university of groningen

faculty of science and engineering

University of Groningen, Groningen

Principal Component Analysis & Autoencoders

Student:
Frixos Meachim (S4502094)

Course:
Contemporary Statistics and Applications

June 30, 2021

Contents

1. Introduction 1
2. Data 1
3. Methods 2
   - 3.1 Pre-processing 2
   - 3.2 Principal Component Analysis 2
   - 3.3 Autoencoders 3
4. Results 6

1. Introduction

The volume of data produced has seen a huge increase over the last decade. Furthermore, the dimensions of the data produced is also growing, while our capacity to think in dimensions higher than 3 is by nature, limited. In this context, dimensionality reduction techniques are critical in order to produce graphical representations that capture the underlying structure of data while maintaining interpretability. Dimensionality reduction techniques aim to solve this problem by reducing the dimensions of a dataset, while preserving as much information as possible.

Besides their use for data visualisation, dimensionality reduction techniques are also crucial in the context of predictive modeling. Reducing the dimensionality of a dataset leads to simpler models, which are in turn easier to interpret, and typically require less computational effort to be trained.

In this paper we compare two such techniques, namely Principal Component Analysis (PCA) and Autoencoders (AEs). We examine how effective they are at reducing the dimensions of the Australian Institute of Sport (AIS) dataset, and of the Boston Housing (BH) dataset.

Both PCA and AEs, derive their utility by learning a representation of the data, which is in a domain different to our original. Autoencoder networks are comprised of two sub-networks: an encoder and a decoder. The encoder projects our data to a lower dimensional "latent space", while the decoder reconstructs our data using this latent space representation. PCA on the other hand, finds a set of orthogonal axes which follow the direction of highest variance in the data. In this context, there are two differences between the methods that should be noted. Firstly, AEs perform a non-linear transformation, while PCA performs a linear one. Secondly, the axes found by PCA, are provided in an ordered manner. That is we can order them in terms of their representational power. AEs on the other hand, provide no such ordering. Furthermore, we have no guarantee that these axes are independent to each other. Overall AEs tend to produce more flexible and powerful latent spaces, at the cost of interpretability and increased computing time[8].

2. Data

Australian Institute of Sport

This dataset consists of measurements on high-performance athletes from the Australian Institute of Sport (AIS). It contains information on 13 variables such as Body Mass Index, height and weight, concerning 202 individuals, with 102 males and 100 females. More detailed information about the collection of this data can be found in [9].

Boston Housing

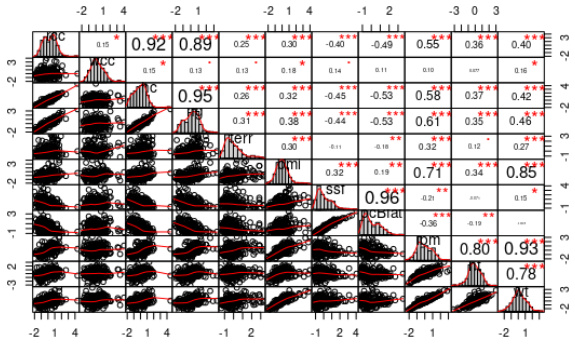
This dataset contains information collected by the U.S Census Service regarding housing in the area of Boston. It was first published in [2]. It consists of measurements on 14 variables concerning 506 cases. These variables include information such as nitric oxides concentration, average number of rooms per dwelling and full-value property-tax.

Comparison of Datasets

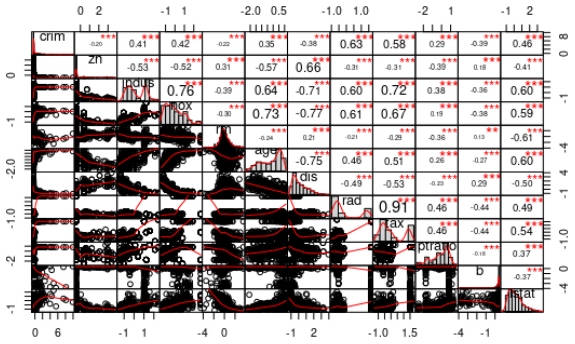
Figure 1 bellow shows the absolute values of correlation between the variables in each dataset. This is shown by the values located to the right of the diagonal. On the diagonal, we can see the histograms of individual variables. Finally to the left of the diagonal, we see the scatter plots between all variables.

