Unsupervised Learning – Part 2

ESM3081 Programming for Data Science

Seokho Kang

성균관대학교
SUNG KYUN KWAN UNIVERSITY(SKKU)

Learning algorithms covered in this course

- **Data Preprocessing and Scaling**

- **Unsupervised Learning**
  - **Dimensionality Reduction & Visualization**
    - **(Projection) Principal Component Analysis (PCA)**
    - (Manifold Learning) t-distributed Stochastic Neighbor Embedding (t-SNE)
    - …
  - **Clustering**
    - K-Means
    - Hierarchical Clustering
    - DBSCAN
    - …

Principal Component Analysis

Principal Component Analysis

- **Principal component analysis (PCA)** is a method that rotates the dataset in a way such that the rotated features are statistically uncorrelated.

- PCA reduces a set of features by removing the overlap of information between the original features.
  - Create new features that are **linear** combinations of the original features.
  - These linear combinations are uncorrelated, and only a few of them contain most of the original information.
  - The directions of linear combinations are called principal components.

Principal Component Analysis

- **Mathematical interpretation of PCA –**
  Eigendecomposition of the covariance matrix of the data
  - Finding the eigenvectors with the largest eigenvalues
  - Eigenvectors – principal components
  - Eigenvalues – variances explained by principal components
  - All pairs of eigenvectors have zero correlation

Principal Component Analysis

- **Pseudocode**

  Given a (training) dataset $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$ such that $\boldsymbol{x}_i = (x_{i1}, \dots, x_{id}) \in \mathbb{R}^d$ is the *i*-th input vector of *d* features

  (assume $\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i = \boldsymbol{0}$)

  Goal: Find r-dim projection that best preserves variance $(r \leq d)$

  1. Compute covariance matrix $\mathbf{S}$ of original data points in the training dataset D
  2. Compute eigenvectors and eigenvalues of $\mathbf{S}$
  3. Select top r eigenvectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_r$ corresponding to the largest eigenvalues (The first/last eigenvector corresponds to the largest/smallest eigenvalue)
  4. Project each data point $\boldsymbol{x}_i$ onto subspace spanned by them

$$\boldsymbol{z}_i = (z_{i1}, \dots, z_{ir}), \qquad z_{ij} = \boldsymbol{u}_j^T \boldsymbol{x}_i$$

compression

$$\boldsymbol{x} \rightarrow \boldsymbol{z} \rightarrow \tilde{\boldsymbol{x}}$$

Principal Component Analysis

- **Mathematical Details**
  - For the projection onto a one-dimensional space, we can define the direction of the space using a d-dimensional vector $\boldsymbol{u}_1$, which we shall choose to be a unit vector so that $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$ because we are only interested in the direction defined by $\boldsymbol{u}_1$, not in the magnitude of $\boldsymbol{u}_1$ itself.
  - Each data point $\boldsymbol{x}_i$ is then projected onto a scalar value $z_{i1} = \boldsymbol{u}_1^T \boldsymbol{x}_i$.
  - Given $D = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n\}$, the variance of the projected data $\{z_{11}, z_{21}, \dots, z_{n1}\}$ is

$$\frac{1}{n}\sum_{i=1}^{n}(z_{i1} - \bar{z}_1)^2 = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{u}_1^T \boldsymbol{x}_i - \boldsymbol{u}_1^T \overline{\boldsymbol{x}})^2 = \boldsymbol{u}_1^T \mathbf{S} \boldsymbol{u}_1,$$

where

$$\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}_i \quad (= \mathbf{0}), \qquad \mathbf{S} = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \overline{\boldsymbol{x}})(\boldsymbol{x}_i - \overline{\boldsymbol{x}})^T .$$

Principal Component Analysis

- **Mathematical Details (cont.)**
  - We now maximize the projected variance $\boldsymbol{u}_1^T \mathbf{S} \boldsymbol{u}_1$ with respect to $\boldsymbol{u}_1$ with the constraint $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$.

$$\max_{\boldsymbol{u}_1} \quad \boldsymbol{u}_1^T \mathbf{S} \boldsymbol{u}_1$$
$$\text{s.t. } \boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$$

  - We introduce a Lagrange multiplier that we shall denote by λ_1, and then make an unconstrained maximization of

$$\boldsymbol{u}_1^T \mathbf{S} \boldsymbol{u}_1 + \lambda_1 (1 - \boldsymbol{u}_1^T \boldsymbol{u}_1).$$

  - By setting the gradient with respect to $\boldsymbol{u}_1$ equal to zero, we have

$$\mathbf{S} \boldsymbol{u}_1 = \lambda_1 \boldsymbol{u}_1$$

    which says $\boldsymbol{u}_1$ must be an eigenvector of $\mathbf{S}$.
  - If we left-multiply by $\boldsymbol{u}_1^T$ and make use of $\boldsymbol{u}_1^T \boldsymbol{u}_1 = 1$, we see that the variance is given by

