Principal Component Analysis

Module 1 | Chapter 4 | Notebook 3

Sometimes data has more features than you really want. In this lesson, we'll look at how we can reduce their number while retaining the most important information. By the end you will be able to:

- Estimate the process of the principal component analysis
- Reduce the dimensionality of the data as specified
- Interpret the results of the principal component analysis

How `PCA` works

In the last exercise we used `PolynomialFeatures` from `sklearn.preprocessing` to enrich our data with new features. You learned that it's possible to create a lot of features very quickly. In other cases, we already have very high-dimensional data, i.e. data with an extremely large number of features. Training algorithms with this kind of data often takes a long time and performance is almost always a very important factor in data science projects. It is also not uncommon that a model with too many features can give even worse results than one with fewer features. In machine learning we refer to what's known as the *curse of dimensionality*, and as we know, dimensions are the features of a data set and so the 'curse' directly influences the data. $ % It states that as the number of dimensions increases, the space covered by a single data point becomes exponentially smaller and the space not covered by the existing data points becomes larger accordingly.$ It states that with more dimensions, distances between points become larger on average (this is because the volume of a higher dimensional object lies almost completely on its surface - crazy but mathematically well understood). This in turn presents problems for many machine learning algorithms, because if individual data points are far apart, it is more difficult for the algorithm to identify and learn correlations. This can be avoided by inserting more and more points or by reducing the dimensionality.

So you should remember this rule of thumb:

With each additional dimension, the number of observations required increases.

For this reason it is often desirable to reduce the number of features without losing too much information.

This is where dimensionality reduction algorithms come in. Dimensionality reduction belongs to the field of unsupervised learning. So it works without a target vector. Instead, information

about the structure of the data is used directly to determine whether and how the data can be transformed so that it has fewer features. Different algorithms have different approaches. However, the basic assumption is always that there are features that contain redundant or little information about the data. Now let's have a look at an example to gain a better understanding of this.

The best known algorithm for dimensionality reduction is the *principal component analysis* or `PCA` for short. This is what we'll focus on in this lesson.

The best way to understand how `PCA` works is to use an example. Import the data from the *swim_records.txt* file in your current working directory. It contains the men's 400 meters freestyle swimming world records from 1908 to 1960. It's a CSV file with tab stops (`'\t'`) as separators. Store the data as a `DataFrame` named `df`. The entries in the `'date'` column are *datetimes*. Convert these accordingly when you import the data. Then print the first five rows.

In [7]:
```
import pandas as pd
df = pd.read_csv('swim_records.txt', sep='\t', parse_dates=['date'])
df.head()
```

Out[7]:

	seconds	date	country	athlete
0	336.48	1908-07-16	United Kingdom	Henry Taylor
1	335.48	1911-09-21	United Kingdom	Sydney Battersby
2	329.00	1912-04-21	Hungary	Alajos Kenyery
3	328.24	1912-06-05	Hungary	Bela Las-Torres
4	324.24	1912-07-14	Canada	George Hodgson

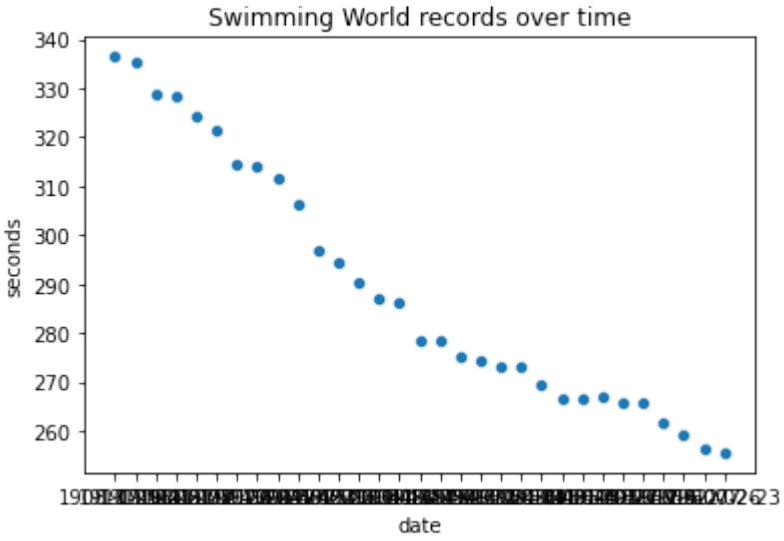
The following data dictionary describes the data.

Column number	Column name	Type	Description
0	`'seconds'`	continuous (`float`)	time of the world record in seconds
1	`'date'`	continuous (`datetime`)	date the world record was set
2	`'country'`	categorical (`string`)	country where the world record was set
3	`'athlete'`	categorical (`string`)	athlete who set the world record

The `'date'` and `'seconds'` columns are particularly relevant for our example. Visualize them with a scatter diagram. Use the date for the x-axis and plot the world record times on the y-axis.