After closely inspecting Figure 1 (a), which refers to the AIS dataset, we can see that several variables are highly correlated. Furthermore, after inspecting the scatter plots, we notice that some of these exhibit near perfect linear relationships.

Similarly we inspect Figure 1 (b). In contrast to the AIS dataset, fewer variables exhibit correlations higher than 0.9, while many of them show sings of non linear relationships. Having made this observation, we could expect to see AEs performing better than PCA in terms of dimensionality reduction, due to their ability to deal with the non linearity present in the data.



(a) Australian Institute of Sport



(b) Boston Housing

Figure 1: Correlation matrices, histograms and scatterplots for the AIS and BH datasets.

3. Methods

3.1 Pre-processing

A Z-score transformation was applied to the numerical variables of both datasets. The variables X_j were transformed as follows:

$$X_j^* = \frac{X_j - \hat{X}_j}{\sigma_j} \tag{1}$$

Where $\hat{X}_j$ and σ_j are the mean and standard deviation of feature j, and X_j^* is the transformed feature. Effectively, the input variables X were scaled to have mean zero and standard deviation one.

Categorical variables were removed from both datasets, and the numerical variable "medv" was also excluded from the BH dataset, which was instead used as a response variable in our visualisations. This left us with 11 and 12 variables for the AIS and BH dataset respectively.

3.2 Principal Component Analysis

Principal Component Analysis was first found by Karl Pearson in 1901, but has been independently discovered under many other names. It is a method which is often used to reduce the dimensionality of data consisting of interrelated variables. This is achieved by transforming the data to a new set of variables, called principal components (PCs), which are no longer correlated [4]. These PCs are constructed as linear combinations of the initial variables.

To construct these PCs, one must first find the eigenvalues $\lambda_1, ...\lambda_N$ and corresponding eigenvectors $a_1, ...a_N$ of the covariance matrix S. In our case, since a Z-Score transformation was applied to our data, the covariance matrix is equal to the correlation matrix, since the variables are 0 centered and have unit variances. By using these eigenvalues and eigenvectors, we can construct the principal components. Effectively, we are trying to reconstruct the matrix S, using lower rank matrices. Let $X = (X_1, ...X_N)^T$ represent the observed variables. Then given the N eigenvectors of the covariance (correlation) matrix, N new variables called principal components, can be constructed as follows:

$$z_j = a_j^T X, \;\; j = 1, .., N \tag{2}$$

For the variables z_j, constructed using formula 2, we know that they are independent linear combinations of the observed variables X. Furthermore, we know that their variance $var(z_j)$ is equal to the corresponding eigenvalue λ_j. That is:

$$var(z_j) = \lambda_j, \;\; j = 1, .., N \tag{3}$$

As mentioned above, we can find a reconstruction of S using rank one matrices, which are constructed using the eigenvalues and eigenvectors as follows:

$$S = \sum_{j=1}^{N} \lambda_j a_j a_j^T \tag{4}$$

Furthermore, PCA has a nice property that allows us to also assess the quality of our reconstruction. Namely, we can calculate how much of the variance in our original variables X, is explained by the principal components. Specifically, we know that the sum of variances of our original variables, is equal to the sum of eigenvalues of the S matrix. That is:

$$trace(S) = \sum_{j=1}^{N} trace(\lambda_j a_j a_j^T) = \sum_{j=1}^{N} \lambda_j \tag{5}$$

Finally, we can calculate how much of the variance each individual principal component z_j explains, by looking at the ratio of total variance explained:

$$TVE(z_j) = \frac{var(z_j)}{\sum_{j=1}^{N} var(z_j)} = \frac{\lambda_j}{\sum_{j=1}^{N} \lambda_j} \tag{6}$$

As a result of its simplicity and relatively modest computational cost, PCA is widely used, and it is often the first method employed to attempt a dimensionality reduction, before more complex techniques are considered [3].

