Factor Analysis: PCA vs Autoencoder

Verena Brufatto

August 7, 2021

Contents

1 Introduction

2 Data preprocessing

The following analysis is based on the Boston Housing dataset, which contains data on the real estate market of Boston, Massacchusetts, collected in 1978. The dataset contains 506 observations of the following variables:

- *crim*: per capita crime rate by town
- *zn*: proportion of residential land zoned for lots over 25,000 sq.ft
- *indus*: proportion of non-retail business acres per town
- *chas*: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- *nox*: nitric oxides concentration (parts per 10 million)
- *rm*: average number of rooms per dwelling
- *age*: proportion of owner-occupied units built prior to 1940
- *dis*: weighted distances to five Boston employment centres
- *rad*: index of accessibility to radial highways
- *tax*: full-value property-tax rate per USD 10,000
- *ptratio*: pupil-teacher ratio by town
- *black*: 1000(B - 0.63)^2 where B is the proportion of blacks by town
- *lstat*: percentage of lower status of the population
- *medv*: median value of owner-occupied homes in USD 1000's

The target variables is *medv*, while the others can be used as explanatory variables. Table 1 and Figures 1-2 show that the feature *black* has up to 15% of outliers, while *zn* and *crime* have about 13%. Figure 3 shows that the features that are most correlated with *medv* are *lstat*, *ptratio*, *tax*, *rm*, *nox*, *indus*.

Table 1: Percentage of outliers

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
13	13.4	0	6.9	0	5.9	0	1	0	0	3	15	1.2	7.3