$$\boldsymbol{u}_1^T \mathbf{S} \boldsymbol{u}_1 = \lambda_1$$

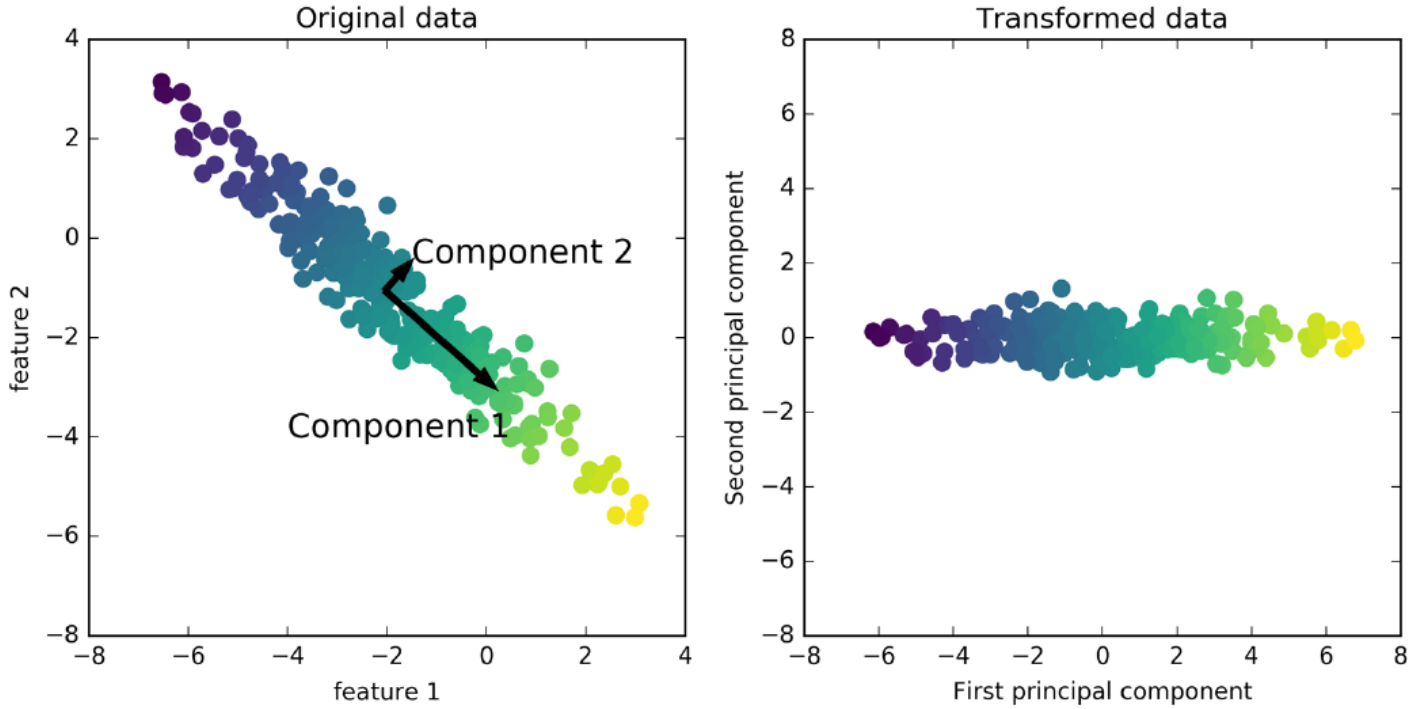
  - The variance will be a maximum when we set $\boldsymbol{u}_1$ equal to the eigenvector having the largest eigenvalue λ_1. The eigenvector $\boldsymbol{u}_1$ is known as the first principal component.

Principal Component Analysis

- **Mathematical Details (cont.)**
  - We can define additional principal components in an incremental fashion by choosing each new direction to be that which maximizes the projected variance amongst all possible directions orthogonal to those already considered.
  - If we consider the general case of a r-dimensional space, the optimal linear projection for which the variance of the projected data is maximized is now defined by the r eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r$ of the data covariance matrix $\mathbf{S}$ corresponding to the r largest eigenvalues $\lambda_1, \ldots, \lambda_r$.
  - Note: the sum of eigenvalues $\sum_{j=1}^{d} \lambda_j$ is equal to the trace of the matrix $\mathbf{S}$

Principal Component Analysis

- ***Example***: PCA on a synthetic two-dimensional dataset $(d = 2, r = 2)$
  - The original dataset is rotated so that the first principal component (PC) aligns with the x-axis and the second principal component aligns with the y-axis.
  - The first PC has the highest variation.
  - It also finds a second PC, orthogonal to the first one, that accounts for the largest amount of remaining variance.



scikit-learn Practice: *PCA*

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

sklearn.decomposition.PCA

class sklearn.decomposition.**PCA**(*n_components=None, *, copy=True, whiten=False, svd_solver='auto', tol=0.0, iterated_power='auto', random_state=None*) [source]

Principal component analysis (PCA).

Linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space. The input data is centered but not scaled for each feature before applying the SVD.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.

Notice that this class does not support sparse input. See `TruncatedSVD` for an alternative with sparse data.

Read more in the User Guide.

Parameters:

n_components : *int, float or 'mle', default=None*

Number of components to keep. if n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

If `n_components == 'mle'` and `svd_solver == 'full'`, Minka's MLE is used to guess the dimension. Use of `n_components == 'mle'` will interpret `svd_solver == 'auto'` as `svd_solver == 'full'`.

If `0 < n_components < 1` and `svd_solver == 'full'`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

If `svd_solver == 'arpack'`, the number of components must be strictly less than the minimum of n_features and n_samples.

scikit-learn Practice: *PCA*

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Methods

`fit`(X[, y])	Fit the model with X.
`fit_transform`(X[, y])	Fit the model with X and apply the dimensionality reduction on X.
`get_covariance`()	Compute data covariance with the generative model.
`get_params`([deep])	Get parameters for this estimator.
`get_precision`()	Compute data precision matrix with the generative model.
`inverse_transform`(X)	Transform data back to its original space.
`score`(X[, y])	Return the average log-likelihood of all samples.
`score_samples`(X)	Return the log-likelihood of each sample.
`set_params`(**params)	Set the parameters of this estimator.
`transform`(X)	Apply dimensionality reduction to X.

