Dimensionality Reduction

Thomas Schwarz, SJ

Introduction

- Feature selection
  - Data sets contain often large numbers of features
    - Some of the features depend on each other
    - Selecting features
      - makes current classification fast
      - can generalize better from training to general data
    - This even works with Neural Networks

Introduction

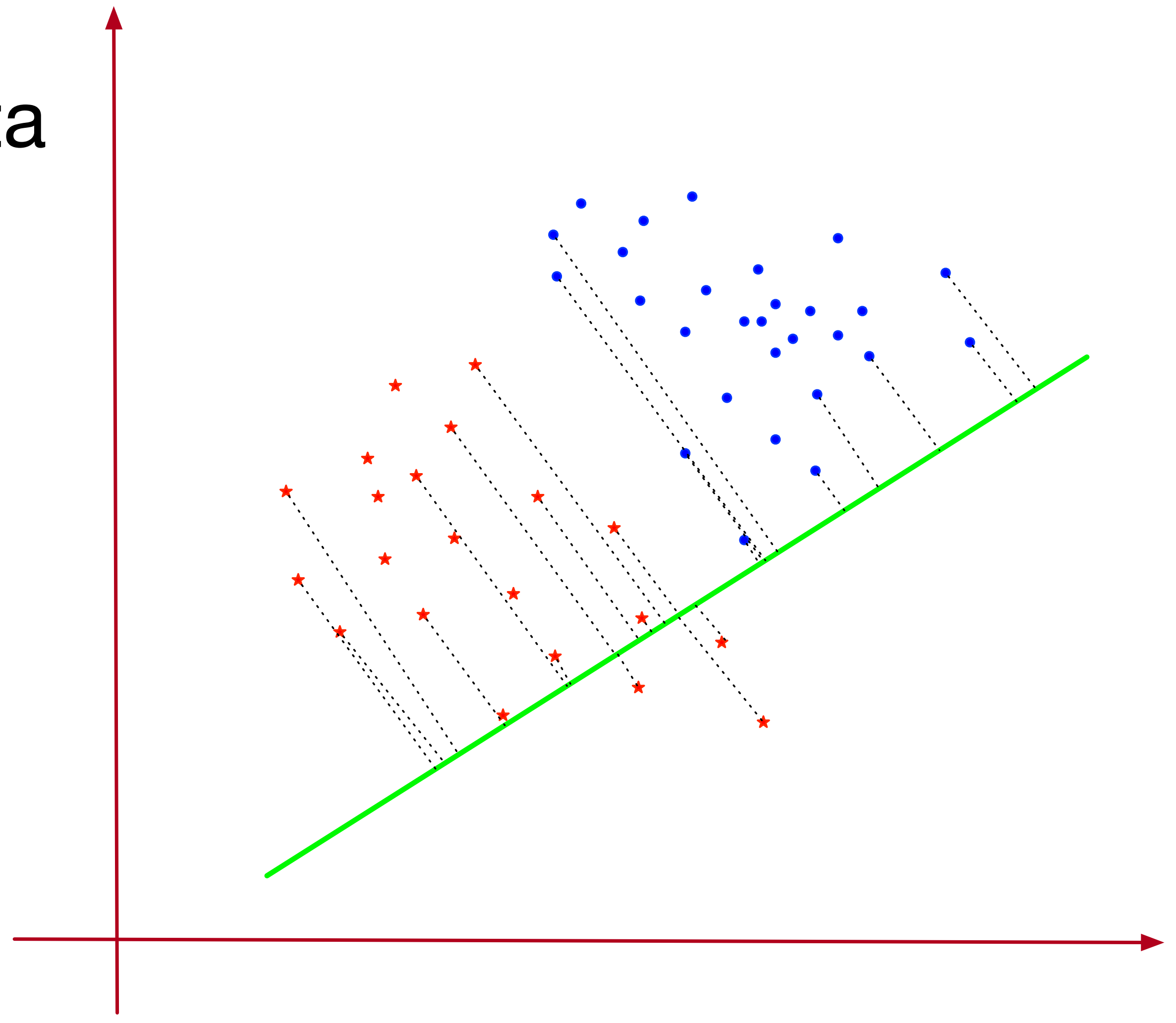
- Feature Combination:
    - Generate artificial features by combining features
        - Then do away with (some of the) old features

Introduction

- Clustering:
    - Automatic clustering
        - Groups similar data points
            - Often allows fewer features to be used