Figura 1: Boxplot of features

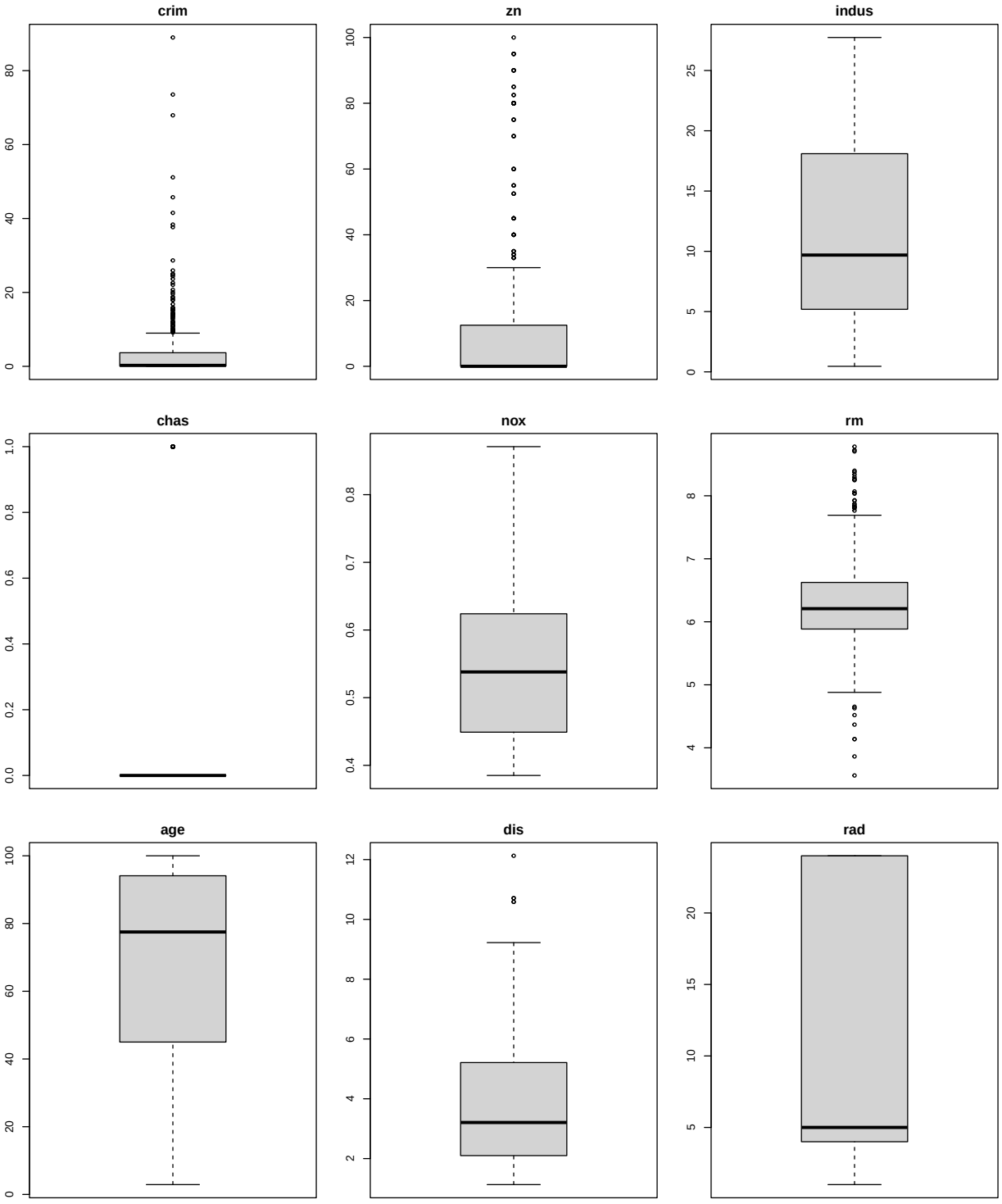


Figura 2: Boxplot of features

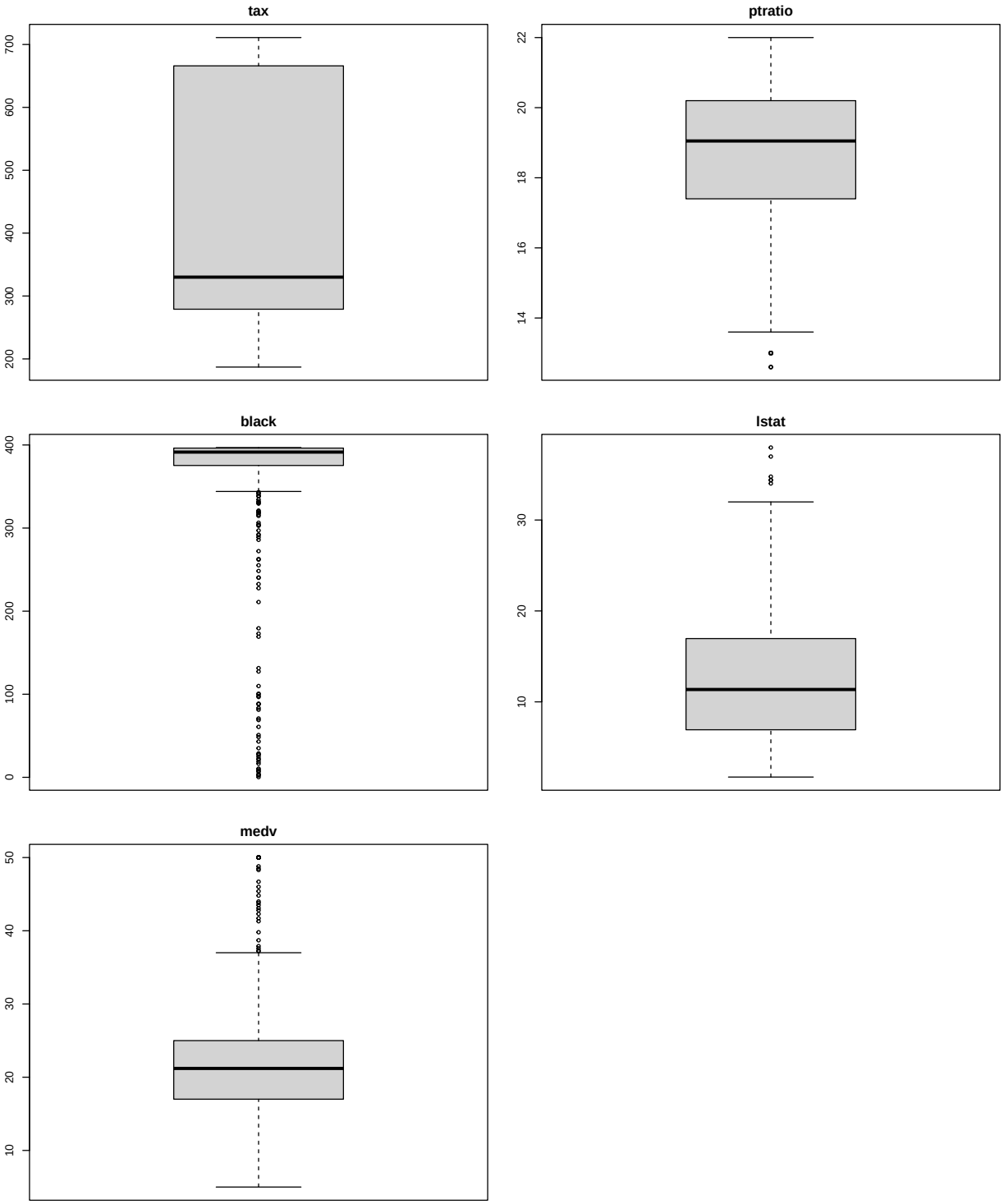
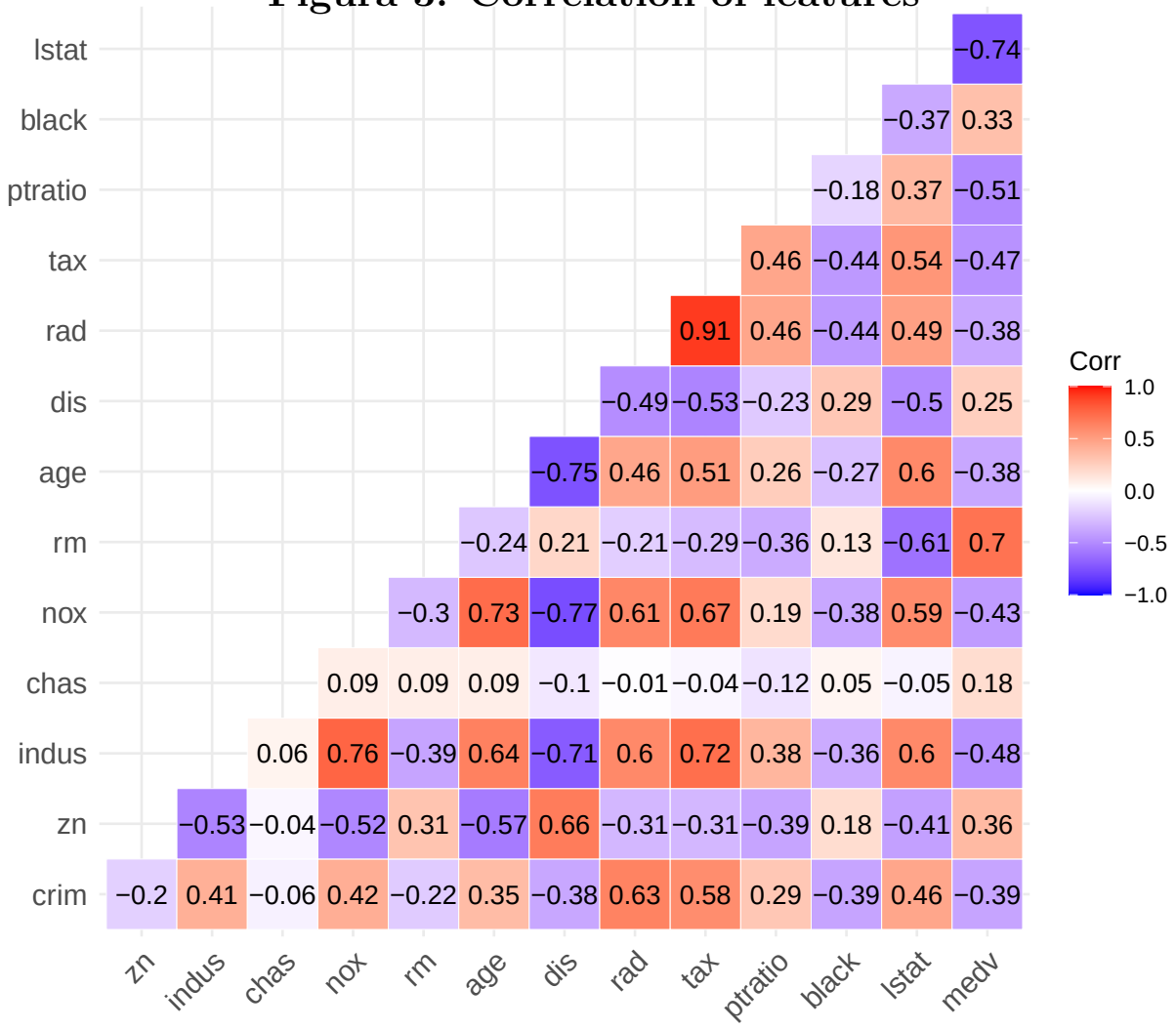