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

In [2]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=42)
```

In [3]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [4]:
```
pca = PCA(n_components=2)
pca.fit(X_train_scaled)
```

Out[4]:
```
PCA(n_components=2)
```

In [5]:
```
X_train_pca=pca.transform(X_train_scaled)
X_test_pca=pca.transform(X_test_scaled)

print("Original shape: {}".format(str(X_train_scaled.shape)))
print("Reduced shape: {}".format(str(X_train_pca.shape)))
```

```
Original shape: (426, 30)
Reduced shape: (426, 2)
```

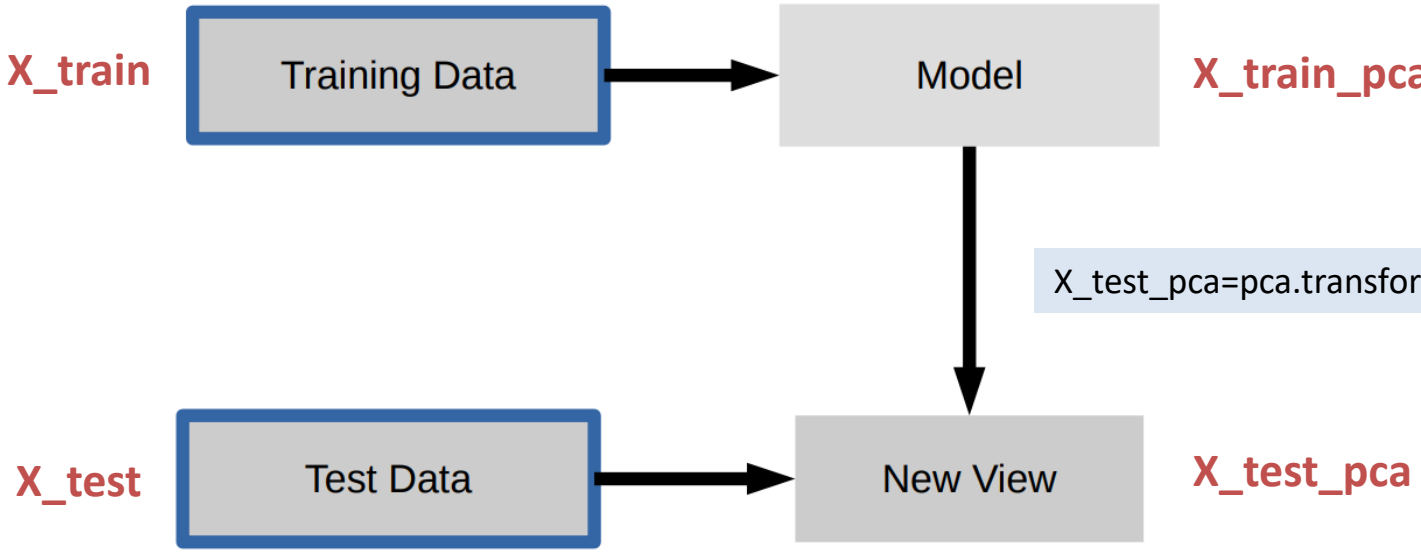
scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