Despite its computational efficiency, it comes with shortcomings. Since the new variables are constructed through linear combinations of our original data, PCA is not an effective dimensionality reduction technique when dealing with variables which exhibit a high *non-linear* correlation, as it cannot capture the underlying structure in the data.

Another issue with PCA is related to the method used in selecting the principal components. It is common to keep only the components which (individually) explain a substantial amount of the total variance in the data, using equation 6, or alternatively to keep the components which cumulatively explain an arbitrarily chosen percentage of the variance (e.g 95%). However in terms of predictive power, components with small variance have been demonstrated to be useful[6]. In this context, the use of scree plots is very beneficial, as it can help us determine how many principal components to keep. To obtain a scree plot, we simply plot the eigenvalues λ_j in descending order. By looking at the rate of decay in the eigenvalues, we can make better informed decisions about the number of components to be used.

Finally, PCA is an unsupervised approach in the sense that it reduces the dimensions of data using only information about the explanatory variables, while no information about the response variable is used. In many applications this may prove to be problematic, when building predictive models, since the direction of highest variance in our explanatory variables may not necessarily be the direction of highest variance in the response variable.

3.3 Autoencoders

Autoencoders are a type of Neural Network which are trained to replicate their input as their output [1]. This may not sound very useful in itself, however autoencoders can be a very effective dimensionality reduction technique. To understand how they work, we must first briefly introduce the concepts of neurons, and neural networks.

Neurons

A neuron i can be thought of as a processing element, which simply computes the weighted average S_i of its inputs $X_0, X_1 ... X_j$. S_i is then passed through an activation function A, which determines whether or not the neuron will activate. The activation function can be linear, but is typically chosen to be non linear. Figure 2 below, shows the architecture of a neuron i which receives n inputs $X_0 ... X_n$. Using the notation described above, the output Y_i of neuron i is:

$$Y_i = A(\sum_{j=1}^{N} w_{ij} X_j) = A(S_i) \tag{7}$$

For this report, our activation function was chosen to be the hyperbolic tangent function, and so the output Y_i of neuron i can be written as follows.

$$Y_i = tanh(S_i) = \frac{e^{S_i} - e^{S_i}}{e^{S_i} + e^{S_i}} \tag{8}$$

The hyperbolic tangent function receives any real value as input, and outputs values in the range -1 to 1. The larger the input is, the closer the output will be to 1, while large negative inputs will produce outputs close to -1. The tanh function effectively shrinks the output of the neuron to values ranging between -1 and 1. It should be noted that there are many possible choices for A, such as the sigmoid and softmax functions. These functions scale the output to different ranges, such as 0-1, however the principle is the same.

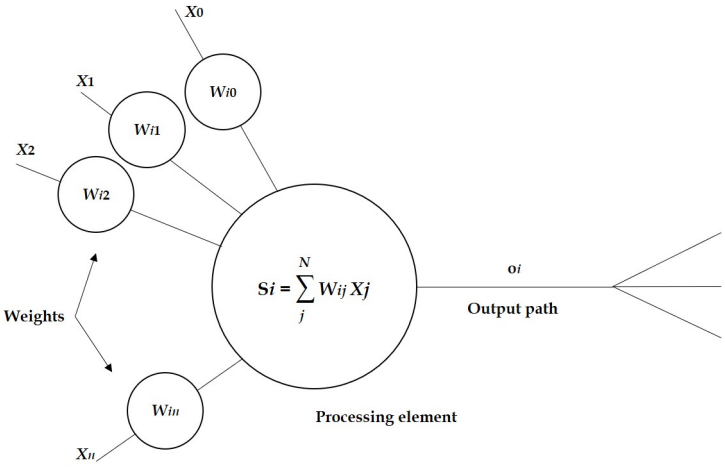


Figure 2: Architecture of a single neuron

Neural Networks

Neural networks (NN) are a collection of interconnected neurons. Figure 3 shows a neural network which consists of three layers L_1, L_2, L_3. Going forward we will use the following notation to refer to the structure of NNs. Let $L_1 - L_2 - ... - L_n$ be a sequence representing the architecture of a NN with n layers. Where L_k is the number of neurons at the kth layer, so that 3-2-3 represents a NN with input layer size 3, hidden layer size 2, and output layer size 3.

