values of k provide detailed manifold structures (noisier structure), and for bigger k's we receive the "big picture" because detailed information within the chosen k neighbors is being lost. This is exactly what we observe in Figure 22 from the left-hand to the right-hand side. The colors of the points in the pictures show that both t-SNE and UMAP result in a similar topological picture describing the inner geometry of the considered cases.

5.3 Self-organizing map and Kohonen map

Self-organizing maps (SOM) is another dimension reduction technique that allows us to illustrate high dimensional cases in low dimensional spaces preserving part of the original topology. The

Electronic copy available at: https://ssrn.com/abstract=3439358

method goes back to Kohonen [19, 20, 21], and for this reason it is also called Kohonen map.

5.3.1 Methodology

We describe the Kohonen map. For this description we choose an explicit two dimensional example. We choose the unit cube $[0,1]^2 \subset \mathbb{R}^p$, $p = 2$, for a two dimensional illustration of high dimensional data $\mathcal{X}$. In this unit cube we choose J^2 uniformly distributed neurons labeled by $j = (j_1, j_2) \in \mathcal{J} = \{1, \ldots, J\} \times \{1, \ldots, J\}$. See Figure 23 as an example for a choice (hyperparameter) of $J = 10$. The natural topology given to this neuron space is induced by the

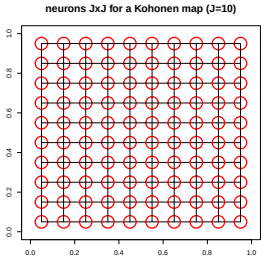

Figure 23: Uniformly distributed neurons $j \in \mathcal{J}$ in $[0,1]^2$ with $J = 10$.

squared Euclidean distance in $\mathbb{R}^2$, that is, the distance between neurons $j \in \mathcal{J}$ and $j' \in \mathcal{J}$ is given by $\|j - j'\|_2^2/J^2$ (we use the scaling $1/J^2$ because the neurons live on the unit cube).
Each of these neurons j is established with a so-called codebook $\boldsymbol{w}_j \in \mathbb{R}^q$, living in the same space as the cases $\boldsymbol{x}_i$. These codebooks are getting trained such that each neuron represents a set of similar cases $\boldsymbol{x}_i$. In this sense, we obtain a clustering to J^2 neurons of the n cases, and this clustering is done such that as much as possible of the original topology is preserved. We use a metric $d(\cdot, \cdot)$ on $\mathbb{R}^q$ to measure dissimilarities between codebooks and original features.

KOHONEN MAP ALGORITHM.

---

(0) Choose initial codebooks $\boldsymbol{w}_j^{(0)} \in \mathbb{R}^q$ for $j \in \mathcal{J}$.

(1) Repeat for $t = 1, \ldots, t_{\max}$: run sequentially through all $i \in \{1, \ldots, n\}$ and perform for each i the following steps

(a) select the best matching neuron (BMN)

$$j^* = j^*(i) = \arg\min_{j \in \mathcal{J}} d(\boldsymbol{w}_j, \boldsymbol{x}_i);$$

(b) update all codebooks $\boldsymbol{w}_j^{(t-1)}$, $j \in \mathcal{J}$, by setting

$$\boldsymbol{w}_j^{(t)} = \boldsymbol{w}_j^{(t-1)} + \theta(j^*(i), j; t)\ \alpha(t) \left(\boldsymbol{x}_i - \boldsymbol{w}_j^{(t-1)}\right). \tag{5.5}$$

Remarks.

Electronic copy available at: https://ssrn.com/abstract=3439358

- The codebooks can be initialized at random. There are also initializations that are based on PCA, which typically lead to faster convergence.

- In step (1) we select each case $\boldsymbol{x}_i$. This selection can be completely at random (sampling without replacements) or we can run sequentially through all cases. Using each case once is called an epoch.

- The BMN selects the neuron whose current codebook is most similar to the selected case $\boldsymbol{x}_i$. All codebooks are then updated w.r.t. the BMN $j^*(i)$. This is hidden in the choice of the temporal scaling θ, more explicitly, we choose kernel

$$\theta(j^*(i), j; t) = \exp\left\{-\frac{1}{2\sigma(t)^2}\left\|j - j^*(i)\right\|_2^2 / J^2\right\}, \tag{5.6}$$

  where $\sigma(t)$ is non-increasing in $t \geq 0$. In this choice of the kernel θ we see the main difference to clustering methods like K-means. In K-means we choose K cluster centers which do not have any topological relationship, i.e. these clusters are considered like unordered categorical. In SOM the neurons build a graph having a topological structure (in our case a Euclidean one), and we use this topology to learn across neighboring neurons. Neurons close to the BMN $j^*(i)$ undergo a bigger change in (5.5) in the sense that they have a bigger temporal scaling θ, and distant neurons are only marginally influenced by this update.

