Principal component analysis

using Python

Authors Elena Chatzimichali

Editor Raphaël Proust

2017

When analysing data, having a high number of features can potentially be a problem. Principal Component Analysis is a tool that helps working with high-dimensional data: it is an algorithm that can produce a modified dataset with fewer features but carrying as much of the "information" of the original dataset as possible.

In this course, you will learn the principles behind Principal Component Analysis and how to apply it to datasets using `pandas` and `sklearn`.

Contents

Setup Instructions

Primer on Pandas

Throughout the bootcamp we will be working with a python library called `pandas` that is very useful for data handling. To get the most from the course, it is worth to have a look at this primer[1]. that covers most of the functionality that we will be using. (To open it, you will need to download the file and use Jupyter, see at the bottom of this Readme the explanations on how to use Jupyter to open a notebook).

Install Python

Install Anaconda from: `https://www.continuum.io/downloads` (any version is fine). This includes Python and the necessary libraries we will be using: `numpy`, `matplotlib`, `pandas`.

Install Packages with pip

Open your terminal and check whether you have the "pip" function installed by typing `pip -V` (and enter). Anaconda installs `pip` so you should have it.

You may need to use `sudo pip install` (for OSX, *nix, etc) or run your command shell as Administrator (for Windows) to be able to perform the installation of the following individual packages:

```
(sudo) pip install seaborn
```

If you already have any of the previously-mentioned libraries installed, you can update them to a newer version using the syntax:

[1] `http://gitlab.cambridgespark.com/pub/intro-to-datascience/blob/master/pandas_primer.ipynb`

```
pip install <package> --upgrade
```

where <package> can be any of the libraries mentioned above.

Finalise the setup

Open and run the `load_libraries.ipynb` file, and check whether the libraries have been successfully loaded.

To execute the notebook, in your terminal run:

```
jupyter notebook load_libraries.ipynb
```

- You can run the notebook document step-by-step (one cell a time) by pressing **shift + enter**.
- You can run the whole notebook in a single step by clicking on the menu Cell -> Run All.
- To restart the kernel (i.e. the computational engine), click on the menu Kernel -> Restart. This can be useful to start over a computation from scratch (e.g. variables are deleted, open files are closed, etc...).
- Click on the menu Help -> User Interface Tour for an overview of the Jupyter Notebook App user interface.

1 Principles of PCA

The amount of data generated each day has been growing exponentially over the past years. This may result in the generation of a large number of input features, a subset of which may be highly correlated, repetitive, not very informative or even related to the particular study. Datasets with large numbers of variables tend to present high dimensionality, correlations and redundancy.

The sheer dimensionality of datasets collected or generated today are often bottlenecks for analyses: be it by Machine Learning models or statistics algorithms, high dimensionality slows down processing and can even degrade the quality of the results. From a large number of features, you may find highly correlated or repetitive subsets – in which case your algorithms waste time processing redundant information. You may also find a subset unrelated to the problem you are trying to solve – in which case your algorithms waste time processing useless and noisy information.

The most widely used technique to reduce the dimensionality a dataset is *Principal Component Analysis* (PCA). The PCA algorithm reduces the initial number of (possibly correlated) variables into a new, lower number of linearly uncorrelated (orthogonal) variables, known as the *Principal Components* (PCs).

The output of PCA is commonly used to identify patterns and trends in the data (used for exploratory data analysis or unsupervised learning), but is also often fed as input into state-of-the-art predictive models (supervised learning).

1.1 Example of PCA application: spreading out data points

Consider the example of the dataset represented in the left-hand side of the following figure.

This set of points is unevenlly distributed in space: points are set in a pattern, grouped around a diagonal line. PCA identifies the main direction (formally known as the *eigenvector*) of that pattern. Replotting the points in the using a coordinate system based on that main component and an orthogonal vector leads to the figure on the right.

In this example, PCA is used to recast the dataset onto a different set of axes. These new axes are *linear combinations* of features from the original dataset. The axes are orthogonals (which reduces correlation) and chosen so that the data is more spread out (which increases variance).

The example presents a dataset with only two features (so that it can be represented visually), however, PCA is commonly applied to datasets high dimensionality.

2 Practicals of PCA

In this module, you will apply the PCA algorithm to the retail dataset. You will then analyse the output of PCA to lower the number of dimensions.

