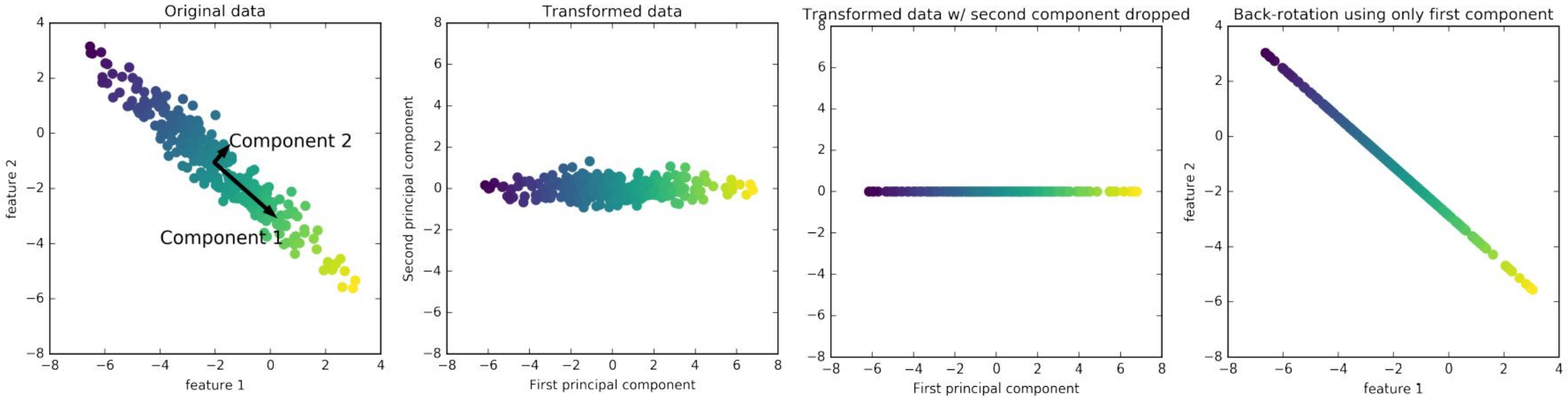
# APRENDIZAJE NO SUPERVISADO

## Transformaciones del dataset 2

La venganza de las componentes

# Principal Component Analysis (PCA)
## Análisis de componentes principales



# PCA para visualizar datos

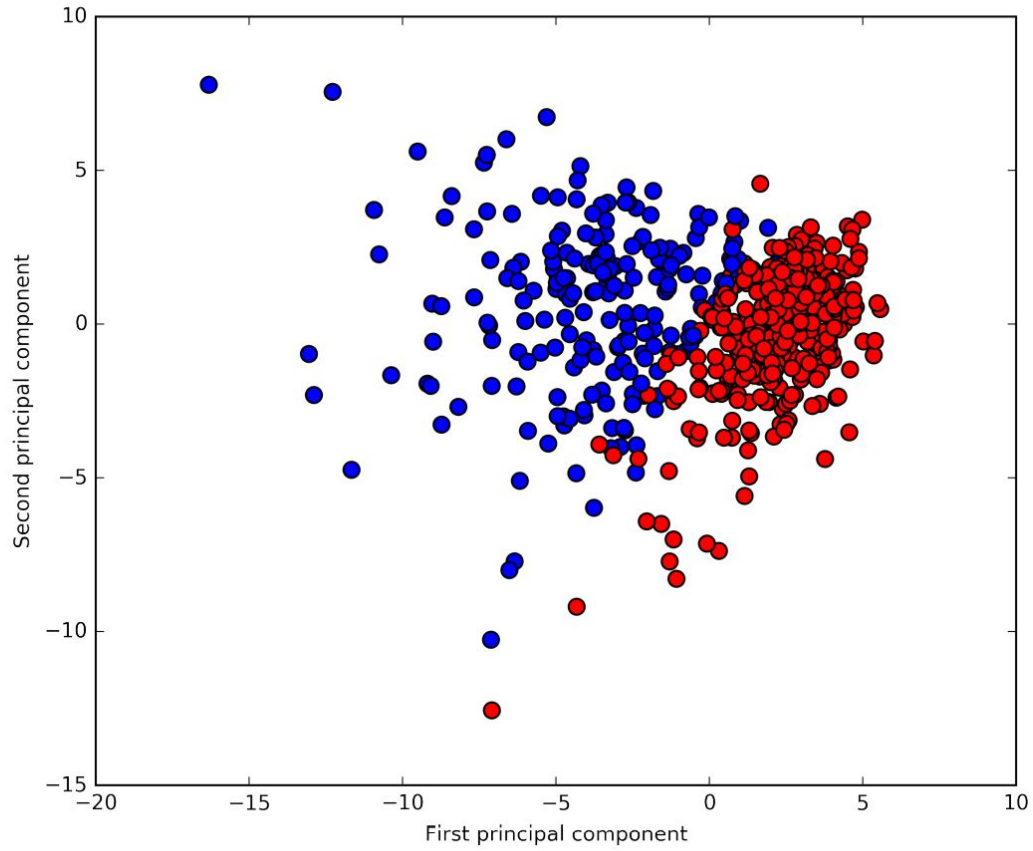
```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()
scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