```
cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, random_state=42)
```

```
pca = PCA(n_components=2)
pca.fit(X_train_scaled)
```

```
X_train_pca=pca.transform(X_train_scaled)
```



scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

Original data (X_test_scaled, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...	27	28	29
0	-0.468099	-0.141713	-0.444680	-0.485979	0.293371	0.064062	-0.094503	-0.252114	0.465735	0.155596	...	-0.173311	0.221172	0.236560
1	1.364457	0.499588	1.306438	1.334411	-0.391720	0.007650	0.261460	0.840001	-0.814742	-1.107774	...	1.029430	-0.531619	-0.994057
2	0.378785	0.066532	0.404309	0.263973	0.977745	0.385023	0.753059	0.875964	0.488134	-0.643707	...	0.602100	-0.066612	-0.179720
3	-0.487926	-0.359424	-0.429027	-0.525583	0.705429	0.565928	-0.128126	-0.522366	0.040153	1.165461	...	-0.605303	-0.523489	0.583365
4	-0.731511	-1.126145	-0.709964	-0.707875	0.306987	0.184665	-0.255992	-0.576575	0.066286	0.722173	...	-0.675541	-0.892568	-0.114232

Principal Component Coefficients (pca.components_, first 2 PCs)

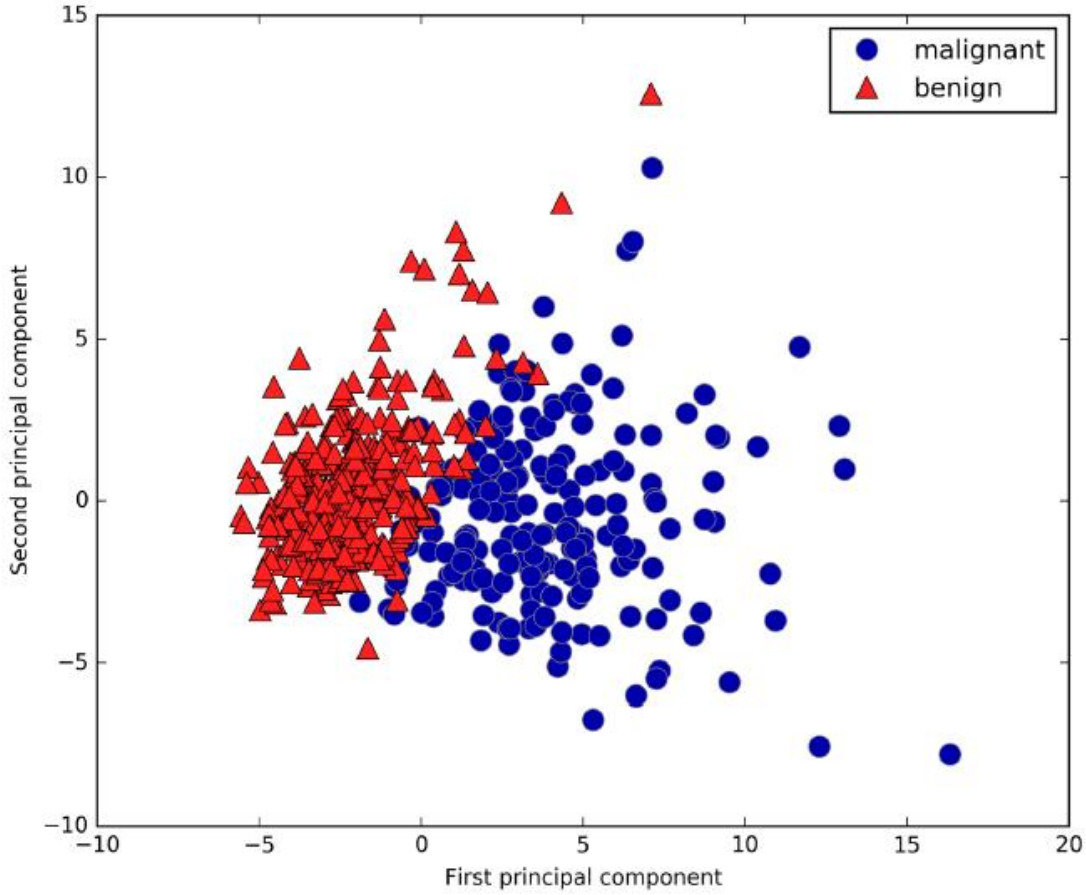
	0	1	2	3	4	5	6	7	8	9	...	27	28	29
0	0.216062	0.102568	0.225108	0.218835	0.148042	0.239289	0.259190	0.262463	0.150702	0.060383	...	0.251481	0.125004	0.125050
1	-0.238263	-0.052822	-0.220454	-0.234486	0.173699	0.155455	0.058801	-0.038336	0.175833	0.363505	...	-0.003334	0.119911	0.287219

Transformed data (X_test_pca, first 5 rows)

	0	1
0	-0.684894	0.704664
1	2.725937	-4.379125
2	1.538614	-1.006967
3	-0.801550	2.535359
4	-1.554325	2.444205

scikit-learn Practice: *PCA*

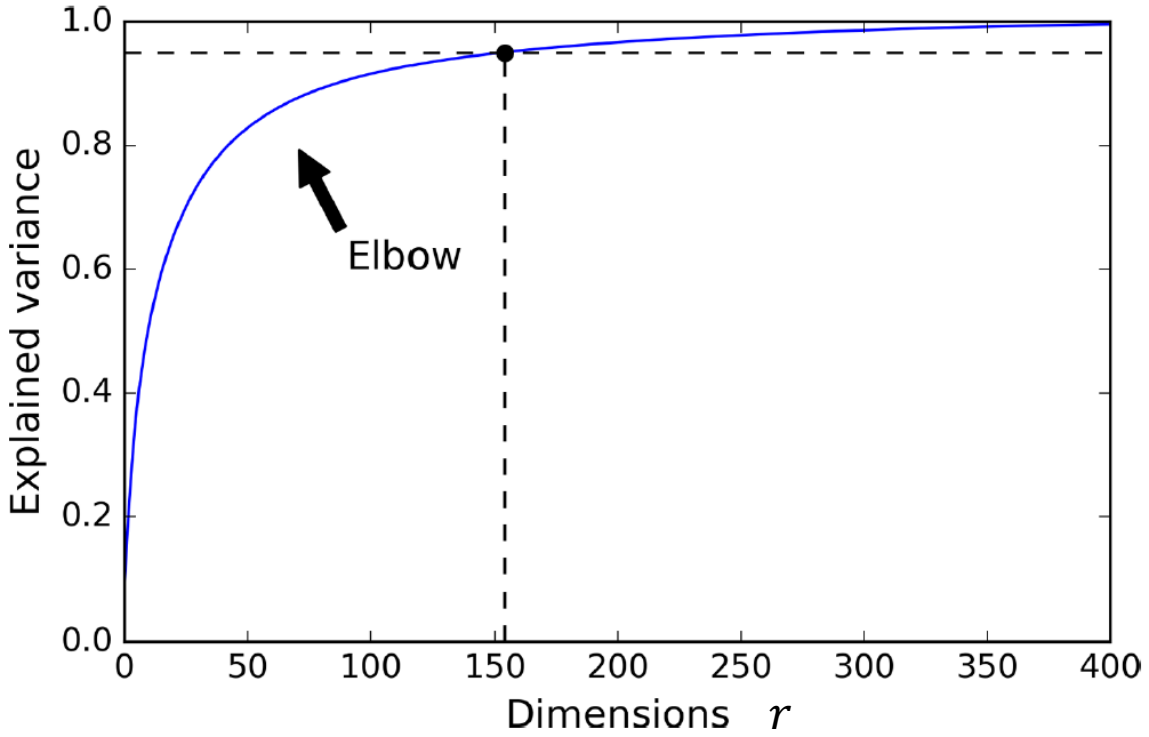
- **Example (*breast_cancer* dataset)**
  - *Two-dimensional scatter plot using the first two principal components*



Choosing the Number of PCs

- In case of reducing dimensionality for data visualization, you will generally want to reduce the dimensionality down to 2 or 3.
- Otherwise, it is generally preferable to choose the number of dimensions that add up to a sufficiently large portion of the variance (e.g., 95%)
  - ***Example:*** *Explained variance as a function of the number of dimensions*

$$\frac{\sum_{j=1}^{r} \lambda_j}{\sum_{j=1}^{d} \lambda_j}$$



scikit-learn Practice: *PCA*

https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Attributes:

components_ : *ndarray of shape (n_components, n_features)*
Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by `explained_variance_`.

explained_variance_ : *ndarray of shape (n_components,)*
The amount of variance explained by each of the selected components.

Equal to n_components largest eigenvalues of the covariance matrix of X.

New in version 0.18.

explained_variance_ratio_ : *ndarray of shape (n_components,)*
Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

instead of specifying the number of principal components you want to preserve, you can set n_components to be a float between 0.0 and 1.0, indicating the ratio of variance you wish to preserve