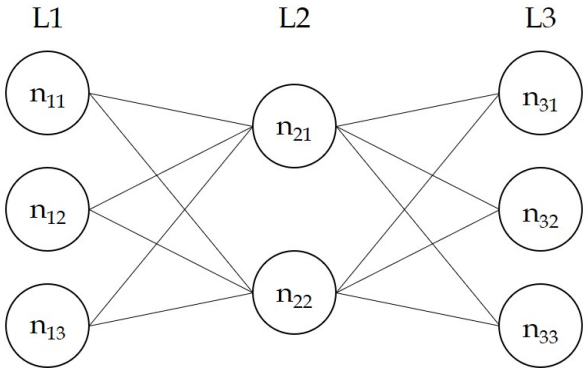


Figure 3: Architecture of a neural network of type 3-2-3.

In this example, each neuron n_L belonging to layer L is connected to all other neurons in the adjacent layers. Furthermore we see that the middle or "hidden" layer L_2 has fewer neurons than both the input and output layers L_1 and L_2. This is called a "bottleneck" and is a defining features of a type of NN called an undercomplete autoencoder.

The general structure of an AE consists of two parts. An encoder function $E = f(X)$ which maps our input X to an encoded representation $f(X)$, and a decoder function $D = g(f(X))$ which maps the encoded representation to a reconstruction r.

Attempting to copy our input as our output isn't very useful in itself. However in the process of training these networks, we hope that the hidden layer will capture some important features of our data. One way of doing so is using undercomplete autoencoders, which means that the dimension of our hidden layer is lower than that of our input dimension. This forces the AE to find a meaningful representation of the data, which is in the dimension of the undercomplete layer.

All AEs were trained using the Adam optimisation algorithm, which is a stochastic gradient descent method, based on adaptive estimation of first-order and second-order moments. For more information on Adam, see [7]. The code used to train the AEs, as well as find the PCs, can be found in the Appendix.

Method Assessment

To assess the effectiveness of PCA and AEs as dimensionality reduction techniques, we will compare the reconstruction error of each method, as measured by the Mean Squared Error (MSE). Let $\hat{Y}_i$ be the reconstructed observation i obtained from our dimensionality reduction technique, and Y_i the original observation. Then given N observations in our dataset, we calculate the sum:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{9}$$

Choice of Latent Space Dimension

Choosing the dimension of our latent space is not a trivial task. Clearly, we would like to reduce the dimensions of our data by as much as possible. However this should not come at the cost of information contained in the new data. As mentioned above, PCA offers a relatively simple solution, which is based on finding the variance that each PC explains, as a proportion of the total variance in our data. AEs on the other hand, have no such clear-cut method of finding the number of variables required to effectively represent our data. With this in mind, we chose the dimension of our latent space using only information about PCA and the variance explained by the PCs.

The left side of Figure 4 shows the cumulative variance explained, plotted against the number of principal components used for the AIS data. Since 6 components explain more than 90% of the variance, and subsequent components yield very little increase in variance explained, we will only keep these 6 components.

Similarly, the right side of Figure 4 shows the cumulative variance explained, plotted against the number of principal components used for the BH data. A total of 10 components are sufficient to explain more than 90% of the variance in the data. Hence only 10 components were kept.