```
from sklearn.decomposition import PCA
# keep the first two principal components of the data
pca = PCA(n_components=2)
# fit PCA model to beast cancer data
pca.fit(X_scaled)
# transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: %s" % str(X_scaled.shape))
print("Reduced shape: %s" % str(X_pca.shape))
```

```
Original shape: (569, 30)
Reduced shape: (569, 2)
```

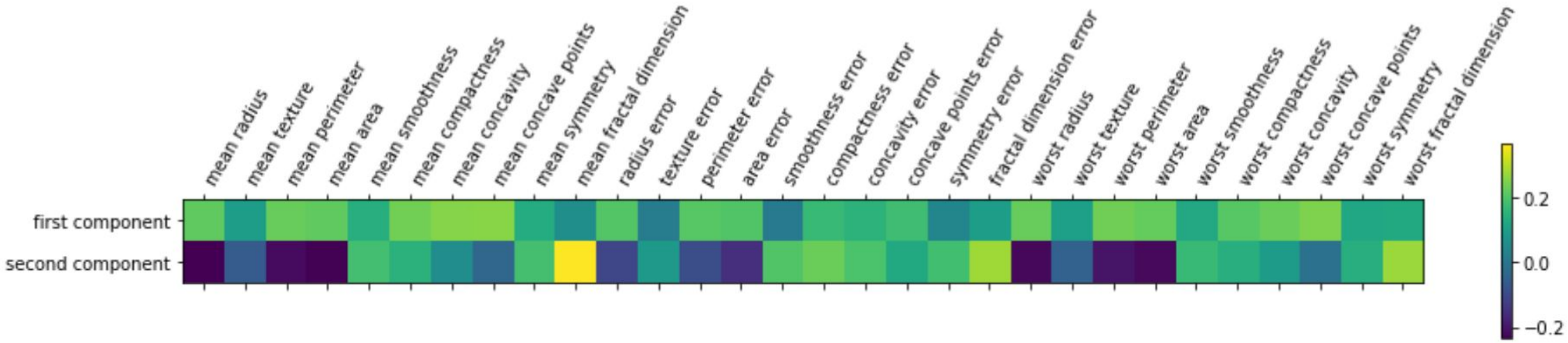
# PCA para visualizar datos

```
# plot fist vs second principal component, color by class
plt.figure(figsize=(8, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=cancer.target, cmap=mglearn.tools.cm, s=60)
plt.gca().set_aspect("equal")
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```



# Interpretando las componentes

```
plt.matshow(pca.components_, cmap='viridis')
plt.yticks([0, 1], ["first component", "second component"])
plt.colorbar()
plt.xticks(range(len(cancer.feature_names)),
 cancer.feature_names, rotation=60, ha='left');
#plt.suptitle("pca_components_cancer")
```



# Feature extraction
## Reducción de dimensionalidad

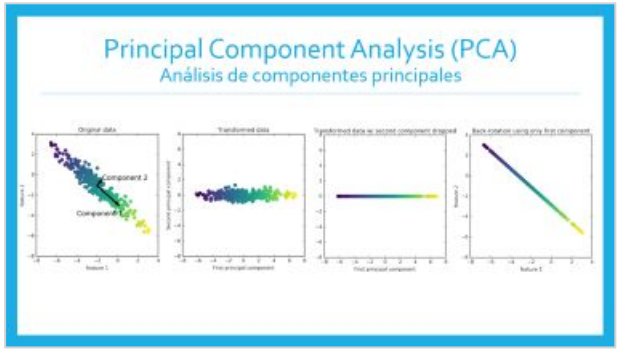
Es muy útiles con imágenes, donde cada pixel es una característica