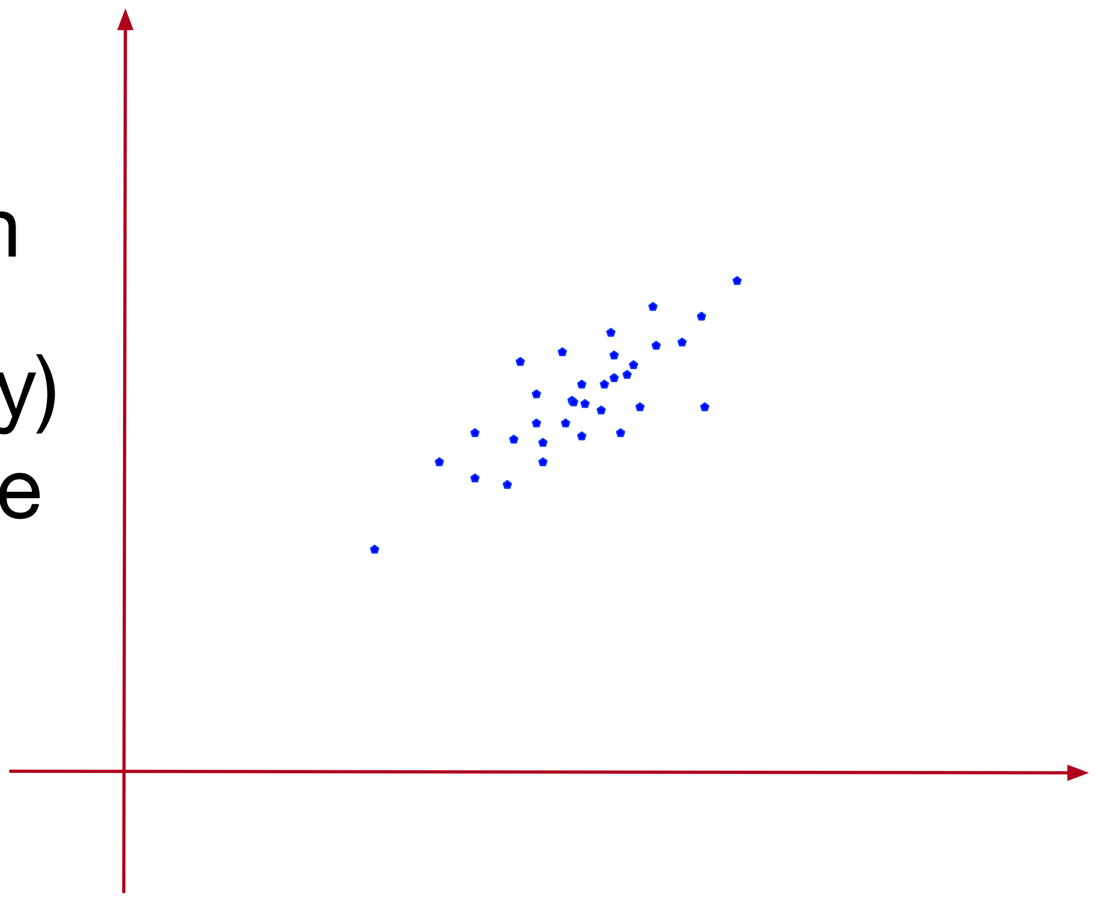
Introduction

- Automatic dimensionality reduction:
    - Project 2-dimensional data set on a single line
    - Projections separates the two data sets
    - Can use a ***single, combined*** feature for classification
        - Linear Discriminant Analysis