- There remains the choices of the temporal functions $\alpha(t)$ and $\sigma(t)$. $\alpha(t)$ acts as a learning rate and a typical choice is

$$\alpha(t) = \alpha_0 \ \frac{t_{\max} - t}{t_{\max}} \ \geq \ 0, \qquad \text{for all } t \leq t_{\max}.$$

  The bandwidth $\sigma(t)$ is often chosen as

$$\sigma(t) = \sigma_0 \ \frac{1.2t_{\max} - t}{t_{\max}} \ > \ 0, \qquad \text{for all } t \leq t_{\max}.$$

- In the Kohonen map algorithm one typically distinguishes two different learning phases:

  - *Ordering phase.* This phase tries to get the network into the right topological shape on a global scale. Depending on the choice of the bandwidth $\sigma(t)$, this phase takes roughly 1000 iterations.
  - *Convergence phase.* This phase adjusts the map locally. This second phase can be specified less precisely, and may be more expensive computationally.

- After each iteration we can calculate the total dissimilarity between the Kohonen codebooks and the original features. Therefore, we define the total dissimilarity at algorithmic time t by

$$d^{\text{total}}(t) = \sum_{i=1}^{n} d\left(\boldsymbol{w}_{j^*(i)}^{(t)}, \boldsymbol{x}_i\right),$$

  that is, we compare $\boldsymbol{x}_i$ to the codebook $\boldsymbol{w}_{j^*(i)}^{(t)}$ at time t of the BMN $j^*(i)$. If this objective function is decreasing the codebook becomes more similar to the original data. However, note that the Kohonen map algorithm does not necessarily need to be decreasing in this total dissimilarity function.

Electronic copy available at: https://ssrn.com/abstract=3439358

- Depending on the chosen data $\mathcal{X}$, different metrics $d(\cdot,\cdot)$ on $\mathbb{R}^q$ may be appropriate. The standard choice is the Euclidean distance or the squared Euclidean distance. These two choices give the same BMNs, but the decrease of the total dissimilarity will differ.

5.3.2 Illustrative examples

We provide two illustrative examples in this section to get an intuition about the functioning of the Kohonen map. These examples are taken from Chapter 3 of Kohonen [20] and the semester thesis of Dandrea [4]; we use exactly the same set-up and the same parameter values as in the semester thesis of Dandrea [4].

Illustrative Example 1. The first illustrative example is special in the sense that we choose the same dimension for the features $\boldsymbol{x}_i \in \mathbb{R}^q$ and for the neurons $j \in \mathcal{J} \subset \mathbb{R}^p$, we select $q = p = 2$. Therefore, the first example does not provide a dimension reduction, but we would like to see whether the codebooks $\boldsymbol{w}_j \in \mathbb{R}^q$ are able to rediscover the topology of the original data $\boldsymbol{x}_i \in \mathbb{R}^q$. Moreover, we choose a two dimensional example, because this allows us to illustrate both features $\boldsymbol{x}_i \in \mathbb{R}^2$ and codebooks $\boldsymbol{w}_j \in \mathbb{R}^2$. This is helpful in understanding the Kohonen map. Note that the following figures therefore always illustrate the data and approximations in the original space $\mathbb{R}^2$. Of course, in general, this is not possible because features (and codebooks) live in a high dimensional space.

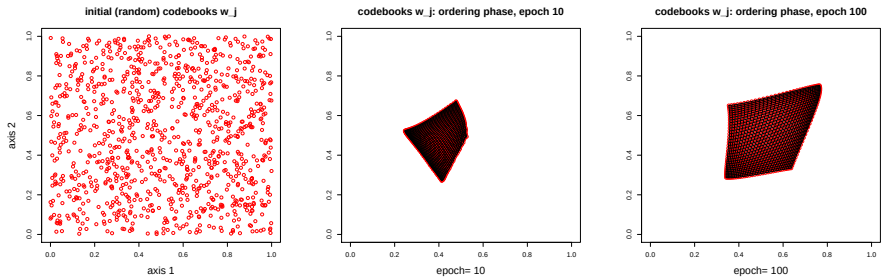

Figure 24: Case $q = p = 2$: (lhs) random initial configuration of codebooks $\boldsymbol{w}_j^{(0)} \in \mathbb{R}^2$, $j \in \mathcal{J}$, (middle, right) codebooks $\boldsymbol{w}_j^{(t)} \in \mathbb{R}^2$, $j \in \mathcal{J}$, after $t = 10$ and 100 epochs during the ordering phase.

We start by selecting the original features $\boldsymbol{x}_i \in \mathcal{X} \subset \mathbb{R}^2$. The original features $\boldsymbol{x}_i$ are chosen i.i.d. uniformly distributed on the unit cube $[0,1]^2$. Thus, the original features do not show any clustering but are uniformly spread across the unit cube. As metric $d(\cdot,\cdot)$ for the dissimilarities on the original space we choose the squared Euclidean distance function.
For the Kohonen map we choose $J^2 = 32^2$ uniformly distributed neurons on the unit cube $[0,1]^2$, this corresponds to Figure 23 with $J = 10$ replaced by $J = 32$. In total this provides us with $J^2 = 1'024$ neurons. Each of these neurons $j \in \mathcal{J}$ is initialized with an i.i.d. codebook $\boldsymbol{w}_j^{(0)}$ being uniformly distributed on the unit cube. These 1'024 randomly initialized codebooks $\boldsymbol{w}_j^{(0)} \in [0,1]^2$ are illustrated in Figure 24 (lhs).
We divide step (1) of the Kohonen map algorithm into the ordering phase and the convergence phase. For these two phases we choose different hyperparameters. For the ordering phase we

