PCA

Learn about PCA and why it's useful for data preprocessing.

Chapter Goals:

- Learn about principal component analysis and why it's used

A. Dimensionality reduction

Most datasets contain a large number of features, some of which are redundant or not informative. For example, in a dataset of basketball statistics, the total points and points per game for a player will (most of the time) tell the same story about the player's scoring prowess.

When a dataset contains these types of correlated numeric features, we can perform principal component analysis (PCA) for dimensionality reduction (i.e. reducing the number of columns in the data array).

PCA extracts the *principal components* of the dataset, which are an uncorrelated set of latent variables that encompass most of the information from the original dataset. Using a smaller set of principal components can make it a lot easier to use the dataset in statistical or machine learning models (especially when the original dataset contains many correlated features).

B. PCA in scikit-learn

Like every other data transformation, we can apply PCA to a dataset in scikit-learn with a transformer, in this case the `PCA` module. When initializing the `PCA` module, we can use the `n_components` keyword to specify the number of principal components. The default setting is to extract *m - 1* principal components, where *m* is the number of features in the dataset.

The code below shows examples of applying PCA with various numbers of principal components.

```
# predefined data
```

```
# predefined data
print('{}\n'.format(repr(data)))

from sklearn.decomposition import PCA
pca_obj = PCA() # The value of n_component will be 4. As m is 5 and default is always m-1
pc = pca_obj.fit_transform(data).round(3)
print('{}\n'.format(repr(pc)))

pca_obj = PCA(n_components=3)
pc = pca_obj.fit_transform(data).round(3)
print('{}\n'.format(repr(pc)))

pca_obj = PCA(n_components=2)
pc = pca_obj.fit_transform(data).round(3)
print('{}\n'.format(repr(pc)))
```

In the code output above, notice that when PCA is applied with 4 principal components, the final column (last principal component) is all 0's. This means that there are actually only a maximum of three uncorrelated principal components that can be extracted.

Time to Code!

The coding exercise in this chapter uses `PCA` (imported in backend) to complete the `pca_data` function.

The function will apply principal component analysis (PCA) to the input NumPy array, `data`.

Set `pca_obj` equal to `PCA` initialized with `n_components` for the `n_components` keyword argument.

Set `component_data` equal to `pca_obj.fit_transform` applied with `data` as the only argument. Then return `component_data`.

```
def pca_data(data, n_components):
  # CODE HERE
  pass
```