2.1 Preparatory steps

You first need to load the usual data analysis libraries (`numpy` and `pandas`), and the `PCA` feature from scikit-learn.

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
```

You also need to import the ploting libraries (`matplotlib` and `seaborn`).

```
import matplotlib.pylab as plt
%matplotlib inline
import seaborn as sns
```

Then, you need to load the dataset from file.

```
customers = pd.read_csv("data/online_retail_afterEDA.csv",
                        index_col = "CustomerID")
customers.head()
```

And finally you need to select the continuous features of the dataset. As you will learn later, PCA only works for continuous features.

```
continuous_features = ['balance', 'max_spent', 'mean_spent',
                       'min_spent', 'n_orders', 'total_items',
                       'total_items_returned', 'total_refunded',
                       'total_spent']

has_returned = customers['has_returned']
customers = customers[continuous_features]
customers.head()
```

2.2 PCA in scikit-learn

In scikit-learn, PCA is implemented as a transformer object that learns n components in its `fit()` method, and can be used on new data to project it on these components. More information on how to use the `pca()` function and its parameters can be found online[1].

```
pca = PCA()
ind = ['PC'+str(i+1) for i in range(customers.shape[1])]
```

The values of the Principal Components (scores) can be computed by the `fit_transform()` (alternatively, `fit()` followed by `transform()`) function. This function returns a matrix with the principal components, where the first column in the matrix contains the first principal component, the second column the second principal component, and so on.

```
# Create the PCA scores matrix and check the dimensionality
scores = pca.fit_transform(customers)
scores = pd.DataFrame(scores, columns = ind,
                      index = customers.index)
print(scores.shape)
```

The loadings for the principal components are stored in a named element `components_`. This contains a matrix with the loadings of each principal component.

```
# Create the PCA loadings matrix and check the dimensionality
loadings = pca.components_
loadings = pd.DataFrame(loadings, columns = customers.columns,
                        index = ind)
print(loadings.shape)
```

Below is an excerpt of the table; you can obtain the full one by executing the code. The table indicates the weights of each feature for each PC. E.g., for each customer, the first PC (PC1) is computed as 0.44 times the balance plus 0.40 times the maximum spent plus 0.32 times the average spent etc.

	balance	max_spent	mean_spent	min_spent	n_orders
PC1	0.44	0.40	0.32	-0.0034	0.35
PC2	-0.02	-0.16	-0.39	-0.51	0.21

[1] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

	balance	max_spent	mean_spent	min_spent	n_orders
PC3	-0.14	0.12	0.34	0.51	-0.33
PC4	-0.05	0.55	0.22	-0.60	-0.49
PC5	0.01	-0.04	-0.03	-0.12	-0.26

3 Reducing dimensionality

Principal components are extracted in order of the amount of variance they contribute to the data matrix, which is ultimately related to *information* content, while the *noise* tends to be relegated to lower PCs.

Consequently, you can drop the last PCs without loosing too much information. But how many should you drop? You will investigate this now.

3.1 Explained variance

How many principal components do you need to keep? How many can you discard? You can make this decision by looking at a useful measure: the explained variance. The explained variance is the proportion of variance extracted by each successive Principal Component – or, in other words, how much information is captured by each Principal Component. This measure is calculated based on the eigenvalues of each eigenvector (PC); remember, an eigenvalue measures the amount of variance captured along that direction/axis.

```
# Calculate the explained variance
exp_var = [i*100 for i in pca.explained_variance_ratio_]

# Calculate the cumulative variance
cum_var = np.cumsum(pca.explained_variance_ratio_*100)

# Combine both in a data frame
pca_var = pd.DataFrame(data={'exp_var': exp_var,
                             'cum_var': cum_var},
                       index=ind)
pca_var.head(10)
```

Remember that Principal Components are defined in order of the amount of variance.

You can observe from the table below that the first Principal Component (PC1) accounts for the maximum variance (over 50% of the total variance),

the second PC for the second maximum variance (over 20% of the total variance), and so on.

Component	Explained variance
PC1	53.45%
PC2	20.91%
PC3	14.99%
PC4	4.37%
PC5	2.85%
PC6	1.86%
PC7	1.12%
PC8	0.46%
PC9	0.00%

You can also plot the explained variance using a barplot with `seaborn`:

```
# Plot the explained variance per PC using a barplot
fig = plt.figure(figsize=(10,7))

ax = sns.barplot(x=pca_var.index, y='exp_var', data=pca_var)
ax.set(xlabel='Principal Components',
       ylabel='Explained Variance')
```

3.2 Cumulative variance

The cumulative variance is obtained by adding the successive proportions of explained variance to obtain the total sum. This is an interesting measure that can help you choose a cutoff point for dropping PCs.