```
In [9]: pca = PCA(n_components=0.95)
        pca.fit(X_train_scaled)
```

```
Out[9]: PCA
        PCA(n_components=0.95)
```

```
In [10]: X_train_pca=pca.transform(X_train_scaled)
         X_test_pca=pca.transform(X_test_scaled)

         print("Original shape: {}".format(str(X_train_scaled.shape)))
         print("Reduced shape: {}".format(str(X_train_pca.shape)))
```

```
Original shape: (426, 30)
Reduced shape: (426, 10)
```

```
In [11]: pca.explained_variance_ratio_
```

```
Out[11]: array([0.43736, 0.19531, 0.09618, 0.06483, 0.05181, 0.04118, 0.02252,
                0.01698, 0.01371, 0.01197])
```

```
In [12]: sum(pca.explained_variance_ratio_)
```

```
Out[12]: 0.9518619710973647
```

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

or, you can compute PCA without reducing dimensionality, then compute the minimum number of dimensions required to preserve a certain ratio of the training set's variance

In [13]:
```
pca = PCA()
pca.fit(X_train_scaled)
```

Out[13]:
```
▾ PCA
PCA()
```

In [14]:
```
X_train_pca=pca.transform(X_train_scaled)
X_test_pca=pca.transform(X_test_scaled)

print("Original shape: {}".format(str(X_train_scaled.shape)))
print("Reduced shape: {}".format(str(X_train_pca.shape)))
```

```
Original shape: (426, 30)
Reduced shape: (426, 30)
```

In [15]:
```
pca.explained_variance_ratio_
```

Out[15]:
```
array([4.37365e-01, 1.95314e-01, 9.61800e-02, 6.48280e-02, 5.18071e-02,
       4.11845e-02, 2.25213e-02, 1.69848e-02, 1.37072e-02, 1.19706e-02,
       1.01161e-02, 9.01401e-03, 7.94309e-03, 5.20908e-03, 2.80842e-03,
       2.30760e-03, 1.96259e-03, 1.78970e-03, 1.61782e-03, 1.04904e-03,
       9.79526e-04, 8.89178e-04, 8.27362e-04, 5.64751e-04, 4.91393e-04,
       2.65341e-04, 2.26001e-04, 4.81653e-05, 2.48752e-05, 3.94240e-06])
```

In [16]:
```
sum(pca.explained_variance_ratio_)
```

Out[16]:
```
1.0
```

In [17]:
```
sum(pca.explained_variance_ratio_[:10])
```

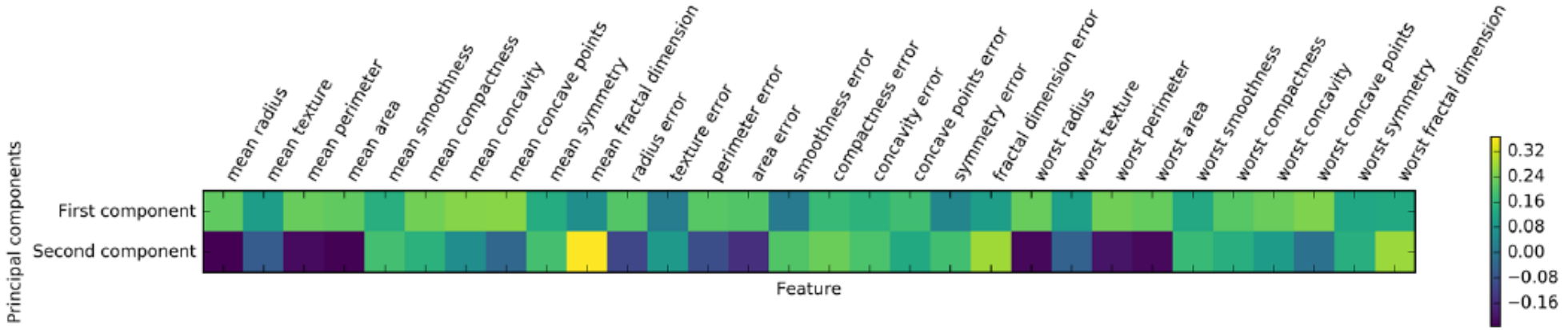
Out[17]:
```
0.9518619710973647
```

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**
  - Visualizing the coefficients of the first and second PCs using a heat map
    - There is a general correlation between all features.
    - It doesn't matter which direction (sign) the arrow points in.
    - Explaining this mixing of all features is so tricky.