In [2]:
```
import seaborn as sns

ax = sns.scatterplot(data=df,
                     x='date',
                     y='seconds')
ax.set(title='Swimming World records over time');
```



For the scatterplot of world records we'll keep things simple by only using two dimensions to understand the principle of `PCA`. We will now use `PCA` to turn two dimensions into just one.

In doing so, `PCA` proceeds as follows: For each dimension that the data should have after the transformation, the algorithm searches for the direction of the greatest variance in the data. This direction is called the *principal component*. The next principal component is chosen so that it is perpendicular to the previous one, pointing in the direction of the greatest remaining variance. In the end the data is projected onto the principal components. Each principle component is then a feature of the transformed data. The number of new features is therefore directly indicated by the number of principal components.

This procedure offers the following advantages:

- Because the principle components point in the direction of the greatest variance, as little information as possible is lost about the structure of the data.
- Because the main components are perpendicular to one another, they are independent of one another. This means that they do not contain redundant information about the data. (Examples for perpendicular vectors in 2 dimensions are [1,0] and [0,1] or also [1,1] and [1,-1])

Let's have a look at this using our data as an example.

First we have to standardize the data again. Otherwise the data is on different scales and the greatest variance is indicated by the column with the largest numbers.

The `'date'` column contains *datetimes*. `StandardScaler` cannot handle these. So we need to convert `'date'` into a numeric format first. Apply the `pd.to_numeric()` function to the columns and store the result as `arr_sil`.

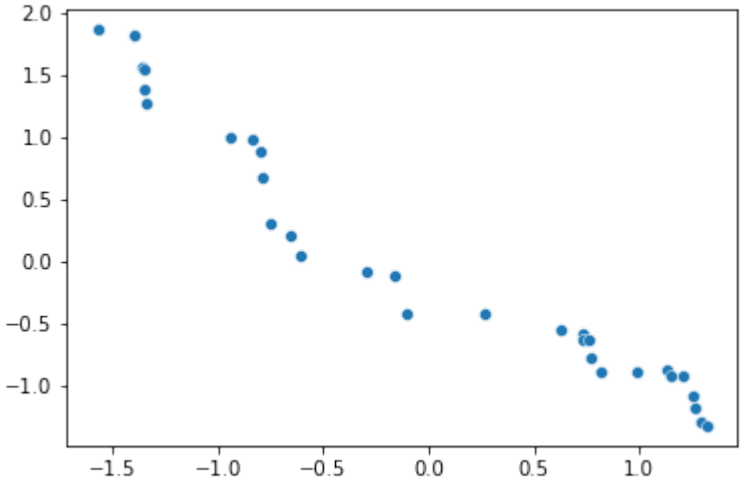
In [8]:
```
df.loc[:, 'date_int'] = pd.to_numeric(df.loc[:, 'date'])
```

Now standardize the `'date_int'` and `'seconds'` columns. Store the result as `arr_std`. You can ignore the resulting `DataConversionWarning`.

In [10]:
```
from sklearn.preprocessing import StandardScaler
arr_std = StandardScaler().fit_transform(df.loc[:, ['date_int', 'seconds']])
```

Now generate a scatterplot of the standardized data.

In [11]:
```
sns.scatterplot(x=arr_std[:,0], y=arr_std[:,1]);
```



In our example you can see that the greatest variance of the data is along a line from top left to bottom right. You can see this in the following image.

The second component would be a line perpendicular to this principal component. But this line would be much shorter, because its length corresponds to the variance in direction. You can see immediately that the first component describes almost all of the variance (i.e. the relevant information) in the data. This line may remind you of a linear regression. Despite the optical similarity, however, it is a completely different algorithm. One difference is that the values on both axes are the same, so there is no need to look for a way to transform the x-values to y-values. Instead, we are looking for a way to represent the values.

To get the direction of the greatest variance, we now need `PCA` from `sklearn.decomposition` with an important parameter (link to documentation):

```
PCA(n_components = int # number of principal axes or required dimensions of
the transformed data
    )
```

This is a *transformer*, so `PCA` has a `model.fit()` and a `model.transform()` method. Instantiate `PCA` under the name `model`. Set the hyperparameter `n_components` to `1`.

Then fit the model to the data in `arr_std`. Since this is unsupervised learning, you only need to pass `arr_std`. There is no target vector.

In [12]:
```
from sklearn.decomposition import PCA
model = PCA(n_components=1)
model.fit(arr_std)
```

Out[12]:
```
PCA(n_components=1)
```

The principal components are now in the `model.components_`. Print the attribute.

In [13]:
```
model.components_
```

Out[13]:
```
array([[-0.70710678,  0.70710678]])
```

The array in `model.components_` contains an array for each principal component with the direction of the component. In our case there is only one principal component and therefore only one array (`[[-0.70710678, 0.70710678]]`). This is indicated with a vector. We can now transform the data so that its on a line which is determined by the direction of the vector. Apply `model.transform()` to `arr_std` for this. Store the result as `arr_1d`.