Electronic copy available at: https://ssrn.com/abstract=3439358

select

$$t_{\max} = 100, \qquad \sigma_0 = 0.5 \qquad \text{and} \qquad \alpha_0 = 0.8,$$

and for each algorithmic step $t = 1, \ldots, t_{\max}$ we only consider one case $\boldsymbol{x}_i \in [0,1]^2$. Under this assumption we need to have $n = t_{\max} = 100$ i.i.d. cases $\boldsymbol{x}_i$ for the ordering phase.
Figure 24 (middle, rhs) shows the updated codebooks $\boldsymbol{w}_j^{(t)}$ at algorithmic times $t = 10$ and 100; note that the black lines show the topology of the neurons in the neuron space $\mathcal{J} \subset \mathbb{R}^2$. We observe that the randomly initialized codebooks are quickly ordered within the first 10 steps, and then this ordering is spread more widely in a uniform way in the remaining steps. This reflects that the underlying (original) data $\boldsymbol{x}_i$ are uniformly spread across the unit cube.

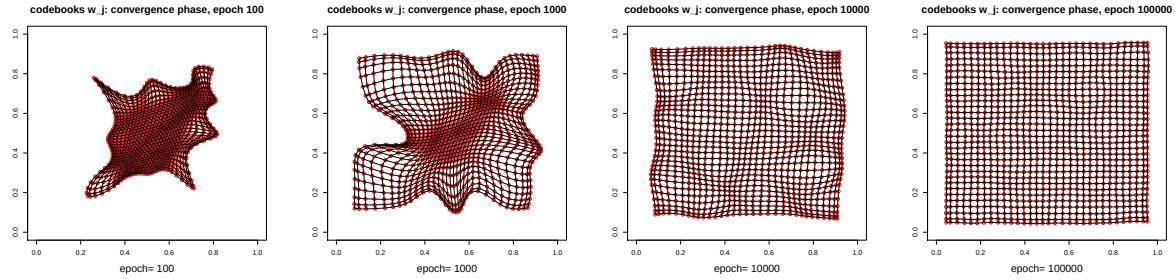

Figure 25: Case $q = p = 2$: codebooks $\boldsymbol{w}_j^{(t)} \in \mathbb{R}^2$, $j \in \mathcal{J}$, after $t = 100, 1'000, 10'000, 100'000$ epochs during the convergence phase.

The next phase is the convergence phase. We use the resulting codebooks of the ordering phase as initial codebooks for the convergence phase. The choice of the hyperparameters for the convergence phase is as follows

$$t_{\max} = 100'000, \qquad \sigma_0 = 0.005 \qquad \text{and} \qquad \alpha_0 = 0.1,$$

and for each algorithmic step t we only consider one case $\boldsymbol{x}_i \in [0,1]^2$. Under this assumption we need to have $n = t_{\max} = 100'000$ i.i.d. cases $\boldsymbol{x}_i$ for the convergence phase.
Figure 25 shows the codebooks $\boldsymbol{w}_j^{(t)}$ during the convergence phase for algorithmic times $t = 100, 1'000, 10'000, 100'000$, the black lines again illustrate the topology in the neuron space $\mathcal{J} \subset \mathbb{R}^2$. We observe that the codebooks converge to a uniform grid on the unit cube $[0,1]^2$. Of course, this makes perfect sense because this uniform grid reflects the topology obtained by uniformly distributed i.i.d. samples $\boldsymbol{x}_i$ on that unit cube. We conclude that the Kohonen map is able to rediscover the right topology in our example, where the dimensions q of the feature space and p of the neuron space are identical.

Illustrative Example 2. In the second illustrative example we consider a real dimension reduction problem. The features $\boldsymbol{x}_i \in \mathbb{R}^q$ live in a $q = 2$ dimensional space, and the neurons are assumed to live in a one dimensional space, i.e. $j \in \mathcal{J} = \{1, \ldots, J\} \subset \mathbb{R}^p$ with $p = 1$. Again, for $q = 2$ we can illustrate original features and codebooks which is useful in analyzing the functioning of the Kohonen map.
The original features $\boldsymbol{x}_i \in \mathcal{X} \subset \mathbb{R}^2$ are again chosen to be i.i.d. uniformly distributed on the unit cube $[0,1]^2$. As metric $d(\cdot,\cdot)$ for the dissimilarities on the original space we choose the squared Euclidean distance function.

Electronic copy available at: https://ssrn.com/abstract=3439358

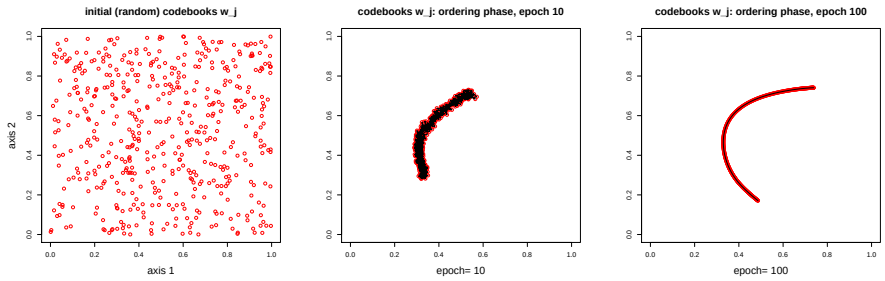

Figure 26: Case $q = 2$ and $p = 1$: (lhs) random initial configuration of codebooks $\boldsymbol{w}_j^{(0)} \in \mathbb{R}^2$, $j \in \mathcal{J}$, (middle, right) codebooks $\boldsymbol{w}_j^{(t)} \in \mathbb{R}^2$, $j \in \mathcal{J}$, after $t = 10$ and 100 epochs during the ordering phase.