Figura 3: Correlation of features



For comparison with the correlation coefficients, we use the R function `ols_step_all_possible` from the package `olsrr` to test all possible subsets of the set of potential explanatory variables. We then select the optimal model specification based on the Bayesian-Shwarz (BIC) information criterion, defined as:

$$BIC = kln(n) - 2ln(\hat{L})$$

where k is the number of estimated parameters, n the number of observations in the dataset and $\hat{L} = p(x|\hat{\theta}, M)$ the maximum likelihood estimator.

```
regression = lm(medv ~ ., data = boston)
selection = ols_step_all_possible(regression)
bic_selection = data.frame( selection[ which.min(selection$sbic),] )
predictors = strsplit(bic_selection$predictors, " +")[[1]]
```

According to the information criterion, the optimal set of features includes *crim*, *zn*, *chas*, *nox*, *rm*, *dis*, *rad*, *tax*, *ptratio*, *black* and *lstat*. Since the variable *indus* is not included in the selection performed by BIC, we omit it from the set of explanatory variables and use the remaining features for the rest of the analysis, since they also exhibit a very low number of outliers.
We furthermore choose to adopt the same feature transformations described in the paper by Harrison and Rubinfeld (1978), from which the Boston dataset stems. The authors apply a logarithmic transformation to the variables *medv* and *lstat* and a quadratic transformation to the variables *nox* and *rm*. Figure 4 show the distribution of the new features, while Figure 5 shows a comparison of the empirical marginal densities of the features to a Gaussian approximation (the Q-Q plot). For the features *medv*, *lstat* and *rm*, the Gaussian approximation works reasonably well, while for the others it does not look reasonable. This fact may be relevant for the subsequent parts of this analysis due to the use of Euclidean distance in the objective function, which is the standard choice for Gaussian random variables.