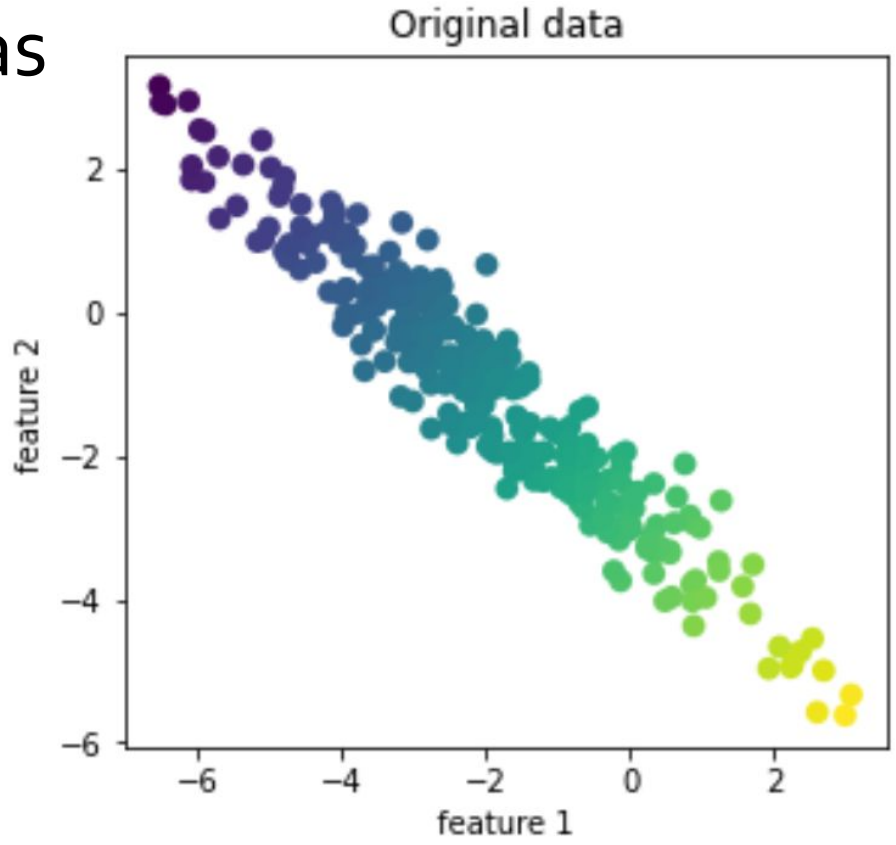
# Queremos saber si una cara pertenece a una persona en la base de datos

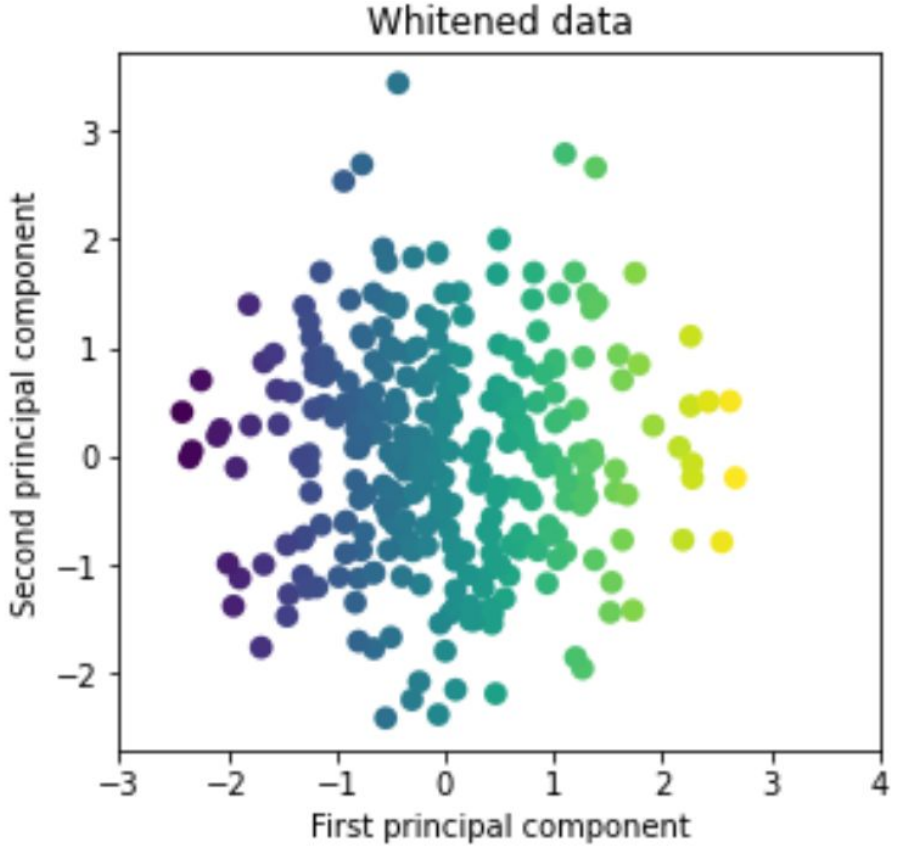
¿Cómo hacemos tal cosa?