```
In [17]: pca.components_[:2]

Out[17]: array([[ 0.21606219,  0.10256791,  0.22510819,  0.21883451,  0.14804204,
                  0.23928857,  0.25918989,  0.26246319,  0.15070222,  0.06038287,
                  0.20440268,  0.03117209,  0.20946517,  0.19911163,  0.02674616,
                  0.16680985,  0.15709119,  0.18483612,  0.05078459,  0.10095405,
                  0.22574568,  0.10634565,  0.23560393,  0.2235323 ,  0.13033081,
                  0.20797412,  0.23176851,  0.25148098,  0.12500401,  0.12505041],
                [-0.23826294, -0.05282173, -0.22045406, -0.23448566,  0.17369883,
                  0.15545513,  0.0588006 , -0.03833648,  0.17583295,  0.3635054 ,
                 -0.1148424 ,  0.09281816, -0.09424561, -0.15653968,  0.19724414,
                  0.23628718,  0.20051477,  0.13534311,  0.15495272,  0.28774197,
                 -0.22335781, -0.03836196, -0.20240362, -0.22167808,  0.16714483,
                  0.15270009,  0.0983546 , -0.00333393,  0.11991076,  0.28721882]])
```



PCA in Supervised Learning

- **PCA can be used as a preprocessing step for supervised learning.**
  - Can sometimes improve the accuracy of supervised algorithms.
  - Can lead to reduced memory and time consumption.

- **The main steps:**
  1. Apply PCA to training data
  2. Decide how many PCs to use
  3. Use PCs to transform validation/test data

PCA in Supervised Learning

- **Example (*breast_cancer* dataset)**

The original data have 30 features

In [19]:
```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

cancer = load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=42)
```

In [20]:
```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [21]:
```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train_scaled, y_train)
```

Out[21]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [22]:
```
y_train_hat = clf.predict(X_train_scaled)
print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))

y_test_hat = clf.predict(X_test_scaled)
print('test accuracy: %.5f'%accuracy_score(y_test, y_test_hat))
```

```
train accuracy: 0.98357
test accuracy: 0.95804
```

PCA in Supervised Learning

- **Example (*breast_cancer* dataset)**

The new data have two features
(correspond to the first two principal components)

In [23]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
pca.fit(X_train_scaled)

X_train_pca=pca.transform(X_train_scaled)
X_test_pca=pca.transform(X_test_scaled)
X_train_pca=pca.transform(X_train_scaled)
X_test_pca=pca.transform(X_test_scaled)
```

In [24]:
```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train_pca, y_train)
```

Out[24]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [25]:
```
y_train_hat = clf.predict(X_train_pca)
print('train accuracy: %.5f'%accuracy_score(y_train, y_train_hat))
y_test_hat = clf.predict(X_test_pca)
print('test accuracy: %.5f'%accuracy_score(y_test, y_test_hat))
```

```
train accuracy: 0.95775
test accuracy: 0.96503
```

PCA for Data Compression

- After dimensionality reduction with PCA, the training set takes up much less space.
- It is also possible to decompress the reduced dataset back to the original dimensions by applying the inverse transformation of the PCA projection.
  - Reconstruction of $\boldsymbol{x} = (x_1, \dots, x_d)$ from $\boldsymbol{z} = (z_1, \dots, z_r), z_j = \boldsymbol{u}_j^T \boldsymbol{x}$

$$\tilde{\boldsymbol{x}} = \sum_{j=1}^{r} z_j \boldsymbol{u}_j$$

  - This won't give you back the original data, but it will likely be quite close to the original data.

compression

$\boldsymbol{x} \rightarrow \boldsymbol{z} \rightarrow \tilde{\boldsymbol{x}}$

reconstruction

PCA for Data Compression

- **Mathematical Details**
  - Given a complete set of eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d$, each data point can be represented exactly by a linear combination of the eigenvectors

$$\boldsymbol{x} = \sum_{j=1}^{d} \alpha_j \boldsymbol{u}_j$$

  - Taking inner product with $\boldsymbol{u}_j$, we obtain $\alpha_j = \boldsymbol{x}^T \boldsymbol{u}_j$, and so without loss of generality we can write

$$\boldsymbol{x} = \sum_{j=1}^{d} (\boldsymbol{x}^T \boldsymbol{u}_j)\boldsymbol{u}_j = \sum_{j=1}^{d} z_j \boldsymbol{u}_j = \sum_{j=1}^{r} z_j \boldsymbol{u}_j + \sum_{j=r+1}^{d} z_j \boldsymbol{u}_j$$

  - Our goal is to approximate the data point using a representation involving a restricted number $r < d$ of features corresponding to a projection onto a lower-dimensional subspace. We approximate each data point $\boldsymbol{x}$ with the first r eigenvectors by

$$\boldsymbol{x} \simeq \tilde{\boldsymbol{x}} = \sum_{j=1}^{r} z_j \boldsymbol{u}_j$$

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

```
In [25]: pca = PCA()
         pca.fit(X_train_scaled)
```

(no compression)