For the one dimensional Kohonen map we choose $J = 500$ uniformly distributed neurons in the unit interval $[0, 1]$, i.e. this just provides equidistant points on the unit interval. Each of these neurons $j \in \mathcal{J} = \{1, \ldots, J\}$ is initialized with an i.i.d. codebook $\boldsymbol{w}_j^{(0)}$ being uniformly distributed on the unit cube. These 500 randomly initialized codebooks $\boldsymbol{w}_j^{(0)} \in [0, 1]^2$ are illustrated in Figure 26 (lhs).
Again, we divide step (1) of the Kohonen map algorithm into the ordering phase and the convergence phase. For these two phases we choose different hyperparameters. For the ordering phase we select

$$t_{\max} = 100, \qquad \sigma_0 = 0.1 \qquad \text{and} \qquad \alpha_0 = 0.5,$$

and for each algorithmic step t we only consider one case $\boldsymbol{x}_i \in [0, 1]^2$. Under this assumption we need to have $n = t_{\max} = 100$ i.i.d. cases $\boldsymbol{x}_i$ for the ordering phase.
Figure 26 (middle, rhs) shows the updated codebooks $\boldsymbol{w}_j^{(t)}$ at algorithmic times $t = 10$ and 100; the black lines show the topology of the neurons in the one dimensional neuron space $\mathcal{J} \subset \mathbb{R}$. We observe that the randomly initialized codebooks are quickly ordered within the first 100 steps, trying to reflect a uniform distribution on the unit cube $[0, 1]^2$ by a one dimensional object (approximation).

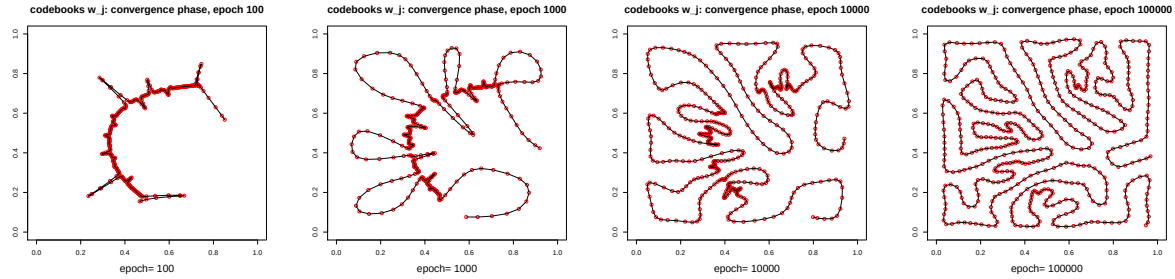

Figure 27: Case $q = 2$ and $p = 1$: codebooks $\boldsymbol{w}_j^{(t)} \in \mathbb{R}^2$, $j \in \mathcal{J}$, after $t = 100, 1'000, 10'000, 100'000$ epochs during the convergence phase.

The next phase is the convergence phase. We use the resulting codebooks of the ordering phase as initial codebooks for the convergence phase. The choice of the hyperparameters for the

Electronic copy available at: https://ssrn.com/abstract=3439358

convergence phase are as follows

$$t_{\max} = 100'000, \qquad \sigma_0 = 0.00001 \qquad \text{and} \qquad \alpha_0 = 0.1,$$

and for each algorithmic step t we only consider one case $\boldsymbol{x}_i \in [0,1]^2$. Under this assumption we need to have $n = t_{\max} = 100'000$ i.i.d. cases $\boldsymbol{x}_i$ for the convergence phase.
Figure 27 shows the codebooks $\boldsymbol{w}_j^{(t)}$ during the convergence phase for algorithmic times $t = 100, 1'000, 10'000, 100'000$, the black lines again illustrate the topology in the one dimensional neuron space $\mathcal{J} \subset \mathbb{R}$. We observe that the codebooks converge to a line in the unit cube $[0,1]^2$ that tries to cover the entire unit cube in a uniform way (Peano curve). This reflects that the lower dimensional object tries to capture as much of the topology induced by the uniform distribution on the unit cube. This way we can capture some neighboring relationships, but of course not all of them, because the dimension is reduced from $q = 2$ down to $p = 1$.

5.3.3 Example, revisited

We revisit the car models example of Listing 1. In Listing 12 we provide the R code to perform

Listing 12: SOM code in R using the `kohonen` library

```
library(kohonen)

set.seed(100)
som.X <- som(as.matrix(X), grid = somgrid(xdim=10, ydim=10, topo="rectangular"),
                           rlen= 100, dist.fcts="sumofsquares")

summary(som.X)
predict(som.X)$unit.classif     # allocation to neurons
plot(som.X,c("changes"))        # training progress
plot(som.X,c("counts"))         # number of allocated neurons
```

the Kohonen map algorithm using the `kohonen` library of R. We choose $J = 10$, $\mathcal{J} = \{1, \ldots, J\} \times \{1, \ldots, J\} \subset \mathbb{R}^2$ and the squared Euclidean distance in $\mathbb{R}^q$. This algorithm needs a seed and correspondingly the solution will depend on the choice of this seed.