In [14]:
```
arr_1d = model.transform(arr_std)
arr_1d
```

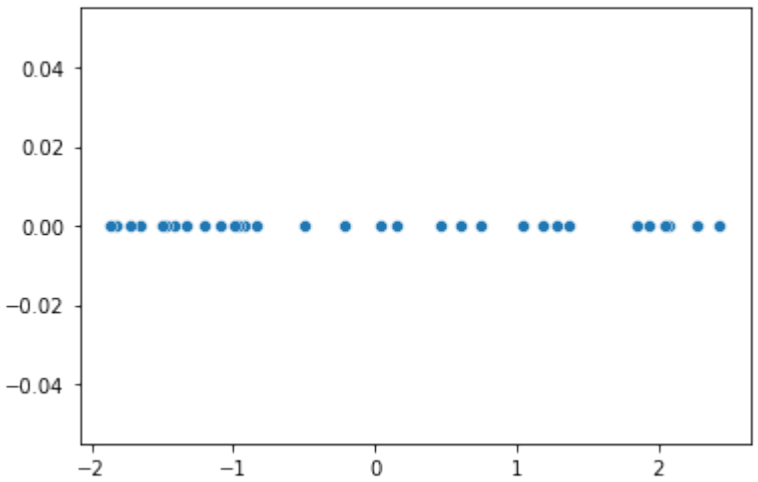
Out[14]:
```
array([[ 2.4304863 ],
       [ 2.27659048],
       [ 2.07310507],
       [ 2.04706778],
       [ 1.93149237],
       [ 1.84329643],
       [ 1.36974532],
       [ 1.28862116],
       [ 1.19053038],
       [ 1.03998153],
       [ 0.75155473],
       [ 0.60987312],
       [ 0.46190607],
       [ 0.15175212],
       [ 0.03827584],
       [-0.22073548],
       [-0.48964736],
       [-0.82909229],
       [-0.9241261 ],
       [-0.95968485],
       [-0.98526464],
       [-1.09220665],
       [-1.20776354],
       [-1.3310884 ],
       [-1.41670248],
       [-1.46569657],
       [-1.50561909],
       [-1.65790537],
       [-1.72975957],
       [-1.82158092],
       [-1.86740538]])
```

What does the transformed data look like? Create a scatter diagram again to look at it. Use the values of `arr_1d` for the x-axis. On the y-axis there should always be the same value, because

the transformed data is only one-dimensional.

Tip: With `arr_1d` each value corresponds to its own array. Therefore use `arr_1d[:, 0]`, for example, to access the data you need as an individual array.

In [15]:
```
import numpy as np
sns.scatterplot(y=0, x=arr_1d[:,0]);
```



We can see that the data points are now on one line. The distances between some points are small and large between others. The larger the distances, the easier it is to distinguish between the data points. The line lies in the direction of greatest variance and the data points were projected onto this. The following figure shows a section of the data and its transformation:

 Projection of data

The blue dots correspond to the original data after standardization. The line shows the direction of the greatest variance. The arrows show how the original data is projected onto this line. The result is the transformed data, which is displayed as orange crosses. Since these are only on one line, the data has only one dimension. We can see immediately that the distances between the points are smaller than we would expect from a dimensionality reduction. It is also interesting to note that the principal axis was learned by the `PCA` algorithm by searching for a line that maximizes the mean distances of the data points on the principal axis. $%By using the direction of the greatest variance in the data for this line, the distances between the transformed data points are as large as possible. So we should lose as little information as possible.$

Now it would be interesting to know how much of the information has been preserved. `PCA` measures this by the explained variance. This is stored for each component in the `my_model.explained_variance_ratio_` attribute. How high is this value with our data? Print it.

In [16]:
```
model.explained_variance_rataio_
```

Out[16]:
```
array([0.98436117])
```

With a single principal component, we can reproduce about 98% of the variance in the data. So we've lost less than 2% of the information, but have saved 50% of the storage space. This is because the year and the swimming record show a high correlation. In this way they provide basically the same information. The aim of dimensional reduction is to sift out this kind of redundant information.

If we only work with 2-dimensional data, it is not really that important to reduce the data's dimensions. However, if the data has hundreds or thousands of features, dimensionality reduction can be useful for the following applications:

- Reducing the amount of storage space needed and increasing speed
- Reducing noise in the data
- Visualizing high-dimensional data
- Extracting relevant features from the data

Congratulations: You have learned about the principal component analysis and reduced the dimensions of the data as a result. You have seen that `PCA` projects the data in a way that preserves as much of the variance as possible. Next, we'll integrate dimensionality reduction into our data pipeline.

Remember:

- Standardize data before `PCA`
- Indicate the number of features (principal components) with `PCA(n_components=1)`
- Evaluate the quality of the projection with `model.explained_variance_ratio_`

Do you have any questions about this exercise? Look in the forum to see if they have already been discussed.

Found a mistake? Contact Support at support@stackfuel.com.