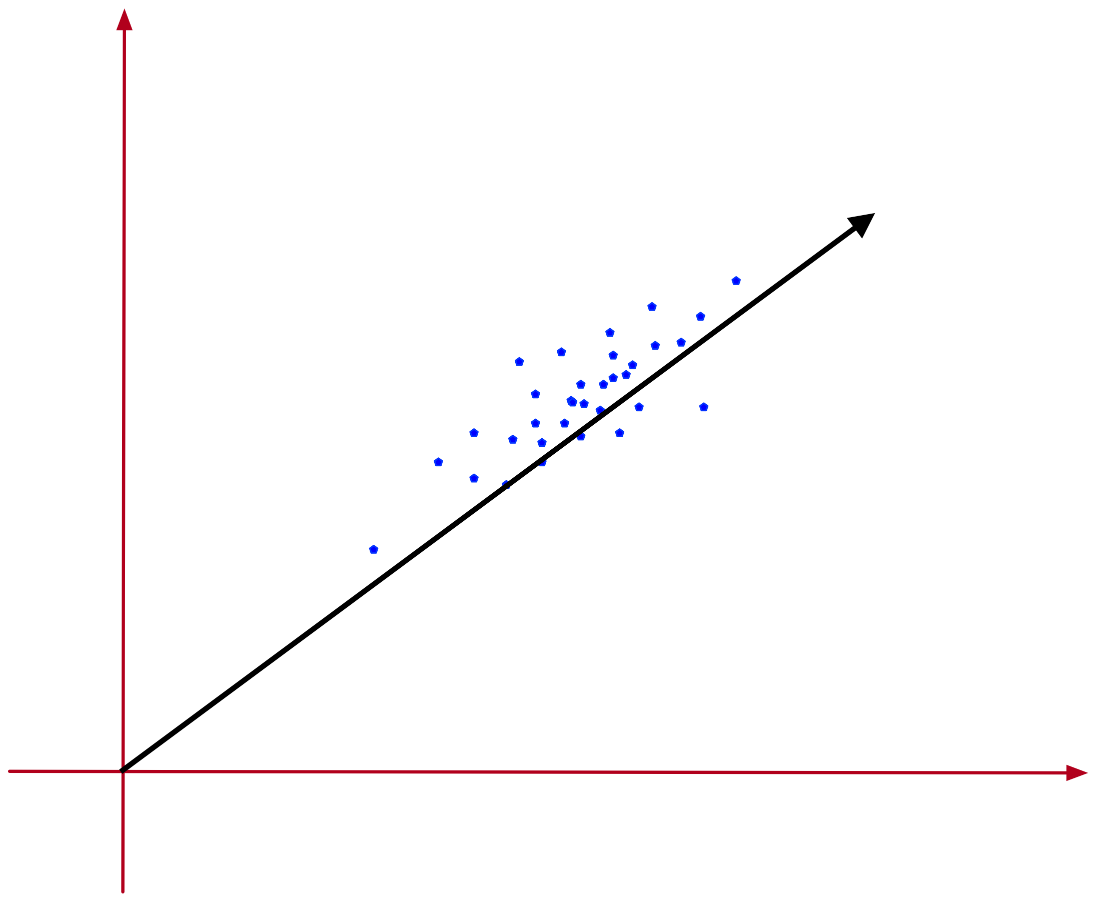
Introduction

- Two-dimensional data set
  - Spread around one dimension
  - Combine the two features (x, y) into one that has almost all the variance
    - Principal component analysis



Principal Component Analysis

- Goal:
  - Find the one direction in which the data sets varies most



Principal Component Analysis

- Given a set of U of data points with d numerical attributes
- Write as an $n \times d$ matrix

$$\mathbf{D} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ x_{3,1} & x_{3,2} & \dots & x_{3,d} \\ & \dots & & \\ x_{n,1} & x_{n,2} & \dots & x_{n,d} \end{pmatrix}$$

Principal Component Analysis

- Each data point is a linear combination of standard basis

$$\mathbf{x}_i = \sum_{j=1}^{d} x_{i,j}\mathbf{e_j}$$

- Dimensionality reduction:
    - Replace standard basis with another orthogonal matrix
    - Weight of data should be concentrated in a few dimensions

Principal Component Analysis

- Assume $(\mathbf{u}_i \,|\, i \in \{1,\ldots,d\})$ is such a basis
    - Then $\mathbf{u}_i \cdot \mathbf{u}_j = \delta_{i,j}$
    - Actually, any d vectors of length one with this property are a basis

Proof: If $\sum_{i=1}^{d} \alpha_i \mathbf{u}_i = 0$, then

$$0 = \mathbf{u}_j \cdot \sum_{i=1}^{d} \alpha_i \mathbf{u}_i = \alpha_j$$

Principal Component Analysis

- Write the vectors in an orthonormal basis as column vectors of a matrix

$$\mathbf{U} = \begin{pmatrix} | & | & \dots & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_d \\ | & | & \dots & | \end{pmatrix}$$

Then: $\mathbf{u}_i^t \mathbf{u}_j = \delta_{i,j}$ implies:

$$\mathbf{U}^T \mathbf{U} = \mathbf{1}_d$$

Principal Component Analysis

A feature vector $\mathbf{x}$ is a linear combination $\mathbf{x} = \sum_{i=1}^{d} \alpha_i \mathbf{u}_i$.

Write: $\mathbf{a} = (\alpha_1, \alpha_2, \ldots, \alpha_n)$

Then $\mathbf{x} = \mathbf{a} \cdot \mathbf{U}^t$ or equivalently $\mathbf{x}^t = \mathbf{U} \cdot \mathbf{a}^t$

Principal Component Analysis

$U = (\frac{1}{\sqrt{2}}, 0, \frac{-1}{\sqrt{2}}), (\frac{1}{\sqrt{6}}, \frac{\sqrt{2}}{\sqrt{3}}, \frac{1}{\sqrt{6}}), (\frac{1}{\sqrt{3}}, \frac{-1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$ is an orthonormal basis of $\mathbb{R}^3$

Matrix is $\mathbf{U} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{3}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{3}} \end{pmatrix}$

Principal Component Analysis

Column vector $(2,1,3)^t$ is a linear combination of the column vectors of $\mathbf{U}$.