```
Out[25]: PCA()
```

```
In [26]: X_test_pca = pca.transform(X_test_scaled)
         X_test_rec = pca.inverse_transform(X_test_pca)
```

Original data (X_test_scaled, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...
0	-0.468099	-0.141713	-0.444680	-0.485979	0.293371	0.064062	-0.094503	-0.252114	0.465735	0.155596	...
1	1.364457	0.499588	1.306438	1.334411	-0.391720	0.007650	0.261460	0.840001	-0.814742	-1.107774	...
2	0.378785	0.066532	0.404309	0.263973	0.977745	0.385023	0.753059	0.875964	0.488134	-0.643707	...
3	-0.487926	-0.359424	-0.429027	-0.525583	0.705429	0.565928	-0.128126	-0.522366	0.040153	1.165461	...
4	-0.731511	-1.126145	-0.709964	-0.707875	0.306987	0.184665	-0.255992	-0.576575	0.066286	0.722173	...

Reconstructed data (X_test_rec, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...
0	-0.468099	-0.141713	-0.444680	-0.485979	0.293371	0.064062	-0.094503	-0.252114	0.465735	0.155596	...
1	1.364457	0.499588	1.306438	1.334411	-0.391720	0.007650	0.261460	0.840001	-0.814742	-1.107774	...
2	0.378785	0.066532	0.404309	0.263973	0.977745	0.385023	0.753059	0.875964	0.488134	-0.643707	...
3	-0.487926	-0.359424	-0.429027	-0.525583	0.705429	0.565928	-0.128126	-0.522366	0.040153	1.165461	...
4	-0.731511	-1.126145	-0.709964	-0.707875	0.306987	0.184665	-0.255992	-0.576575	0.066286	0.722173	...

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

```
In [29]: pca = PCA(n_components=15)
         pca.fit(X_train_scaled)
```

(50% compression)

```
Out[29]: PCA(n_components=15)
```

```
In [30]: X_test_pca = pca.transform(X_test_scaled)
         X_test_rec = pca.inverse_transform(X_test_pca)
```

Original data (X_test_scaled, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...
0	-0.468099	-0.141713	-0.444680	-0.485979	0.293371	0.064062	-0.094503	-0.252114	0.465735	0.155596	...
1	1.364457	0.499588	1.306438	1.334411	-0.391720	0.007650	0.261460	0.840001	-0.814742	-1.107774	...
2	0.378785	0.066532	0.404309	0.263973	0.977745	0.385023	0.753059	0.875964	0.488134	-0.643707	...
3	-0.487926	-0.359424	-0.429027	-0.525583	0.705429	0.565928	-0.128126	-0.522366	0.040153	1.165461	...
4	-0.731511	-1.126145	-0.709964	-0.707875	0.306987	0.184665	-0.255992	-0.576575	0.066286	0.722173	...

Reconstructed data (X_test_rec, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...
0	-0.440428	-0.166360	-0.430651	-0.419197	0.339104	-0.090563	-0.147726	-0.207868	0.476511	0.245009	...
1	1.504630	0.403638	1.458232	1.513841	-0.469489	0.070780	0.290905	0.834965	-0.847375	-1.079679	...
2	0.546799	0.090629	0.548257	0.507273	1.038165	0.229139	0.621847	0.707702	0.477708	-0.409445	...
3	-0.598424	-0.396025	-0.557413	-0.554995	0.784465	0.375775	-0.005908	-0.233889	-0.020986	1.112887	...
4	-0.792764	-1.178386	-0.756230	-0.731848	0.286410	0.111417	-0.192210	-0.463400	0.081428	0.695957	...

scikit-learn Practice: *PCA*

- **Example (*breast_cancer* dataset)**

```
In [33]: pca = PCA(n_components=3)
         pca.fit(X_train_scaled)
```

(90% compression)