Figura 4: Histogram of features

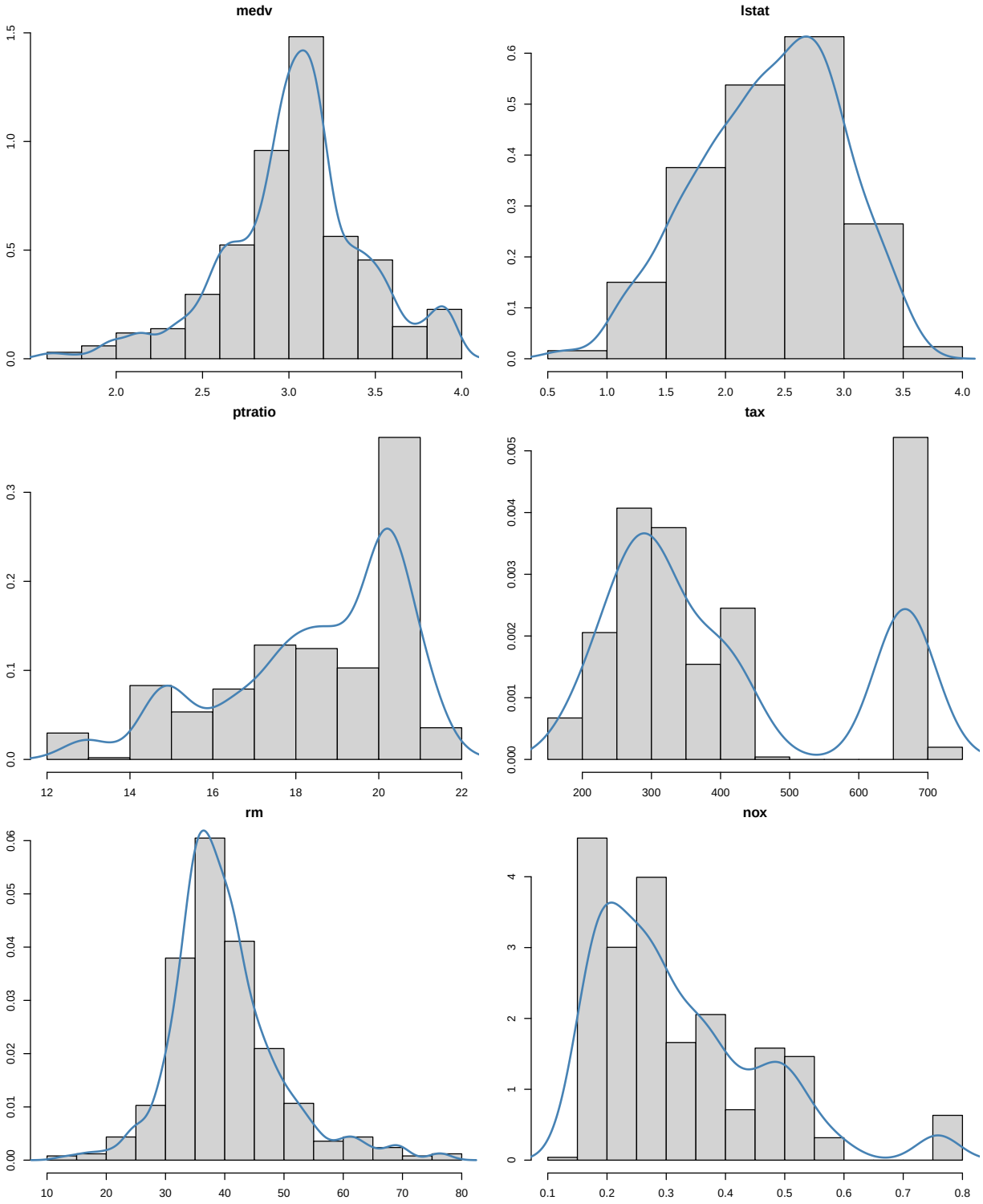
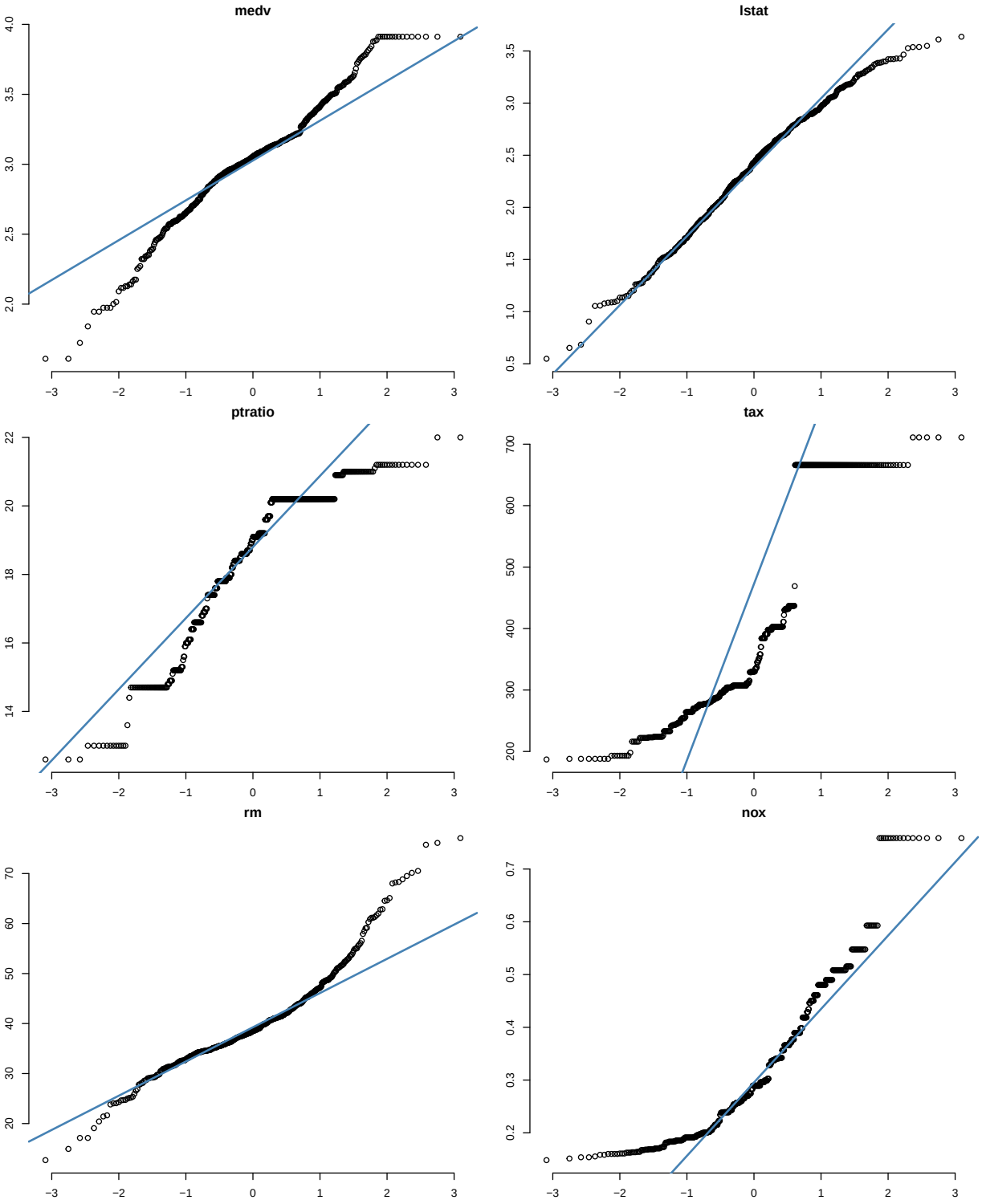


Figura 5: Q-Q plot



3 Principal component analysis

Principal component analysis (PCA) aims at reducing the dimension of a dataset while preserving the original information by minimizing the recostruction error.

In order to apply PCA, we need to standardize the features $X = (x_1, ..., x_n) \in \mathbb{R}^{n \times q}$. PCA finds an orthonormal basis $v_1, .., v_q$ that spans the space $\mathbb{R}^q$ such that v_1 explains the direction of the largest heterogeneity in X, v_2 the direction of the second largest heterogeneity orthogonal to v_1 and so on.

The goal is to find a lower dimensional approximation $Y = (y_1, ..., y_n) \in \mathbb{R}^p$ of X for which $p \leq q$ and such that

$$y_{i,j} = v_{j,1} x_{i,1} +, ..., + v_{j,q} x_{i,q}$$

for $i = 1, ..., n$ and $j = 1, .., q$.

The first and second weights are determined as

$$v_1 = \underset{||\omega^T \omega|| = 1}{argmax} \left(\omega^T X^T X \omega \right)$$

and

$$v_2 = \underset{||\omega^T \omega|| = 1}{argmax} \left(\omega^T X^T X \omega \right), \qquad \langle v_1, \omega \rangle = 0$$

Therefore, the j-th weight is given by

$$v_j = \underset{||\omega^T \omega|| = 1}{argmax} \left(\omega^T X^T X \omega \right), \qquad \langle v_l, \omega \rangle = 0, \forall 1 \leq l \leq j - 1$$

The matrix $A = X^T X$ is positive definite and symmetric by assumption, so that the orthonormal basis $v_1, ..., v_q$ is given by the ordered eigenvalues of A.

```
dati_norm = scale(dataset[, -1])
pca = prcomp(dati_norm, center = F, scale = F)
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5
## Standard deviation     1.6620 0.9723 0.8811 0.52636 0.48881
## Proportion of Variance 0.5525 0.1891 0.1553 0.05541 0.04779
## Cumulative Proportion  0.5525 0.7415 0.8968 0.95221 1.00000
```

```
pca$rotation
```

```
##                PC1        PC2        PC3        PC4        PC5
## lstat    0.5229044 -0.1160902  0.3076645 -0.5296231  0.5813226
## ptratio  0.3697382 -0.4683430 -0.6934675 -0.2673204 -0.3026402
## tax      0.4761593  0.3676301 -0.3868548  0.5848317  0.3826692
## rm      -0.4187016  0.4996117 -0.4976821 -0.4927123  0.2909025
## nox      0.4333889  0.6183964  0.1646248 -0.2514869 -0.5825916
```

The first two principal components explain about 75% of the total variation (or in this case correlation, since the data is standardized), meaning that most of the variability in X can be explained by v_1 and v_2.