# Usando KNeighborsClassifier

Nos conviene tener buenas distancias



# Diferencias al no usar PCA y al si usarlo

```
from sklearn.neighbors import KNeighborsClassifier
# split the data in training and test set
X_train, X_test, y_train, y_test = train_test_split(
 X_people, y_people, stratify=y_people, random_state=0)
# build a KNeighborsClassifier with using one neighbor:
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

```
0.23255813953488372
```

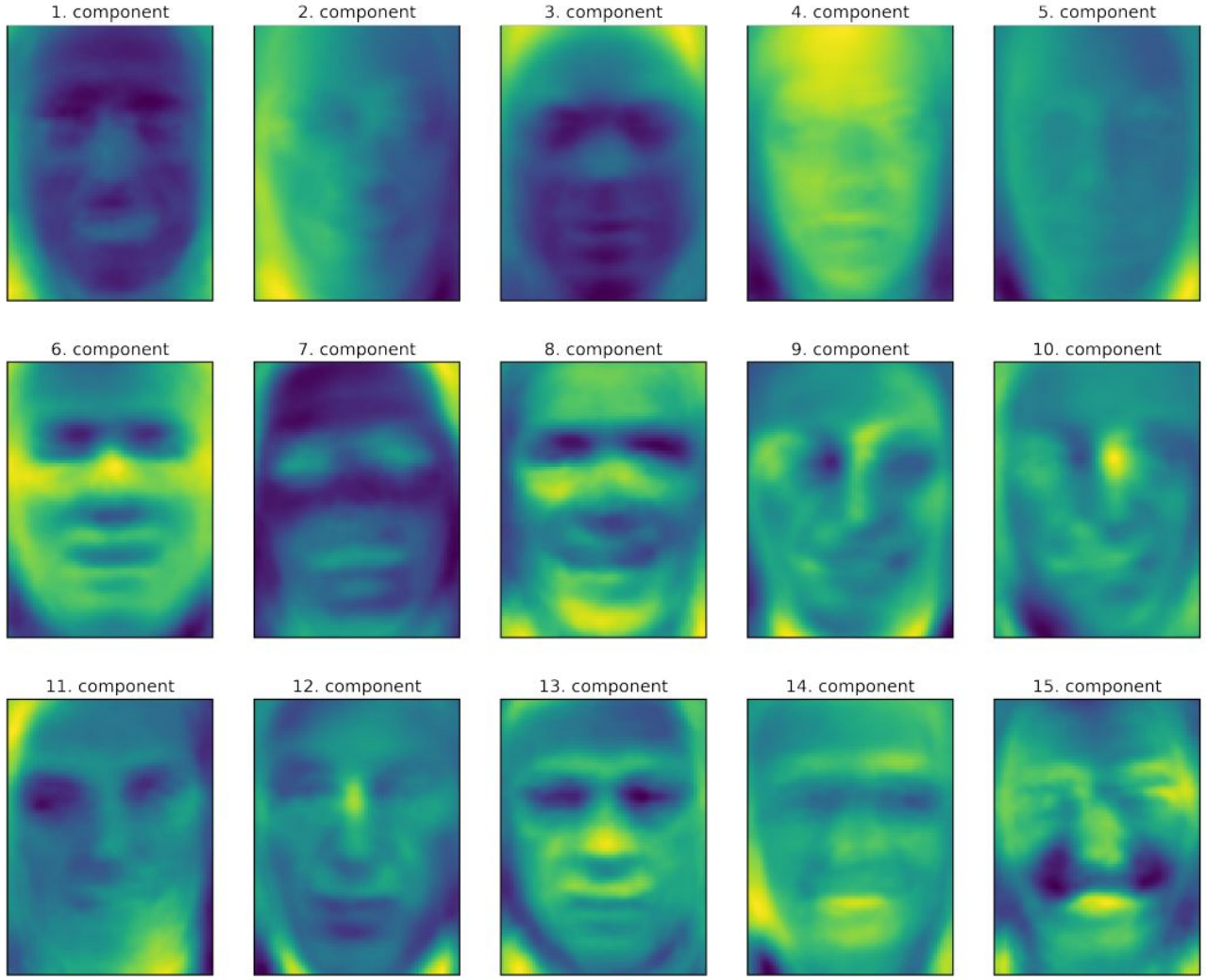
```
pca = PCA(n_components=100, whiten=True).fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape)
```

```
(1547, 100)
```

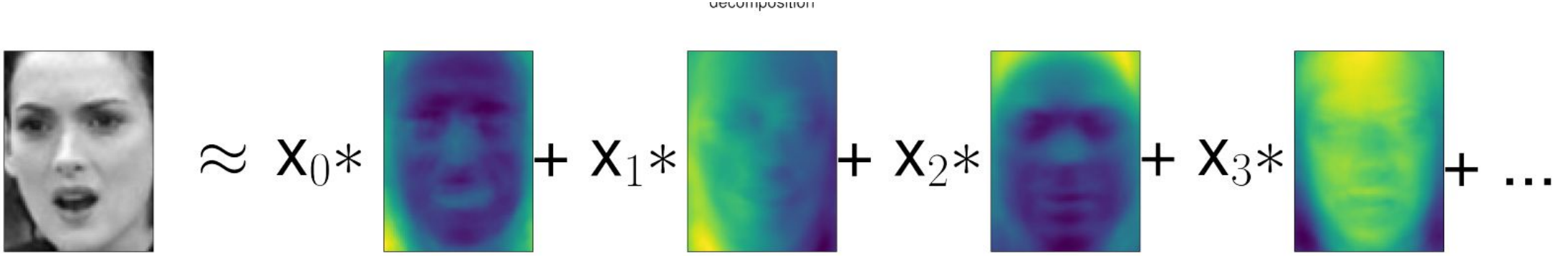
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train_pca, y_train)
knn.score(X_test_pca, y_test)
```

```
0.31007751937984496
```