```
exp_var = [i*100 for i in pca.explained_variance_ratio_]
cum_var = np.cumsum(pca.explained_variance_ratio_*100)
pca_var = pd.DataFrame(data={'cum_var': cum_var}, index=ind)
# the plot
fig = plt.figure(figsize=(10,7))
ax = sns.barplot(x=pca_var.index, y='cum_var', data=pca_var)
ax.set(xlabel='Principal Components',
       ylabel='Explained Variance')
```

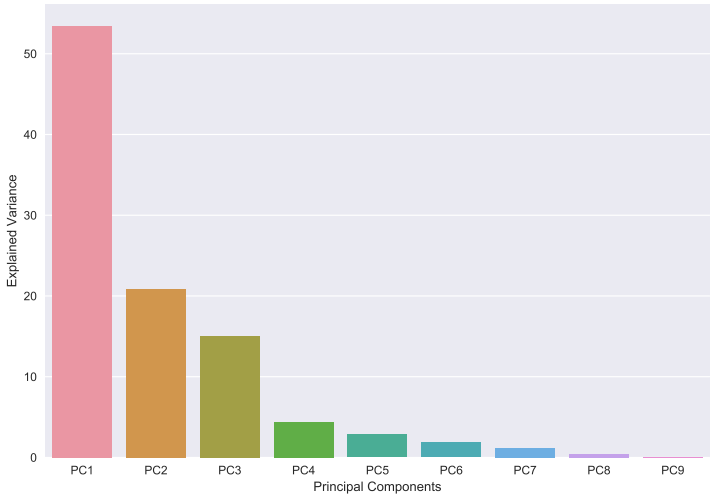

Figure 3.1: Explained variance

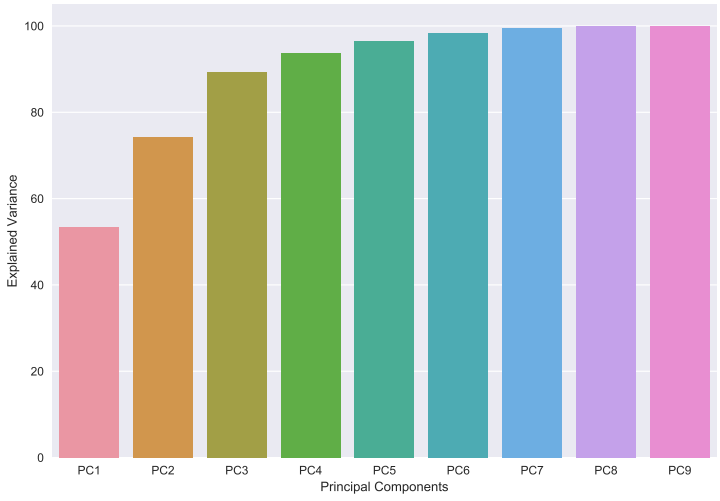

Figure 3.2: Cumulative variance

4 Exporting the results

You have applied an important transformation on the dataset. You can preserve these changes for further tasks – such as Machine Learning algorithms – by saving the new dataset on disk.

Writing a `pd.Dataframe` to disk is very easy - you just use the `.to_csv()` method, and specify the file path to where you want it saved.

```
# Save to a csv file with the '.to_csv()' method
# and give the file a name you want
scores_withresponse.to_csv('data/pca_scores.csv')
```

Key points

PCA is an algorithm used for dimensionality reduction and feature extraction. It involves:

- Projection of high-dimensional data into a lower-dimensional subspace.
- Orthogonal transformation of possibly correlated into linearly uncorrelated variables.
- The components are uncorrelated, since they are orthogonal to each other.
- The original variables are combined into a smaller number of Principal Components (PCs), which describe the data without losing essential information (minimal loss).