Figure 6 shows the Q-Q plots of the principal components against a Gaussian distribution. For some of the principal components, a Gaussian assumption is questionable.

Figura 6: Q-Q plot of principal components

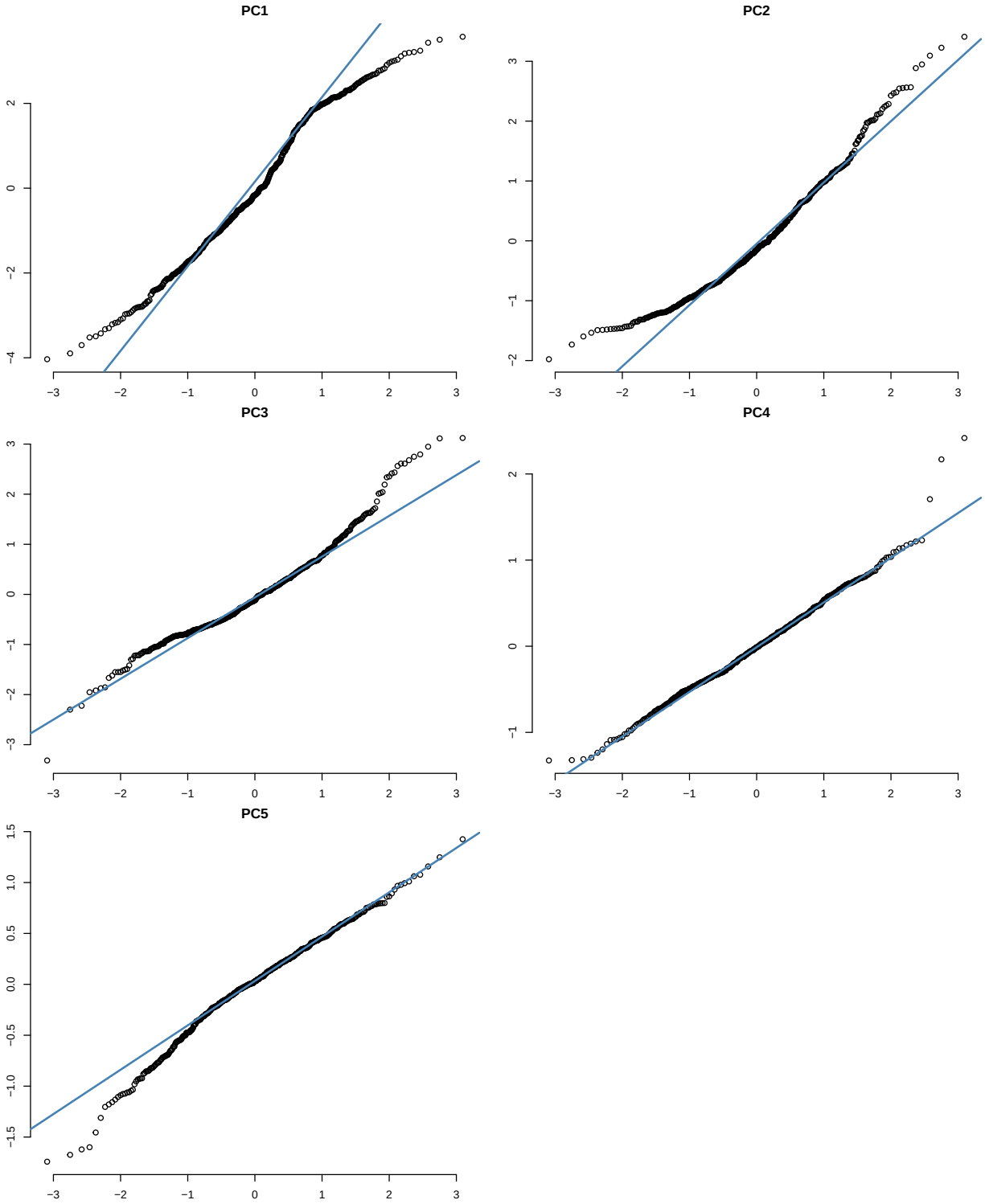
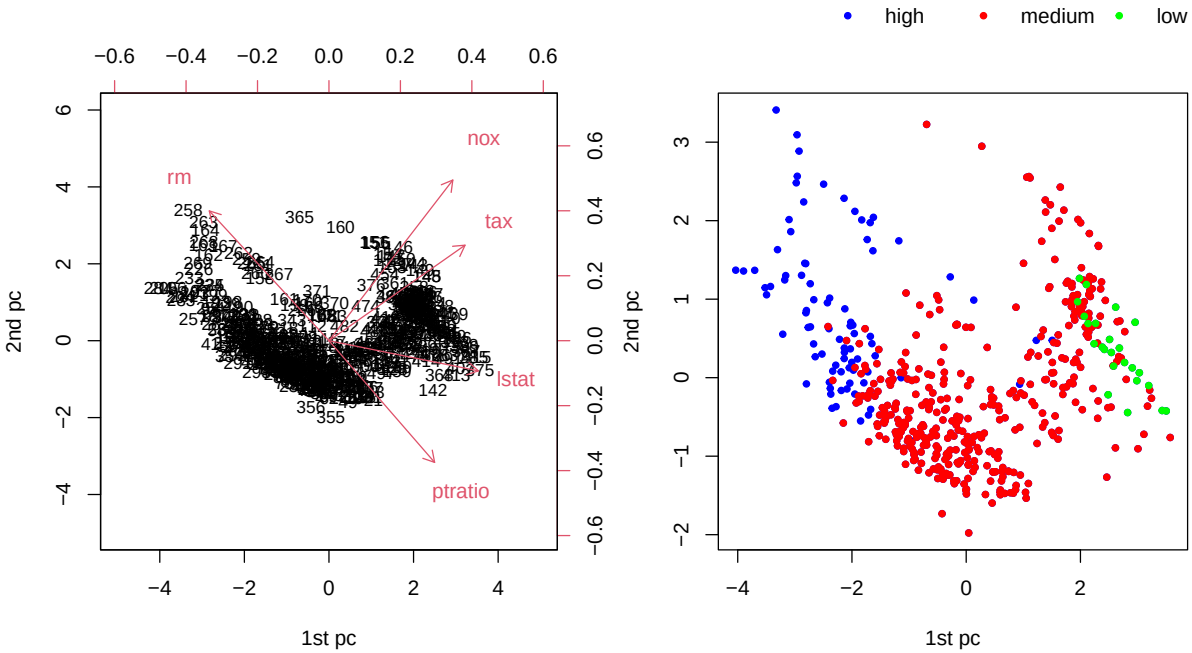