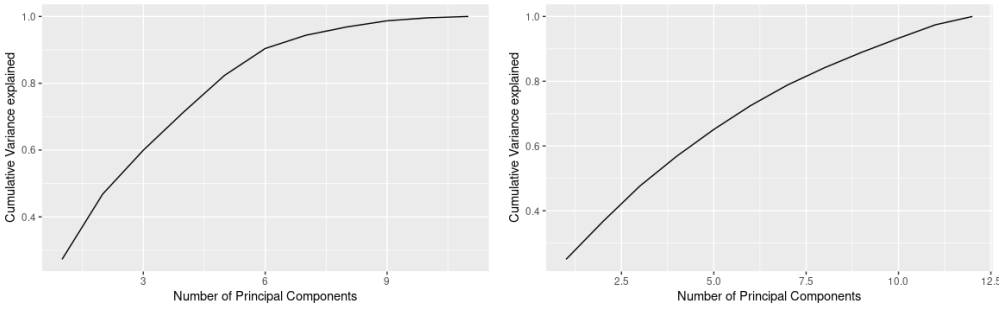
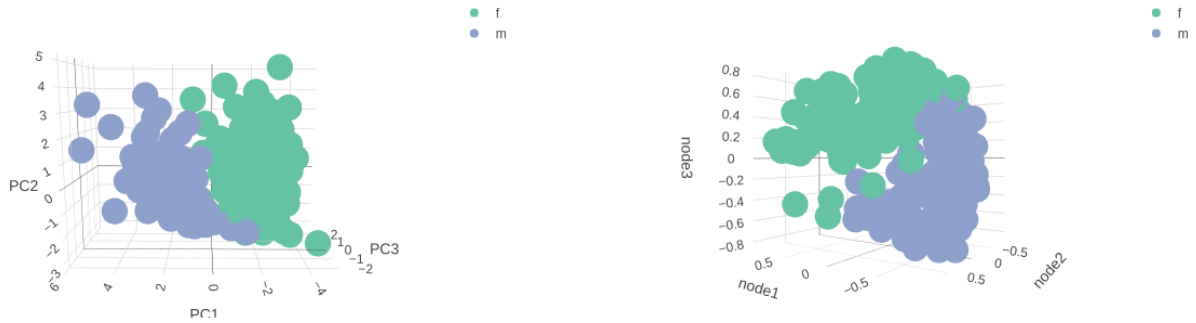


Figure 4: Percentage of variance explained by PCs for AIS and BH

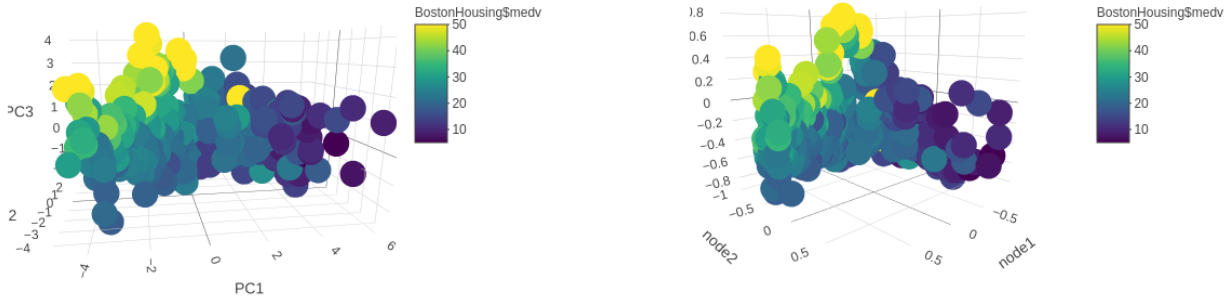
4. Results

To get an intuition of how well PCA and AEs can visualize our data, we begin by plotting it in three dimensions. Figure 5 shows how three PCs can differentiate between male and female athletes, alongside the performance of our AE on the same task. Figure 6 shows the same visualisation for the BH dataset, using the "medv" (medial value of owner occupied homes) as the variable of interest.



(a) Australian Institute of Sport (b) Boston Housing

Figure 5: A 3-dimensional representation of the Australian Institute of Sport data, achieved through PCA on the left, and a 11-6-3-6-11 AE on the right.



(a) Australian Institute of Sport (b) Boston Housing

Figure 6: A 3-dimensional representation of the Boston Housing data, achieved through PCA on the left, and a 12-6-3-6-12 AE on the right. The color scale represents the median value of owner occupied homes, measured in $1000's.

The AE used for both datasets used a hyperbolic tangent activation function, with structure 11-6-3-6-11 and 12-6-3-6-12 for the AIS and BH datasets respectively. Applied to the AIS data, both methods managed to effectively separate the two classes, with no significant differences between the two representations. The same can be said about the representations produced for the BH dataset, with both methods somewhat successfully capturing the structure of the data. Finally neither method produced a representation which is visibly superior to the other.