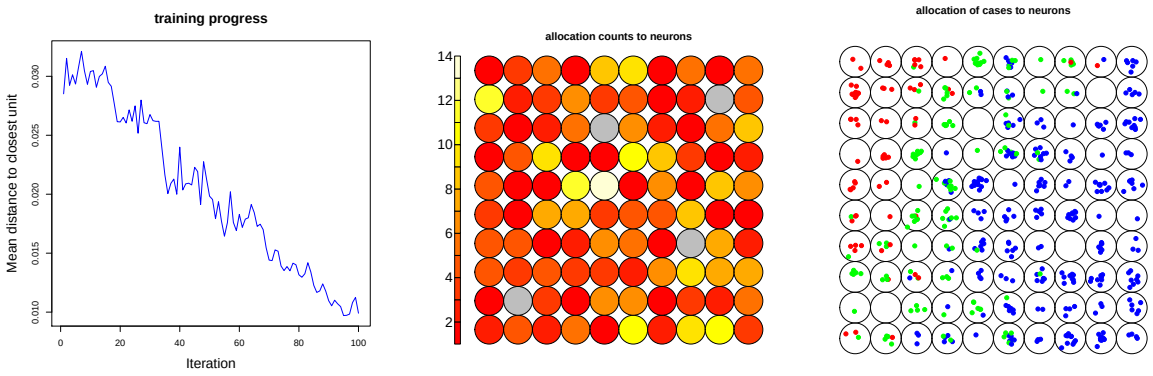

Figure 28: SOM illustration of the data $\mathcal{X}$: (lhs) decrease of loss over training iterations, (middle) number of cases allocated to each neuron, (rhs) cases allocation to neurons.

Electronic copy available at: https://ssrn.com/abstract=3439358

The resulting Kohonen map is illustrated in Figure 28. The left-hand side shows the mean distance between the cases $\boldsymbol{x}_i$ and the BMN's codebooks $\boldsymbol{w}^{(t)}_{j^*(i)}$ over the different training iterations $t \geq 1$ (epochs). In general, this function is not monotone because the Kohonen map algorithm does not provide a monotonically decreasing optimization algorithm, but it should be decreasing in average over a rolling time window, otherwise the codebooks of the neurons do not learn the topology of the feature space.
The middle picture of Figure 28 shows the number of cases allocated to each neuron. We note that not each neuron receives at least one case. This suggests to also study other neuron spaces $\mathcal{J} \subset \mathbb{R}^2$. The right-hand side of Figure 28 illustrates these cases with red color for sports cars $\tau < 17$, green color for cars with $\tau \in [17, 21)$ and blue color for the cars with $\tau \geq 21$.

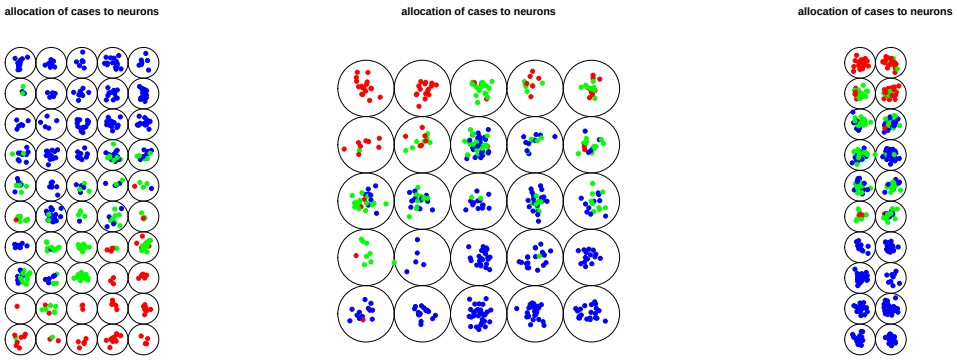

Figure 29: SOM to neurons for different neuron spaces $\mathcal{J}$: (lhs) $\mathcal{J} = \{1, \ldots, 5\} \times \{1, \ldots, 10\}$, (middle) $\mathcal{J} = \{1, \ldots, 5\} \times \{1, \ldots, 5\}$, and (rhs) $\mathcal{J} = \{1, \ldots, 2\} \times \{1, \ldots, 10\}$.

In Figure 29 we illustrate the Kohonen maps for different neuron spaces (lhs) $\mathcal{J} = \{1, \ldots, 5\} \times \{1, \ldots, 10\}$, (middle) $\mathcal{J} = \{1, \ldots, 5\} \times \{1, \ldots, 5\}$, and (rhs) $\mathcal{J} = \{1, \ldots, 2\} \times \{1, \ldots, 10\}$. If we consider the last Kohonen map (right-hand side of Figure 29), we are tempted to define the sports cars to be those cars whose BMNs are in the upper two rows of that map. Most of the dots that are allocated to these neurons have a red color, which implies (bottom line) that the judgment of the Kohonen map is quite similar to the Belgium expert choice. On the other hand, the example also shows that there are quite some similarities between the different neurons, for this reason one should explore the individual neurons in more detail before making conclusions.