Figure 7 (lhs) shows the biplot of the first two principal components $y_i = (v_1, v_2)^T x_i$ on the primary axis and their weight vectors $v_{1,l}$ and $v_{2,l}$, $l = 1, ..., q$, on the secondary axis (LHS). The principal components y_i are represented by black numbers, while the weight vectors, i.e. the components of the first two orthonormal vectors v_1 and v_2, are represented by red arrows. The length of the arrows measures how strongly each feature influences the principal components, and the cosines of the angles between arrows measure the corresponding correlations.

Figure 7 (rhs) shows the values of the first two principal components $y_i \in \mathbb{R}^2$ for each house $i = 1, ..., 506$, divided in three groups based on house price range ($medv$). In this case, each car x_i is represented by a point $y_i = (v_1, v_2)^T x_i \in \mathbb{R}^2$. The two principal components seem to explain the dependent variable quite well since the clusters of different price ranges are well defined.

Figura 7: Biplot of PCA



4 Singular value decomposition

A second way to find an orthonormal basis for dimensionality reduction is singular value decomposition (SVD). Considering an orthogonal matrix $U \in \mathbb{R}^{n \times q}$, an orthogonal matrix $V \in \mathbb{R}^{q \times q}$ and a diagonal matrix $\Lambda = diag(\lambda_1, ..., \lambda_q)$ with singular values $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_q \geq 0$, SVD is given by

$$X = U \Lambda V^T$$

By substitution, we find that

$$V^T X^T X V = V^T V \Lambda U^T U \lambda V^T V = \Lambda^2 = diag(\lambda_1^2, ..., \lambda_q^2)$$

so that $\lambda_j^2, j = 1, ..., q$, are the eigenvalues of $A = X^T X$ and the column vectors of V are the eigenvectors of A which represent the orthonormal basis $v_1, ..., v_q$. The principal components are given by the column vectors of

$$XV = U\Lambda = Udiag(\lambda_1, ..., \lambda_q) \in \mathbb{R}^{n \times q}$$

This result can be used to construct the matrices X_p, i.e. the best rank p approximation of X that keeps as much variability of X as possible by choosing the optimal $p \leq q$ orthonormal basis vectors $v_1, ..., v_p$. Formally, this matrix is given by

$$X_p = Udiag(\lambda_1, ..., \lambda_p, 0, ..., 0)V^T \in \mathbb{R}^{n \times q}$$

and it minimizes the total squared reconstruction error, measured by the Frobenius norm

$$||X_p - X||_F = \sqrt{\sum_{i=1}^{n} ||\pi(x) - x_i||_2^2}$$

with respect to X among all rank p matrices

$$X_p = \underset{B \in \mathbb{R}^{n \times q}}{argmin} ||X - B||_F, \quad s.t. \quad rank(B) \leq p$$

Hence, it is possible to replace a q dimensional representation of X with a p dimensional representation for which the reconstruction error is minimal.

The singular values λ_p are shown below.

```
SVD = svd(dati_norm)
SVD$d                                   # singular values
```

```
## [1] 37.34972 21.84887 19.80022 11.82855 10.98464
```

```
pc_1 = dati_norm %*% SVD$v[,1]      # 1st principal component
pc_2 = dati_norm %*% SVD$v[,2]      # 2nd principal component
```

5 Autoencoders

An autoencoder consists of two mappings

$$\varphi : \mathbb{R}^q \to \mathbb{R}^p \quad \text{and} \quad \psi : \mathbb{R}^p \to \mathbb{R}^q$$

with $p \leq q$ so that an autoencoder typically leads to a loss of information. The function φ is called encoder, while the function ψ is called decoder. Hence, $y = \varphi(x) \in \mathbb{R}^p$ is a p-dimensional representation of $x \in \mathbb{R}^q$. Let $d(.,.)$ be a dissimilarity function, such that $d(x, x') = 0$ iif $x = x'$. An autoncoder is defined as a pair of mappings (φ, ψ) whose composition $\pi = \varphi \circ \psi$ leads to a small reconstruction error, i.e. $d(\pi(x), x)$ is small.

If we view PCA as an autoencoder, a common choice for the dissimilarity function is the Euclidean distance on $\mathbb{R}^q$

$$d(x', x) = ||x' - x||_2^2 = \sum_{j=1}^{q} (x'_j - x_j)^2$$

which results in the Frobenius norm

$$||X' - X||_F^2 = \sum_{i=1}^{n} ||x'_i - x_i||_2^2$$

The chunk below shows the reconstruction error of the pca for 5 principal components, scaled by the number of samples. For $p = 2$ principal components we obtain a reconstruction error of 1.14.

```
reconstruction <- array(NA, c(length(SVD$d)))
for (p in 1:length(SVD$d)){
  Xp <- SVD$v[,1:p] %*% t(SVD$v[,1:p]) %*% t(dati_norm)
  Xp <- t(Xp)
  error <-  as.matrix((dati_norm-Xp)^2)
  reconstruction[p] <- sqrt(sum(error)/nrow(dati_norm))
}
round(reconstruction,2)
```

```
## [1] 1.49 1.14 0.72 0.49 0.00
```

In this example, we use bottleneck neural networks as an example of a non-linear autoencoder. Ideally, the BNN should have an odd number d of hidden layers and

the central hidden layer should be low dimensional, with $p < q$ hidden neurons. Furthermore, the remaining hidden layers should be symmetrical around the central hidden layer and the number of input units should be equal to the number of output units $q_0 = q$.