# Interpretamos las componentes



# Entendiendo el modelo de otro modo



# La influencia de las componentes

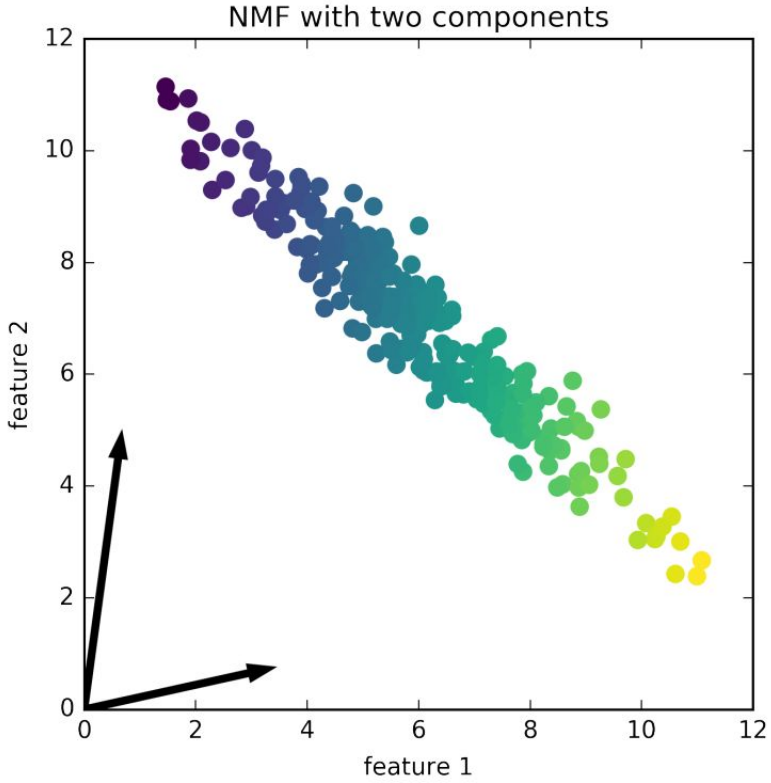
```
mglearn.plots.plot_pca_faces(X_train, X_test, image_shape)
plt.suptitle("pca_reconstructions")
```

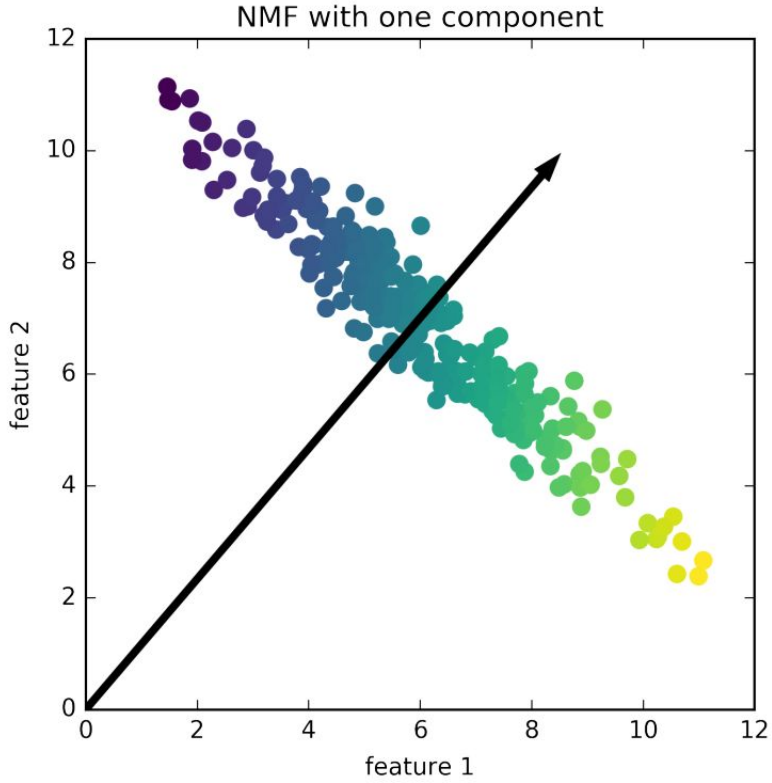
# Non-Negative Matrix Factorization (NMF)
## Factorización de matrices no negativas

- Como PCA ayuda a encontrar características útiles
- Puede ser usado para reducción de dimensionalidad
- También queremos ver cada datapoint como combinación lineal de componentes
- Queremos componentes y coeficientes no negativos
- Solo puede aplicarse a datos con características no negativas