6 Categorical variables

The mindful reader will have noticed that we have avoided any discussion about categorical variables, so far. All feature components chosen above are continuous (or at least ordered), and therefore they are well-suited for Euclidean distances, Manhattan distances, Gaussian kernels, etc. The treatment of *nominal categorical* variables is more difficult, i.e. how can we measure distances, for instance, between car brands, models or colors? In this section we present and discuss possible approaches for the treatment of categorical variables, but we refrain from giving explicit examples.

Electronic copy available at: https://ssrn.com/abstract=3439358

6.1 Univariate categorical variables

Assume that x is a nominal categorical feature with labels in $\mathcal{L}$.[18] We denote by $L = |\mathcal{L}|$ the number of different labels. The most natural dissimilarity measure between categorical labels $x, x' \in \mathcal{L}$ is to check whether they are different, i.e.

$$d(x, x') = \mathbb{1}_{\{x \neq x'\}}. \tag{6.1}$$

This distance measure reflects that each label $x \in \mathcal{L}$ is mapped to its own unit vector in $\mathbb{R}^L$. Thus, we consider the corners $\mathcal{S}_L^\circ$ of the $(L-1)$-unit simplex $\mathcal{S}_L$, see (4.8). These corners describe one-hot encoding of categorical variables and each label $x \in \mathcal{L}$ has the same distance from any other label $x' \neq x \in \mathcal{L}$.
Eskin et al. [6] proposed to cushion dissimilarity by accounting for a weight factor of $1/L^2$ for mismatches

$$d(x, x') = \frac{1}{L^2}\, \mathbb{1}_{\{x \neq x'\}}. \tag{6.2}$$

Of course, this weight factor is only of relevance if we have multiple categorical features having different numbers L of labels.
Another possible differentiation is done be weighting the dissimilarity measure (6.1) with the number of occurrences of the labels in the data. Basically, we move from the corners $\mathcal{S}_L^\circ$ of the $(L-1)$-unit simplex $\mathcal{S}_L$ closer to the origin. Assume that label $x \in \mathcal{L}$ occurs n_x times in the data set of size n. We define the categorical probability in the $(L-1)$-unit simplex

$$\boldsymbol{p} = (p_x)_{x \in \mathcal{L}} = \Big(\frac{n_x}{n}\Big)_{x \in \mathcal{L}} \;\in\; \mathcal{S}_L.$$

This then allows us to define the probability weighted dissimilarity

$$d(x, x') = p_x p_{x'} \mathbb{1}_{\{x \neq x'\}}. \tag{6.3}$$

If all labels are equally likely, i.e. $p_x \equiv 1/L$, we have (6.2), and otherwise a mismatch $x \neq x'$ is more heavily punished for more likely labels $x, x' \in \mathcal{L}$.

6.2 Multiple categorical variables

For multiple categorical variables we can benefit from the structure of contingency tables to construct dissimilarity measures. Let us consider the special case of two categorical feature components here, having L_1 and L_2 labels, respectively. This allows us to consider an $L_1 \times L_2$ contingency table with entries

$$p_{\boldsymbol{x}} = \frac{n_{\boldsymbol{x}}}{n} \;\in\; [0, 1] \qquad \text{for } \; \boldsymbol{x} = (x_1, x_2) \in \mathcal{L} = \mathcal{L}_1 \times \mathcal{L}_2,$$

where $\mathcal{L}_1$ and $\mathcal{L}_2$ denote the labels of the first and second categorical feature component, respectively, and $n_{\boldsymbol{x}}$ denotes the number of occurrences of $\boldsymbol{x} \in \mathcal{L}$ among the n data points. The row marginals and the column marginals are defined by

$$p_{x_1} = \sum_{z_2 \in \mathcal{L}_2} p_{(x_1, z_2)} \qquad \text{and} \qquad p_{x_2} = \sum_{z_1 \in \mathcal{L}_1} p_{(z_1, x_2)},$$

[18] We call nominal categorical feature values $x \in \mathcal{L}$ either labels or levels.

Electronic copy available at: https://ssrn.com/abstract=3439358

for $x_1 \in \mathcal{L}_1$ and $x_2 \in \mathcal{L}_2$, respectively. Thus, we have row and column normalizations

$$1 = \sum_{z_2 \in \mathcal{L}_2} \frac{p_{(x_1,z_2)}}{p_{x_1}} = \sum_{z_1 \in \mathcal{L}_1} \frac{p_{(z_1,x_2)}}{p_{x_2}}.$$

This allows us to consider the conditional probabilities, given x_1 and x_2, respectively,

$$\begin{aligned} p_{z_2|x_1} &= \frac{p_{(x_1,z_2)}}{p_{x_1}} \qquad \text{for } z_2 \in \mathcal{L}_2, \\ p_{z_1|x_2} &= \frac{p_{(z_1,x_2)}}{p_{x_2}} \qquad \text{for } z_1 \in \mathcal{L}_1. \end{aligned}$$

6.2.1 χ^2-test

A special role among all probability distributions $(p_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{L}}$ is played by the one which allocates the labels $x_1 \in \mathcal{L}_1$ and $x_2 \in \mathcal{L}_2$ independently from each other. For given marginals $(p_{x_1})_{x_1\in\mathcal{L}_1}$ and $(p_{x_2})_{x_2\in\mathcal{L}_2}$ this special distribution is given by

$$\pi_{\boldsymbol{x}} = p_{x_1} p_{x_2} \qquad \text{for } \boldsymbol{x} = (x_1, x_2) \in \mathcal{L}.$$