Use $\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} = \mathbf{U} \cdot \mathbf{a}^t$

Multiply with $\mathbf{U}^t$

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{6}} & \frac{\sqrt{2}}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix} = \mathbf{U}^t \cdot \mathbf{U} \cdot \mathbf{a} = \mathbf{a}^t$$

Obtain: $(-\frac{3}{\sqrt{2}} + \sqrt{2}, 2\sqrt{\frac{2}{3}} + \sqrt{\frac{3}{2}}, \frac{1}{\sqrt{3}} + \sqrt{3}) = \mathbf{a}$

Principal Component Analysis: Projection on Subspace

Write a data point as $\mathbf{x} = \sum_{i=1}^{d} \alpha_i \mathbf{u}_i$.

Assume that we have ordered the basis by importance

We select only the first r components:

Write: $\mathrm{U}_r = \begin{pmatrix} | & | & \dots & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r \\ | & | & \dots & | \end{pmatrix}$

Then set $\pi_r(\mathbf{x}) = \sum_{i=1}^{r} \alpha_i \mathbf{u}_i = \mathrm{U}_r \cdot (\alpha_1, \alpha_2, \dots, \alpha_r)^t$

Principal Component Analysis: Projection on Subspace

Since $\mathbf{a}^t = \mathbf{U}^t \cdot \mathbf{x}$, it follows $\pi_r(\mathbf{a}^t) = \mathbf{U}_r^t \mathbf{x}^t$ and

$$\pi_r(\mathbf{x}^t) = \mathbf{U}_r \pi_r(\mathbf{a}^t) = \mathbf{U}_r \mathbf{U}_r^t \mathbf{x}^t$$

$\Pi_r = \mathbf{U}_r \mathbf{U}_r^t$ is called the projection matrix since

(a) $\Pi_r \cdot \Pi_r = \mathbf{U}_r \mathbf{U}_r^t \mathbf{U}_r \mathbf{U}_r^t = \mathbf{U}_r \mathbf{U}_r^t$

(b) $\Pi_r^t = (\mathbf{U}_r \mathbf{U}_r^t)^t = \mathbf{U}_r^{t\,t} \mathbf{U}^t = \mathbf{U}_r \mathbf{U}_r^t \Pi_r = \mathbf{U}_r \mathbf{U}_r^t$

Principal Component Analysis: Projection on Subspace

Example (continued): Project on the first two coordinates with respect to U

$$\mathbf{U}_r = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \\ 0 & \frac{\sqrt{2}}{\sqrt{3}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{6}} \end{pmatrix}$$

Principal Component Analysis: Projection on Subspace

Then we calculate the projection matrix

$$\Pi_2 = \mathbf{U}_2\mathbf{U}_2^t = \begin{pmatrix} \frac{2}{3} & \frac{1}{3} & \frac{-1}{3} \\ \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ \frac{-1}{3} & \frac{1}{3} & \frac{2}{3} \end{pmatrix}$$

Principal Component Analysis: Projection on Subspace

Projection of $\mathbf{x}^t = (2,1,3)$ is

$$\Pi_2(\begin{pmatrix} 2 \\ 1 \\ 3 \end{pmatrix}) = \begin{pmatrix} \frac{2}{3} \\ \frac{7}{3} \\ \frac{5}{3} \end{pmatrix}$$

Principal Component Analysis: Projection on Subspace

- Now we know how to project
    - Need to find the best orthonormal matrix for the projection

Single Principal Component Analysis

- There are infinitely many choices of orthonormal bases
- Start out with reduction to a single dimension
- First step: Center the data set
  - By subtracting the mean of the data set
- Therefore: **The mean of the data set is zero**

Single Principal Component Analysis

- If we reduce to a single dimension, than the partial basis is given by a **single** vector $\mathbf{u}$.
- *Optimality criterion:* Projection maximizes the variance