# Las componentes en NMF
# Vectores salen el cero hacia los datos

nmf_illustration



# Comparado con PCA

**Similitudes**

- Como ayuda a encontrar características útiles
- Puede ser usado para reducción de dimensionalidad
- También queremos ver cada datapoint como combinación lineal de componentes

**Diferencias**

# Reconstruyendo los datos según el número de componentes


```
from sklearn.decomposition import NMF
nmf = NMF(n_components=15, random_state=0)
nmf.fit(X_train)
X_train_nmf = nmf.transform(X_train)
X_test_nmf = nmf.transform(X_test)
fix, axes = plt.subplots(3, 5, figsize=(15, 12),
 subplot_kw={'xticks': (), 'yticks': ()})
for i, (component, ax) in enumerate(zip(nmf.components_, axes.ravel())):
 ax.imshow(component.reshape(image_shape))
 ax.set_title("%d. component" % i)
```


# Es más fácil interpretar las componentes

# Fotos con componente 3 grande

```
compn = 3
# sort by 3rd component, plot first 10 images
inds = np.argsort(X_train_nmf[:, compn])[::-1]
fig, axes = plt.subplots(2, 5, figsize=(15, 8),
 subplot_kw={'xticks': (), 'yticks': ()})
fig.suptitle("Large component 3")
for i, (ind, ax) in enumerate(zip(inds, axes.ravel())):
 ax.imshow(X_train[ind].reshape(image_shape))
for i, (ind, ax) in enumerate(zip(inds, axes.ravel())):
 ax.imshow(X_train[ind].reshape(image_shape))
```

# Otros algoritmos similares

- Independent Component Analysis (ICA)

  Intenta maximizar la independencia entre las componentes

- Factor Analysis (FA)

  Encuentra correlaciones (interdependencias) entre las características que son dificiles de observar. Se agrupan aquellas caracteristicas que estan muy relacionadas entre si

- Sparse Coding (dictionary learning)

  Intenta encontrar una representación de los datos donde cada dato tenga pocas características

# Manifold learning with t-SNE
# Algoritmos de aprendizaje múltiple

- Son usados principalmente para visualización de los datos
- Solo puede transformar al conjunto con el que fue entrenado
- No suele usarse para aplicar aprendizaje supervisado

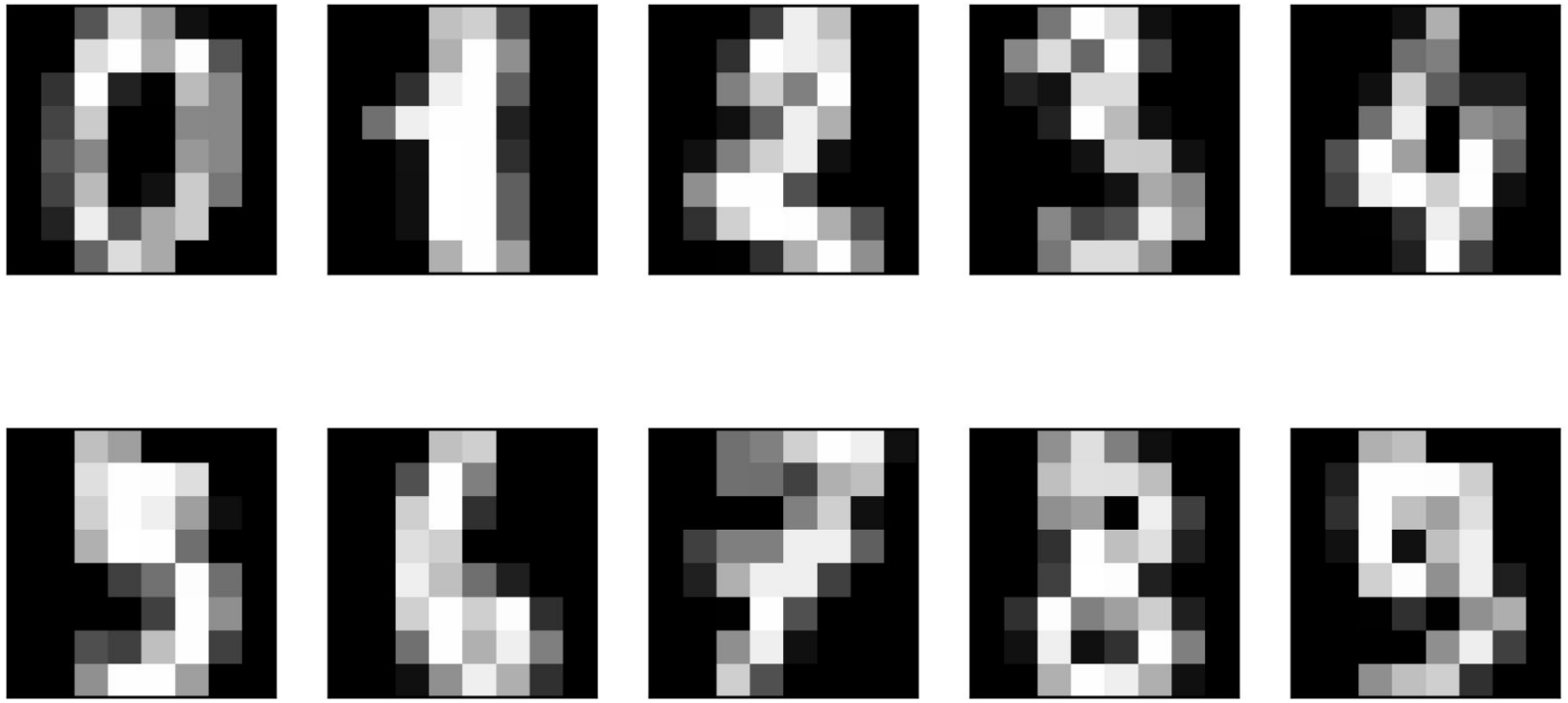
# ¿Cómo funciona T-SNE?