Hence, for a BNN with $d = 3$, we choose the following numbers of neurons

$$(q_0, q_1, q_2, q_3, q_4) = (5, 7, 2, 7, 5)$$

The number of input and output units $q_0 = q_4 = 5$ are equal to the number of features, while the number of neurons in the central hidden layer $q_2 = p = 2$ is equal to the number of principal components considered in the previous PCA example.

The activation function for the output layer is linear, since $x \in \mathbb{R}$, while the activation function for the hidden layers is the hyperbolic tangent. The same results obtained with the PCA would be achieved if we chose a linear activation function for the hidden layers of the BNN. In order to calibrate the BNN, we use the mean squared error loss function, which scales the Frobenius norm with the constant $(nq)^{-1}$, so that we minimize the same objective function in the PCA and BNN.

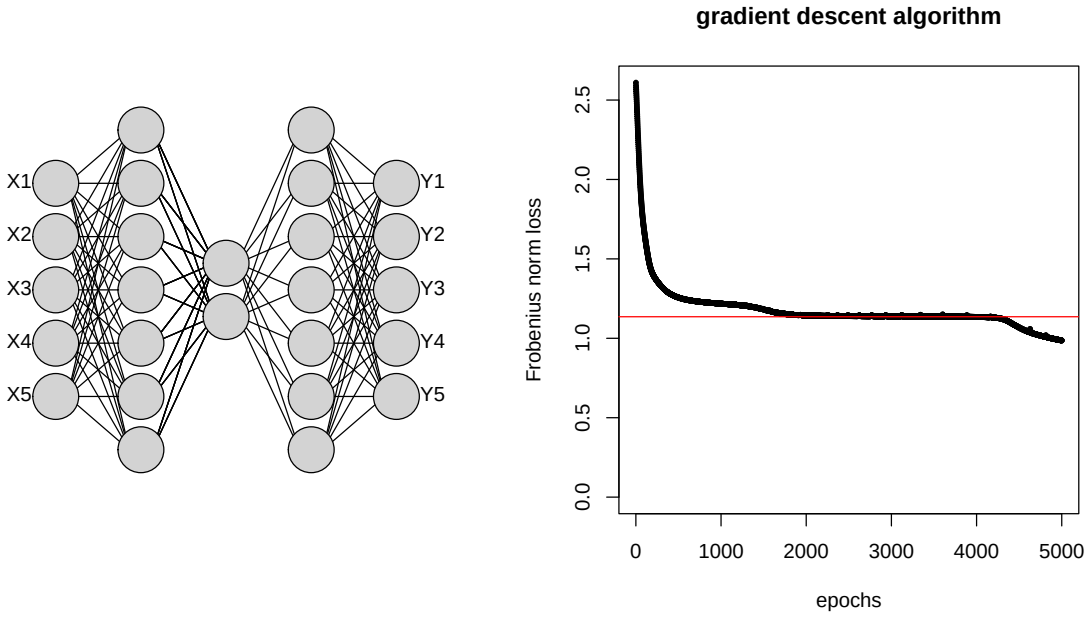
```
set.seed(123)
Input <- layer_input(shape = 5, dtype = 'float32', name = 'Input')

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')
model
```

```
## Model
## Model: "model"
## _________________________________________________________________________
## 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
## _________________________________________________________________________
```

Figure 8 lhs shows the architecture of the neural network. Bias units have not been included since the features are already standardized, which results in a BNN with 98 estimated parameters. Figure 8 rhs shows the decrease of Frobenius loss in the gradient descent algorithm. Training the BNN over 5000 epochs results in a reconstruction error of about 0.95, which is much smaller than the one obtained with PCA and marked by the red horizontal line.

Figura 8: BNN architecture and loss function




```
fit <- model %>% predict(as.matrix(dati_norm))
round(frobenius_loss(dati_norm, fit), 2)
```

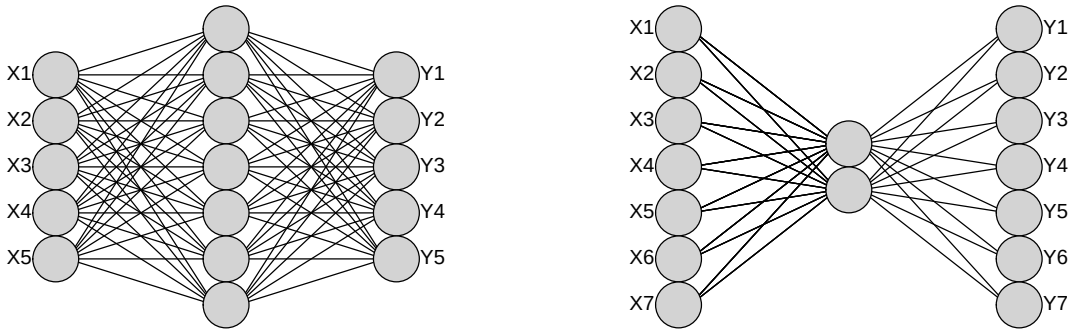
```
## [1] 0.99
```

Another way of training the BNN is by splitting the estimation process into three steps:

- In the first step, we train a BNN with depth 1 and neurons $(q_0, q_1, q_4) = (5, 7, 5)$
- In the second step, we train a BNN with depth 1 and neurons $(q_1, q_2, q_3) = (7, 2, 7)$
- In the third step, we use the pre-trained weights of the outer and inner part of the BNN as initial weights for the calibration of the full BNN with neurons $(q_0, q_1, q_2, q_3, q_4) = (5, 7, 2, 7, 5)$

The BNN architecture for the outer and inner part is shown in Figure 9.