In Figure 7 we plotted the reconstruction error that AEs with structure 11-9-k-9-11 obtained for k=1-8, as well as the reconstruction error produced by PCA using k PCs on the AIS dataset. Similarly we show the same results for the BH dataset. The AEs used for the reconstruction of the BH data, had a structure 12-10-k-10-12 for k=1-9.

The AEs produced very similar results to PCA when applied to the AIS data, however AEs produced slightly better results, for all values of k. Specifically we see that as the number of variables increases, the reconstruction errors converge to approximately the same values for PCA and AEs. Furthermore, we notice that the point at which the reconstruction errors converge, is close to 6 components. Using Figure 4, we saw that 6 components explain a substantial amount of the variance in the data (90%), while subsequent components yield very little information. This could mean that the performance of the methods converges, simply because we have provided them with most of the useful information contained in the dataset.

For the BH data however, we see that the performance of AEs is consistently better than that of PCA for all values of k. This could be attributed to the non-linearity present in the BH data. Since the AEs tested here, used

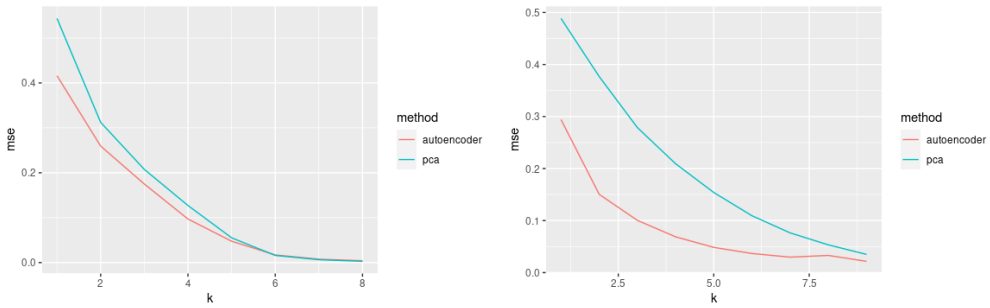


Figure 7: Mean squared error achieved by an AE with bottleneck of size k, and PCA using k components. Results for the AIS dataset are on the left, while the BH results are on the right.

a hyperbolic tangent function, they were more successful at obtaining good reconstructions of the data, from a dimensionaly reduced representation.

To confirm this, we also trained the same AEs described above, this time using a linear activation function instead. The result obtained using a linear activation function were practically identical for AEs and PCA, which supports the idea that the improved performance of AEs is a result of their ability to capture non-linear relationships in the data.

In conclusion, AEs outperformed PCA in terms of reconstruction error. However it is important to note, that training AEs was very computationally intensive, especially considering the small size of our datasets. It is easy to see that when dealing with larger datasets, AEs can quickly become impractical. PCA on the other hand scales better with large amounts of data, and considering the lack of interpretability of AEs, PCA is rightfully the first choice of method considered, when attempting to reduce the dimensions of a dataset.