T-SNE intenta encontrar una representación en dos dimensiones que preserve distancias lo mejor posible.

Comienza al azar y después va acercando a aquellos que estaban cerca en el espacio original y alejando aquellos que están lejos.

# El dataset

Imágenes de 8x8 de números escritos a mano



# PCA vs t-SNE
# t-SNE no tiene método de transformación

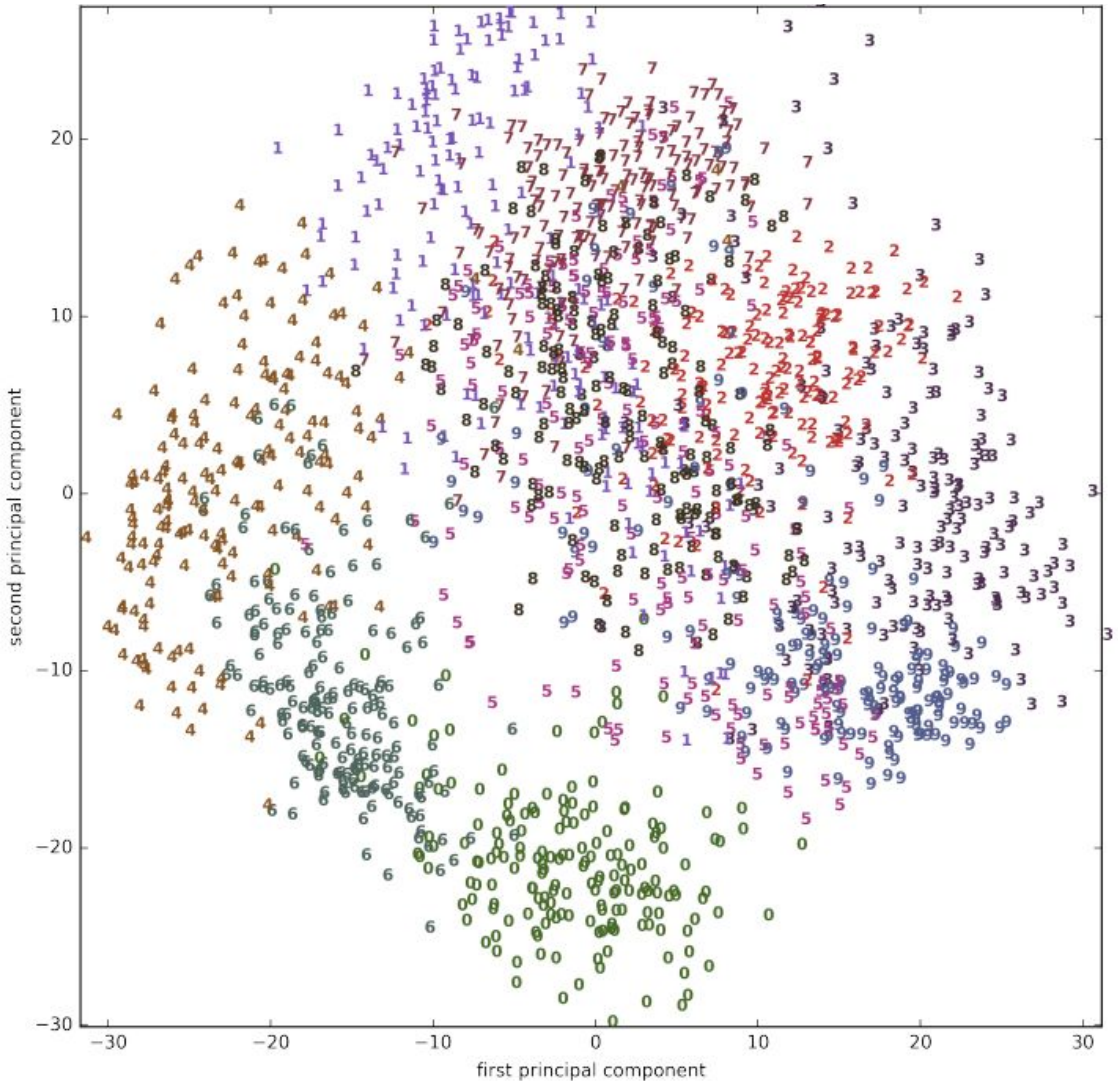
```python
from sklearn.decomposition import PCA
# build a PCA model
pca = PCA(n_components=2)
pca.fit(digits.data)
# transform the digits data onto the first two principal components
digits_pca = pca.transform(digits.data)
```