Figura 9: BNN architecture for outer and inner part




```
BNN <- function(q0, q1){
  set.seed(123)
  Input <- layer_input(shape = c(q0), dtype = 'float32', name = 'Input')

  Output = Input %>%
    layer_dense(units=q1, activation='tanh', use_bias=FALSE, name='Bottleneck') %>%
    layer_dense(units=q0, activation='linear', use_bias=FALSE, name='Output')

  model <- keras_model(inputs = Input, outputs = Output)

  model %>% compile(optimizer = optimizer_nadam(), loss = 'mean_squared_error')
  return(model)
}

# outer part
set.seed(123)
outer <- BNN(5, 7)
```

```
fit <- outer %>% fit(as.matrix(dati_norm), as.matrix(dati_norm),
                     epochs= 5000, batch_size= nrow(dati_norm), verbose=0)

zz <- keras_model(inputs= outer$input,
                  outputs=get_layer(outer, 'Bottleneck')$output)

yy <- zz %>% predict(as.matrix(dati_norm))

# inner part
set.seed(123)
inner <- BNN_1(7, 2)

fit <- inner %>% fit(as.matrix(yy), as.matrix(yy),
                     epochs = 5000, batch_size = nrow(yy), verbose=0)

# get pre trained weights
weight_outer <- get_weights(outer)
weight_inner <- get_weights(inner)

# use pre-trained weights to fit the full BNN
weights <- get_weights(model)
weights[[1]] <- weight_outer[[1]]
weights[[4]] <- weight_outer[[2]]
weights[[2]] <- weight_inner[[1]]
weights[[3]] <- weight_inner[[2]]
set_weights(model, weights)

fit0 <- model %>% predict(as.matrix(dati_norm))

# reconstruction error
round(frobenius_loss(dati_norm, fit0), 2)
```

```
## [1] 1.23
```

Using the pre-trained weights extracted above, we retrain the full BNN and obtain a reconstruction error of 1.3, which is worse than the one obtained with PCA (1.19) as well as the one obtained by training the full model. However, by using these weights to initialize the full BNN, we obtain a reconstruction error of 0.84, which is slightly smaller than the one obtained without initialization. Figure 10 shows the decrease of the Frobenius loss over 5000 training epochs. We see that after about 1000 epochs the reconstruction error falls below the one of the PCA with 2 principal components.

Figura 10: Loss function of pre-trained BNN

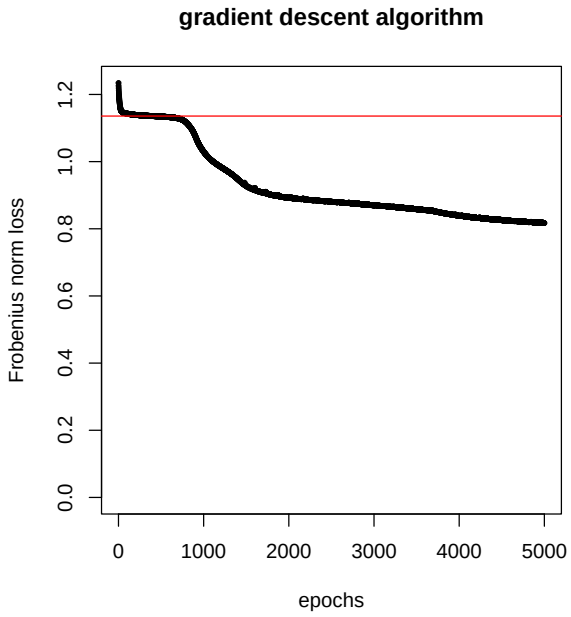
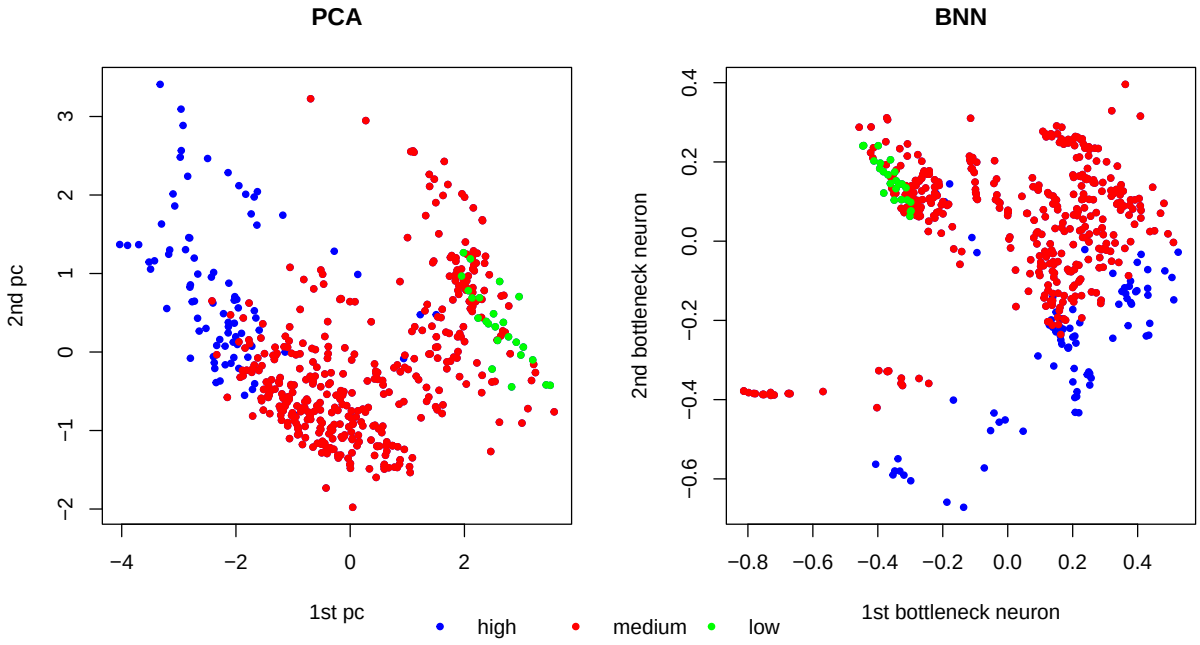


Figure 11 shows a comparison of the clusters obtained with PCA and BNN. We see that the dimension reduction of the BNN appears to be a slightly rotated and scaled version of the PCA results.

Figura 11: Cluster analysis



6 References

Harrison Jr, D., & Rubinfeld, D. L. (1978). Hedonic housing prices and the demand for clean air. Journal of environmental economics and management, 5(1), 81-102.

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

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

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

Rentzmann, S., & Wuthrich, M. V. (2019). Unsupervised Learning: What is a Sports Car?. Available at SSRN 3439358.