Appendix

Australian Institute of Sport

```
#Install required packages
library(PerformanceAnalytics)
library(tidyverse)
library(DAAG)
library(plotly)
library(caret)
library(keras)
library(tensorflow)
#Set seed for reproducibility
set.seed(123)

#Keep only numerical variables
df_x=ais[,1:11]

#Scale data to have mean 0 and standard deaviation 1
preproc1 <- preProcess(df_x, method=c("center", "scale"))
df_x <- predict(preproc1, df_x)

#Compute Principal Components
pca <- prcomp(df_x)

#Plot cumulative variance explained
qplot(x=1:11,y =cumsum(pca$sdev)/sum(pca$sdev),geom="line",xlab="PCs",ylab="CVE")

#Plot individual variance explained
qplot(x = 1:11, y = pca$sdev/sum(pca$sdev), geom = "line",xlab = "PCs",ylab="IVE")

#Create correlation chart
chart.Correlation(df_x, histogram=TRUE, pch=25)

#Plot first three PCs
plot_ly(as.data.frame(pca$x),x =~PC1,y =~PC2,z=~PC3,color=~ais$sex) %>% add_markers()

#Autoencoders

#Transform data to suitable format for keras package
x_train <- as.matrix(df_x)

#Set model with 5 layers. (first layer is size ncol(x_train))
model <- keras_model_sequential()
model %>%
  layer_dense(units = 6, activation = "tanh", input_shape = ncol(x_train)) %>%
  layer_dense(units = 3, activation = "tanh", name = "bottleneck") %>%
  layer_dense(units = 6, activation = "tanh") %>%
  layer_dense(units = ncol(x_train))

#Inspect model layers
summary(model)


#Compile model
model %>% compile(
  loss = "mean_squared_error",
  optimizer = "adam")


#Fit model
```

```
model %>% fit(x = x_train, y = x_train,epochs = 1000,verbose = 0)

#Evaluate the performance of the model
mse.ae3 <- evaluate(model, x_train, x_train)
mse.ae3

#Extract intermediate layer
intermediate<-keras_model(inputs=model$input,outputs=get_layer(model,"bottleneck")$output)
intermediate_output <- predict(intermediate, x_train)

#Plot the reduced data set
aedf3 <- data.frame(node1 = intermediate_output[,1], node2 = intermediate_output[,2]
, node3 = intermediate_output[,3])
plot_ly(aedf3, x = ~node1, y = ~node2, z = ~node3, color = ~ais$sex) %>% add_markers()

#Calculate reconstruction error for different values of k (variables)
#tried activation = "tanh" and "linear"

#PCA reconstruction
pca.recon <- function(pca, x, k){
  mu <- matrix(rep(pca$center, nrow(pca$x)), nrow = nrow(pca$x), byrow = T)
  recon <- pca$x[,1:k] %*% t(pca$rotation[,1:k]) + mu
  mse <- mean((recon - x)^2)
  return(list(x = recon, mse = mse))
}
xhat <- rep(NA, 11)
for(k in 1:11){
  xhat[k] <- pca.recon(pca, x_train, k)$mse
}

#AE reconstruction
ae.mse <- rep(NA, 8)
for(k in 1:8)
{
  modelk <- keras_model_sequential()
  modelk %>%
    layer_dense(units = 9, activation = "linear", input_shape = ncol(x_train)) %>%
    layer_dense(units = k, activation = "linear",name = "bottleneck") %>%
    layer_dense(units = 9, activation = "linear") %>%
    layer_dense(units = ncol(x_train))
  modelk %>% compile(
    loss = "mean_squared_error",
    optimizer = "adam"
  )
  modelk %>% fit(
    x = x_train,
    y = x_train,
    epochs = 2000,
    verbose = 0
  )
  ae.mse[k] <- unname(evaluate(modelk, x_train, x_train))
}

df <- data.frame(k = c(1:8, 1:8), mse = c(xhat[1:8], ae.mse),
method = c(rep("pca", 8), rep("autoencoder", 8)))

#Plot reconstruction errors against k
ggplot(df, aes(x = k, y = mse, col = method)) + geom_line()
```