Obviously, $(\pi_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{L}}$ has marginals $(p_{x_1})_{x_1\in\mathcal{L}_1}$ and $(p_{x_2})_{x_2\in\mathcal{L}_2}$, and the conditional probability of observing one label does not depend on the other label under $(\pi_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{L}}$.
The χ^2-test is a statistical test that allows us to verify whether given observations $(n_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{L}}$ may have been generated by a distribution having independence between the components in $\boldsymbol{x}$. We define the test statistics

$$\chi^2 = \sum_{\boldsymbol{x}\in\mathcal{L}} \frac{(p_{\boldsymbol{x}} - \pi_{\boldsymbol{x}})^2}{\pi_{\boldsymbol{x}}}.$$

The test statistics χ^2 measures the amount of independence in $(p_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{L}}$. This test statistics has its roots in the binomial modeling of $\boldsymbol{x} = (x_1, x_2)$, and is based on the central limit theorem

$$\frac{p_{\boldsymbol{x}} - \pi_{\boldsymbol{x}}}{\sqrt{\pi_{\boldsymbol{x}}(1 - \pi_{\boldsymbol{x}})}} \overset{\text{(d)}}{\Longrightarrow} \mathcal{N}(0, 1), \tag{6.4}$$

as $n_{\boldsymbol{x}} \to \infty$, and under the null hypothesis that $(p_{\boldsymbol{x}})_{\boldsymbol{x}\in\mathcal{L}}$ comes from independent marginals. The square of the right hand side can further be modified to

$$\begin{aligned} \frac{(p_{\boldsymbol{x}} - \pi_{\boldsymbol{x}})^2}{\pi_{\boldsymbol{x}}(1 - \pi_{\boldsymbol{x}})} &= \frac{(p_{\boldsymbol{x}} - \pi_{\boldsymbol{x}})^2}{\pi_{\boldsymbol{x}}} + \frac{(p_{\boldsymbol{x}} - \pi_{\boldsymbol{x}})^2}{1 - \pi_{\boldsymbol{x}}} \\ &= \frac{(p_{\boldsymbol{x}} - \pi_{\boldsymbol{x}})^2}{\pi_{\boldsymbol{x}}} + \frac{((1 - p_{\boldsymbol{x}}) - (1 - \pi_{\boldsymbol{x}}))^2}{1 - \pi_{\boldsymbol{x}}}, \end{aligned} \tag{6.5}$$

which is approximately χ^2-distributed under the null hypothesis. Summing up over all $\boldsymbol{x}$ and omitting the second term in (6.5) (of the complementary probabilities), we get the χ^2-test of independence. For more details we refer to [1] and [25].

6.2.2 Euclidean distance

The χ^2-test of independence provides us with the natural idea to compare the probability $p_{\boldsymbol{x}}$ to its independent counterpart $\pi_{\boldsymbol{x}}$. To do so, we may simply choose the Euclidean distance (as in the χ^2-test), but any other distance function or even divergence will also do, see pages 375-379

Electronic copy available at: https://ssrn.com/abstract=3439358

in Mardia et al. [23] for particular examples of other distance functions. We define the Euclidean distance measure between $x_1, x_1' \in \mathcal{L}_1$ by

$$d(x_1, x_1') = \sqrt{\sum_{z_2 \in \mathcal{L}_2} \left(\frac{p_{(x_1,z_2)}}{p_{x_1}} - \frac{p_{(x_1',z_2)}}{p_{x_1'}} \right)^2} = \sqrt{\sum_{z_2 \in \mathcal{L}_2} \left(p_{z_2|x_1} - p_{z_2|x_1'} \right)^2}, \tag{6.6}$$

and similarly for $x_2, x_2' \in \mathcal{L}_2$

$$d(x_2, x_2') = \sqrt{\sum_{z_1 \in \mathcal{L}_1} \left(\frac{p_{(z_1,x_2)}}{p_{x_2}} - \frac{p_{(z_1,x_2')}}{p_{x_2'}} \right)^2} = \sqrt{\sum_{z_1 \in \mathcal{L}_1} \left(p_{z_1|x_2} - p_{z_1|x_2'} \right)^2}.$$

If the distance $d(x_1, x_1') = 0$ in (6.6), then we have that the conditional probability vectors of the second component satisfy, given x_1 and x_1', respectively,

$$\left(p_{z_2|x_1}\right)_{z_2 \in \mathcal{L}_2} = \left(p_{z_2|x_1'}\right)_{z_2 \in \mathcal{L}_2},$$

i.e. the distribution of the second label $z_2 \in \mathcal{L}_2$ does not depend on the fact whether we have observed x_1 or x_1' in the first label.

If $d(x_1, x_1') = 0$ for all $x_1, x_1' \in \mathcal{L}_1$, this implies that $p_{\boldsymbol{x}} = p_{x_1} p_{x_2} = \pi_{\boldsymbol{x}}$ for all $\boldsymbol{x} \in \mathcal{L}$. This brings us back to the χ^2-test, namely, that if the distances between all $x_1, x_1' \in \mathcal{L}_1$ are zero we are in the case of independence from marginals. This also says that if we want to discriminate a component $x_1 \in \mathcal{L}_1$ as described above, we need to have a contingency table that does not stem from independent marginals, otherwise x_1 does not have explanatory power for x_2, and $x_2 \in \mathcal{L}_2$ is not helpful in discriminating $x_1 \in \mathcal{L}_1$.