```
Out[33]: PCA(n_components=3)
```

```
In [34]: X_test_pca = pca.transform(X_test_scaled)
         X_test_rec = pca.inverse_transform(X_test_pca)
```

Original data (X_test_scaled, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...
0	-0.468099	-0.141713	-0.444680	-0.485979	0.293371	0.064062	-0.094503	-0.252114	0.465735	0.155596	...
1	1.364457	0.499588	1.306438	1.334411	-0.391720	0.007650	0.261460	0.840001	-0.814742	-1.107774	...
2	0.378785	0.066532	0.404309	0.263973	0.977745	0.385023	0.753059	0.875964	0.488134	-0.643707	...
3	-0.487926	-0.359424	-0.429027	-0.525583	0.705429	0.565928	-0.128126	-0.522366	0.040153	1.165461	...
4	-0.731511	-1.126145	-0.709964	-0.707875	0.306987	0.184665	-0.255992	-0.576575	0.066286	0.722173	...

Reconstructed data (X_test_rec, first 5 rows)

	0	1	2	3	4	5	6	7	8	9	...
0	-0.309534	-0.143593	-0.302718	-0.334790	0.109484	-0.004814	-0.140942	-0.183032	0.053695	0.228804	...
1	1.630674	0.520485	1.577223	1.628589	-0.380557	-0.041605	0.450329	0.877043	-0.367942	-1.430951	...
2	0.584940	0.139337	0.581841	0.533781	0.228403	0.309896	0.329943	0.489535	0.120298	-0.245336	...
3	-0.780359	-0.198518	-0.742683	-0.760315	0.278576	0.178178	-0.056303	-0.319153	0.308907	0.866384	...
4	-0.931776	-0.211161	-0.903297	-0.871124	0.004946	-0.098051	-0.248737	-0.552506	0.124836	0.764619	...

PCA for Data Compression

Example: *Schematic view of PCA as decomposing an image into a weighted sum of components*

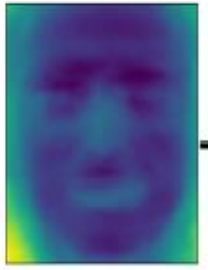
compression

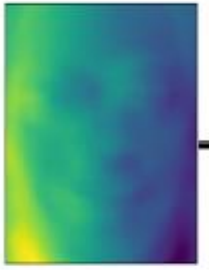
$$\boldsymbol{x} \rightarrow \boldsymbol{z} \rightarrow \tilde{\boldsymbol{x}}$$

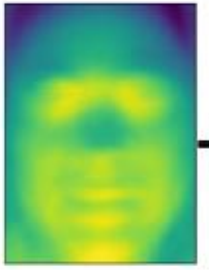
reconstruction

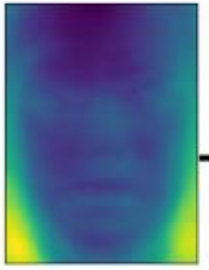
$\boldsymbol{x} \in \mathbb{R}^{5655}$

$\approx z_1 * \boldsymbol{u}_1 + z_2 * \boldsymbol{u}_2 + z_3 * \boldsymbol{u}_3 + z_4 * \boldsymbol{u}_4 + \ldots$





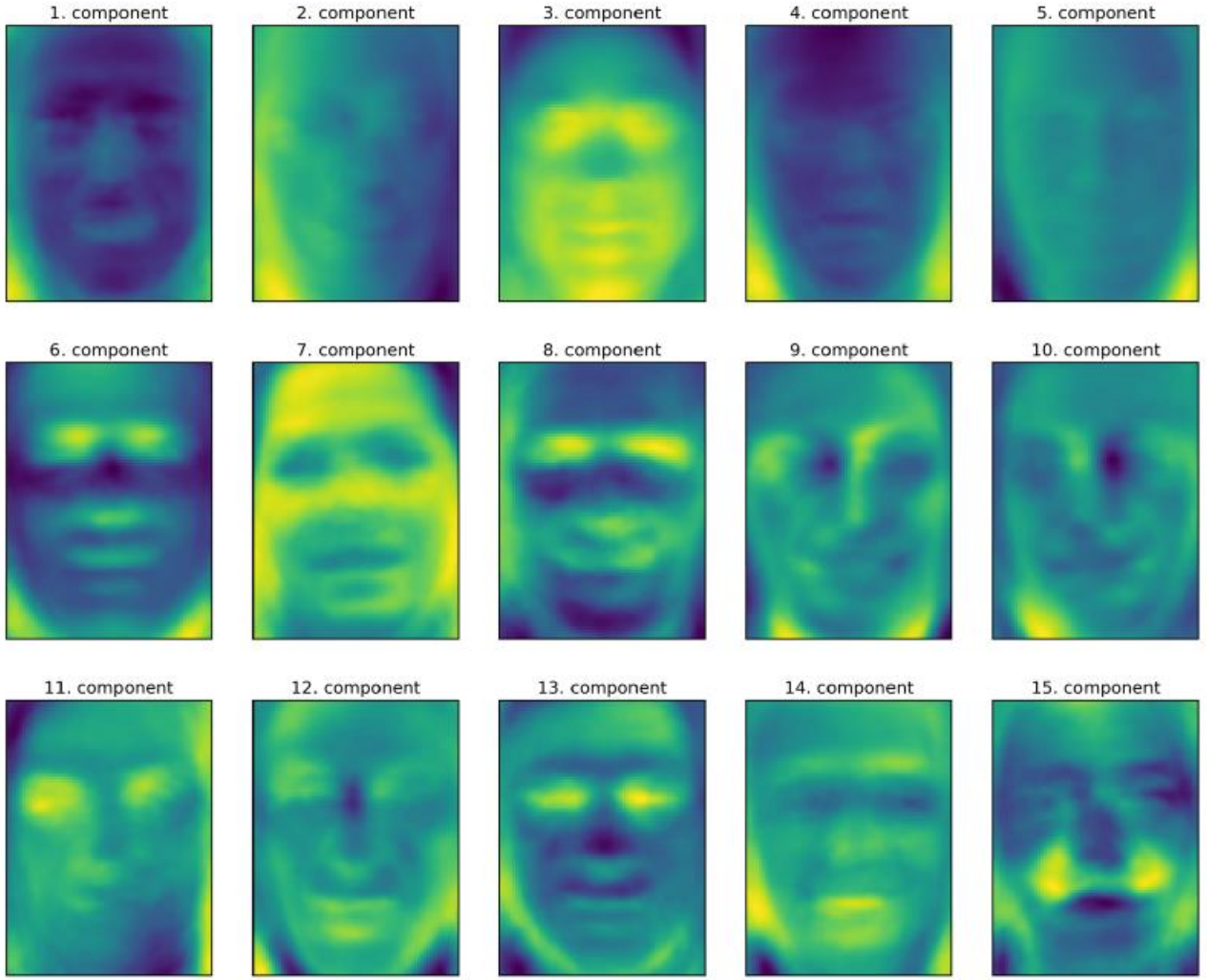




$$z_j = \boldsymbol{u}_j^T \boldsymbol{x}$$

PCA for Data Compression

Example: *Component vectors of the first 15 principal components of the faces dataset*



$$\boldsymbol{u}_j, j = 1,2, \ldots$$

PCA for Data Compression

Example: *Reconstructing three face images using increasing numbers of principal components*

$$\tilde{\boldsymbol{x}} \in \mathbb{R}^{5655}$$