Boston Housing

```
#Install required packages
library(PerformanceAnalytics)
library(mlbench)
library(tidyverse)
library(DAAG)
library(plotly)
library(caret)
library(keras)
library(tensorflow)
#Set seed for reproducibility
set.seed(123)

#Keep only numerical variables
data(BostonHousing)
df_x=BostonHousing[,1:13]
df_x$chas=NULL

#Scale data to have mean 0 and standard deviation 1
preproc1 <- preProcess(df_x, method=c("center", "scale"))
df_x <- predict(preproc1, df_x)

#Compute Principal Components
pca <- prcomp(df_x)

#Plot cumulative variance explained
qplot(x=1:12,y=cumsum(pca$sdev)/sum(pca$sdev),geom ="line",xlab = "PCs",ylab="CVE")

#Plot individual variance explained
qplot(x = 1:12, y = pca$sdev/sum(pca$sdev),geom="line",xlab = "PCs",ylab="IVE")

#Create correlation chart
chart.Correlation(df_x, histogram=TRUE, pch=25)

#Plot first three PCs
plot_ly(as.data.frame(pca$x),x=~PC1,y=~PC2,z=~PC3,
color=~BostonHousing$medv)%>%add_markers()

#Autoencoders

#Transform data to suitable format for keras package
x_train <- as.matrix(df_x)

#Set model with 5 layrs. First layer isn't defined explicitly,
#but through the number of variables we pass to the first layer

model <- keras_model_sequential()
model %>%
  layer_dense(units = 6, activation = "tanh", input_shape = ncol(x_train)) %>%
  layer_dense(units = 3, activation = "tanh", name = "bottleneck") %>%
  layer_dense(units = 6, activation = "tanh") %>%
  layer_dense(units = ncol(x_train))

#Inspect model layers
#summary(model)

#Compile model
model %>% compile(
  loss = "mean_squared_error",
  optimizer = "sgd")
```

```
#Fit model
model %>% fit(
  x = x_train,
  y = x_train,
  epochs = 2000,
  verbose = 0)

#Evaluate the performance of the model
mse.ae3 <- evaluate(model, x_train, x_train)
mse.ae3

#Extract the intermediate layer
intermediate<-keras_model(inputs=model$input,
outputs=get_layer(model,"bottleneck")$output)
intermediate_output <- predict(intermediate, x_train)

#Plot the reduced dataset
aedf3<-data.frame(node1=intermediate_output[,1],
node2=intermediate_output[,2],node3 =intermediate_output[,3])
plot_ly(aedf3, x = ~node1, y = ~node2, z = ~node3,
color = ~BostonHousing$medv) %>% add_markers()

#Calculate reconstruction error for different values of k
#tried activation="tanh" and "linear"

#PCA reconstruction
pca.recon <- function(pca, x, k){
  mu <- matrix(rep(pca$center, nrow(pca$x)), nrow = nrow(pca$x), byrow = T)
  recon <- pca$x[,1:k] %*% t(pca$rotation[,1:k]) + mu
  mse <- mean((recon - x)^2)
  return(list(x = recon, mse = mse))
}
xhat <- rep(NA, 13)
for(k in 1:13){
  xhat[k] <- pca.recon(pca, x_train, k)$mse
}
#AE reconstruction
ae.mse <- rep(NA, 9)
for(k in 1:9)
{
  modelk <- keras_model_sequential()
  modelk %>%
    layer_dense(units = 10, activation = "linear", input_shape = ncol(x_train)) %>%
    layer_dense(units = k, activation = "linear",name = "bottleneck") %>%
    layer_dense(units = 10, activation = "linear") %>%
    layer_dense(units = ncol(x_train))
  modelk %>% compile(
    loss = "mean_squared_error",
    optimizer = "adam"
  )
  modelk %>% fit(
    x = x_train,
    y = x_train,
    epochs = 2000,
    verbose = 0
  )
  ae.mse[k] <- unname(evaluate(modelk, x_train, x_train))
}
df <- data.frame(k = c(1:9, 1:9), mse = c(xhat[1:9], ae.mse),
method = c(rep("pca", 9), rep("autoencoder", 9)))
```

```
#Plot reconstruction errors against k
ggplot(df, aes(x = k, y = mse, col = method)) + geom_line()
```

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[2] David Harrison and Daniel L Rubinfeld. "Hedonic housing prices and the demand for clean air". In: *Journal of Environmental Economics and Management* 5.1 (1978), pp. 81–102. ISSN: 0095-0696. DOI: https://doi.org/10.1016/0095-0696(78)90006-2. URL: https://www.sciencedirect.com/science/article/pii/0095069678900062.

[3] Herbert Jaeger. *Machine Learning. Lecture Notes*. 2019.

[4] I. T Jolliffe. *Principal Component Analysis*. Springer, 2002.

[5] I. T. Jolliffe and Jorge Cadima. "Principal component analysis: a review and recent developments". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (2016).

[6] Ian T. Jolliffe. "A Note on the Use of Principal Components in Regression". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 31.3 (1982), pp. 300–303. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2348005.

[7] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[8] Saïd Ladjal and Alasdair Newson. *A PCA-like Autoencoder*. Apr. 2019.

[9] R. Telford and R. Cunningham. "Sex, sport, and body-size dependency of hematology in highly trained athletes." In: *Medicine and science in sports and exercise* 23 7 (1991), pp. 788–94.