Acknowledgment. We would like to kindly thank Jürg Schelldorfer (Swiss Re) for his detailed comments that have helped us to substantially improve this tutorial, and, in particular, for his Figure 1.

References

[1] Benhamou, E., Melot, V. (2018). Seven proofs of the Pearson Chi-squared independence test and its graphical interpretation. *arXiv*:1808.09171

[2] Burt, C. (1950). The factorial analysis of qualitative data. *British Journal of Mathematical and Statistical Psychology* **3**, 166-185.

[3] Croux, C., Filzmoser,P., Oliveira, M. (2007). Algorithms for projection pursuit robust principal component analysis. *Chemometrics and Intelligent Laboratory Systems* **87**, 218-225.

[4] Dandrea, D. (2018). Self-organizing maps applied to car insurance. *Semester Thesis, ETH Zurich*, Spring term 2018.

[5] Efron, B., Hastie, T. (2016). *Computer Age Statistical Inference*. Cambridge University Press.

[6] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In: *Applications of Data Mining in Computer Security*, Barbara, D., Jajodia, S. (eds.), Kluwer Academic Publisher, 78-100.

Electronic copy available at: https://ssrn.com/abstract=3439358

[7] Ferrario, A., Noll, A., Wüthrich, M.V. (2018). Insights from inside neural networks. *SSRN Manuscript* ID 3226852. Version November 14, 2018.

[8] Fisher, J. (2004). Visualizing the connection among convex hull, Voronoi diagram and Delaunay triangulation. In: *37th Midwest Instruction and Computing Symposium*, SemanticScholar.

[9] Fraley, C., Raftery, A.E. (2003). `MCLUST`: Software for model-based clustering, density estimation and discriminant analysis. Technical Report No. 415, University of Washington.

[10] Golub, G., Van Loan, C. (1983). *Matrix Computations*. John Hopkins University Press.

[11] Hastie, T., Tibshirani, R., Friedman, J. (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction.* 2nd edition. Springer Series in Statistics.

[12] Hinton, G.E., Salakhutdinov, R.R. (2006). Reducing the dimensionality of data with neural networks. *Science* **313**, 504-507.

[13] Hothorn, T., Everitt, B.S. (2014). *A Handbook of Statistical Analyses using R.* 3rd edition. CRC Press.

[14] Ingenbleek, J.-F., Lemaire, J. (1988). What is a sports car? *ASTIN Bulletin* **18/2**, 175-187.

[15] James, G., Witten, D., Hastie, T., Tibshirani, R. (2015). *An Introduction to Statistical Learning. With Applications in R.* Corrected 6th printing. Springer Texts in Statistics.

[16] Kaufman, L., Rousseeuw, P.J. (1987). Clustering by means of medoids. In: *Statistical Data Analysis Based on the L_1 Norm and Related Methods*, Y. Dodge (ed.), North-Holland, 405-416.

[17] Kaufman, L., Rousseeuw, P.J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons.

[18] Kingma, D.P., Welling, M. (2014). Auto-encoding variational Bayes. *arXiv*:1312.6114v10.

[19] Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics* **43**, 59-69.

[20] Kohonen, T. (2001). *Self-Organizing Maps*. 3rd edition. Springer.

[21] Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks* **37**, 52-65.

[22] Kramer, M.A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* **37/2**, 233-243.

[23] Mardia, K.V., Kent, J.T., Bibby, J.M. (1979). *Multivariate Analysis*. Academic Press.

[24] McInnes, L., Healy, J., Melville, J. (2018). UMAP: uniform manifold approximation and projection for dimension reduction. *arXiv*:1802.03426v2.

[25] Prillaman, J. (1956). *The Derivation of the Chi-Square Test of Goodness of Fit*, McGill University, Master of Science.

[26] Richman, R. (2018). AI in actuarial science. *SSRN Manuscript* ID 3218082, Version August 20, 2018.

[27] Schelldorfer, J., Wüthrich, M.V. (2019). Nesting classical actuarial models into neural networks. *SSRN Manuscript* ID 3320525.

[28] Schubert, E., Rousseeuw, P.J. (2019). Faster k-medoids clustering: improving the PAM, CLARA, and CLARANS algorithms. *arXiv*:1810.05691v3.

[29] Stahel, W. (2011). *Applied Multivariate Statistics*. ETH Zurich, Lecture Notes. https://stat.ethz.ch/∼stahel/courses/multivariate/script/

Electronic copy available at: https://ssrn.com/abstract=3439358

[30] van der Maaten, L.J.P., Hinton, G.E. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research* **9**, 2579-2605.

[31] Vathy-Fogarassy, A., Abonyi, J. (2013). *Graph-Based Clustering and Data Visualization Algorithms*. Springer.

[32] Wüthrich, M.V., Buser, C. (2016). Data analytics for non-life insurance pricing. *SSRN Manuscript* ID 2870308. Version June 5, 2019.

Electronic copy available at: https://ssrn.com/abstract=3439358