PCA

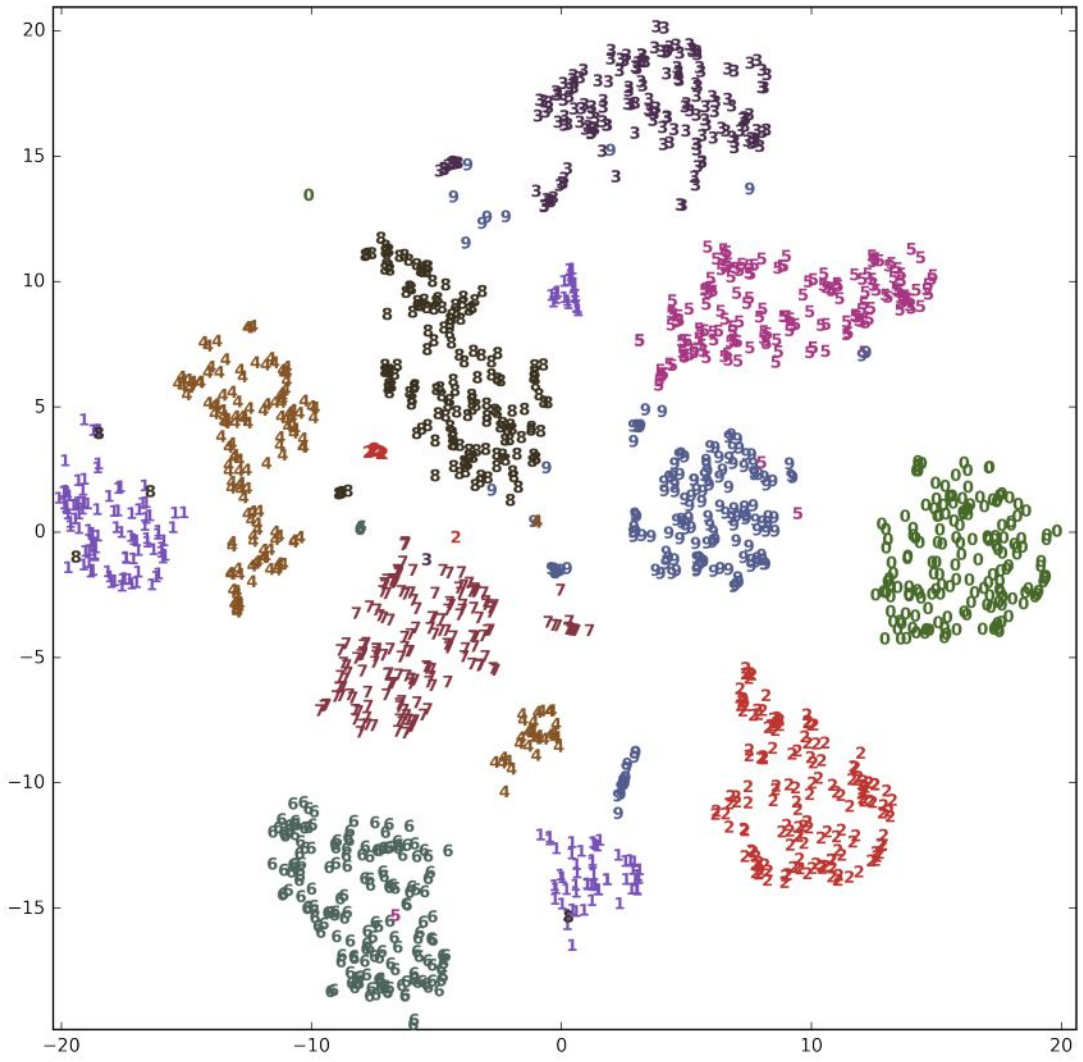
```python
from sklearn.manifold import TSNE
tsne = TSNE(random_state=42)
# use fit_transform instead of fit, as TSNE has no transform method:
digits_tsne = tsne.fit_transform(digits.data)
```

t-SNE

# PCA


second principal component
first principal component
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

N


20
15
10
5
0
−5
−10
−15
−20
−10
0
10
20