Single Principal Component Analysis

$$\text{var}(\{\mathbf{u}^t\mathbf{x}_i \mid i \in \{1,\ldots,n\}\}) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^t\mathbf{x}_i - \mathbf{u}^t(\overline{\mathbf{x}}))^2$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^t\mathbf{x}_i)^2 \qquad \text{(Average is zero)}$$

$$= \frac{1}{n}\sum_{i=1}^{n}(\mathbf{u}^t\mathbf{x}_i)(\mathbf{u}^t\mathbf{x}_i)^t$$

$$= \frac{1}{n}\sum_{i=1}^{n}\mathbf{u}^t\mathbf{x}_i\mathbf{x}_i^t\mathbf{u}$$

$$= \mathbf{u}^t\left(\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^t\right)\mathbf{u} = \mathbf{u}^t\Sigma\mathbf{u}$$

Single Principal Component Analysis

- Therefore: $\mathbf{u}^t \Sigma \mathbf{u} \longrightarrow \max$ subject to $\mathbf{u}^t \mathbf{u} = 1$

- Use Lagrange multiplier λ and now maximize

$$J(\mathbf{u}) := \mathbf{u}^t \Sigma \mathbf{u} - \lambda(\mathbf{u}^t \mathbf{u} - 1)$$

- So, we differentiate:

$$\frac{\delta}{\delta \mathbf{u}} J(\mathbf{u}) = 2\Sigma \mathbf{u} - 2\lambda$$

Single Principal Component Analysis

- Result: Maximum obtained if $\Sigma\mathbf{u} = \lambda\mathbf{u}$
- With other words: $\mathbf{u}$ has to be an eigenvector of Σ with eigenvalue λ.
- And to maximize, we want the eigenvector with the largest eigenvalue
-

Single Principal Component Analysis

- Turns out that finding the maximum eigenvector and eigenvalue is quite simple:
    - Write any non-zero vector as a combination of eigen-vectors
    - Then repeatedly apply the matrix, but always normalize the product
    - The coefficient corresponding to the largest eigenvalue gets more and more magnified
    - And in the limit, the product will be the eigenvector corresponding to the largest eigenvalue

Single Principal Component Analysis

- Another goodness criterion:
  - Minimize the sum of squares of the differences between projected values and original values of the feature vector
  - Error is

$$||\mathbf{x} - \Pi_1(\mathbf{x})||^2 = (\mathbf{x} - \Pi_1(\mathbf{x}))^t(\mathbf{x} - \Pi_1(\mathbf{x}))$$

Single Principal Component Analysis

$$\sum_{i=1}^{n} ||\mathbf{x}_i - \Pi_1(\mathbf{x} - i)||^2$$

$$= \sum_{i=1}^{n} (\mathbf{x}_i - \Pi_1(\mathbf{x}_i)^t(\mathbf{x}_i - \Pi_1(\mathbf{x}_i))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - 2\mathbf{x}_i^t\Pi_1(\mathbf{x}_i) + \Pi_1(\mathbf{x})^t\Pi_1(\mathbf{x}))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - 2(\mathbf{u}^t\mathbf{x}_i)(\mathbf{x}_i^t\mathbf{u}) + (\mathbf{u}^t\mathbf{x}_i)(\mathbf{u}^t\mathbf{x}_i)\mathbf{u}^t\mathbf{u})$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - 2(\mathbf{u}^t\mathbf{x}_i)(\mathbf{x}_i^t\mathbf{u}) + (\mathbf{u}^t\mathbf{x}_i)(\mathbf{u}^t\mathbf{x}_i))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2 - (\mathbf{u}^t\mathbf{x}_i)(\mathbf{x}_i^t\mathbf{u}))$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - \sum_{i=1}^{n} (\mathbf{u}^t\mathbf{x}_i\mathbf{x}_i^t\mathbf{u})$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - \mathbf{u}^t(\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^t)\mathbf{u}$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - \mathbf{u}^t(\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^t)\mathbf{u}$$

$$= \sum_{i=1}^{n} (||\mathbf{x}_i||^2) - n\mathbf{u}^t\Sigma\mathbf{u}$$

Single Principal Component Analysis

- This means:
    - In order to minimize the sum of squared errors,
    - Need to minimize the projected variance
- Our two criteria are the **same**

Dual Principal Component Analysis

- We can redo our calculation for two dimensions
- Calculate just as before the minimum variance
- Obtain: minimum variance is the sum of the two largest eigenvalues
- Need to pick the two eigenvectors with the two largest eigenvalues

PCA in Python

- Part of sklearn.decomposition
  - Import bunch of modules

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
```

  - Create random, but skewed data set

```
rng = np.random.RandomState(2020716)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
```

PCA in Python

- Here is some code to draw a vector

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=1,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)
```

PCA in Python

- Calculate the PCA (with two components)

  -
    ```
    pca = PCA(n_components=2)
    pca.fit(X)

    print(pca.components_)
    print(pca.explained_variance_)
    ```

PCA in Python

- First component has almost all the variance:

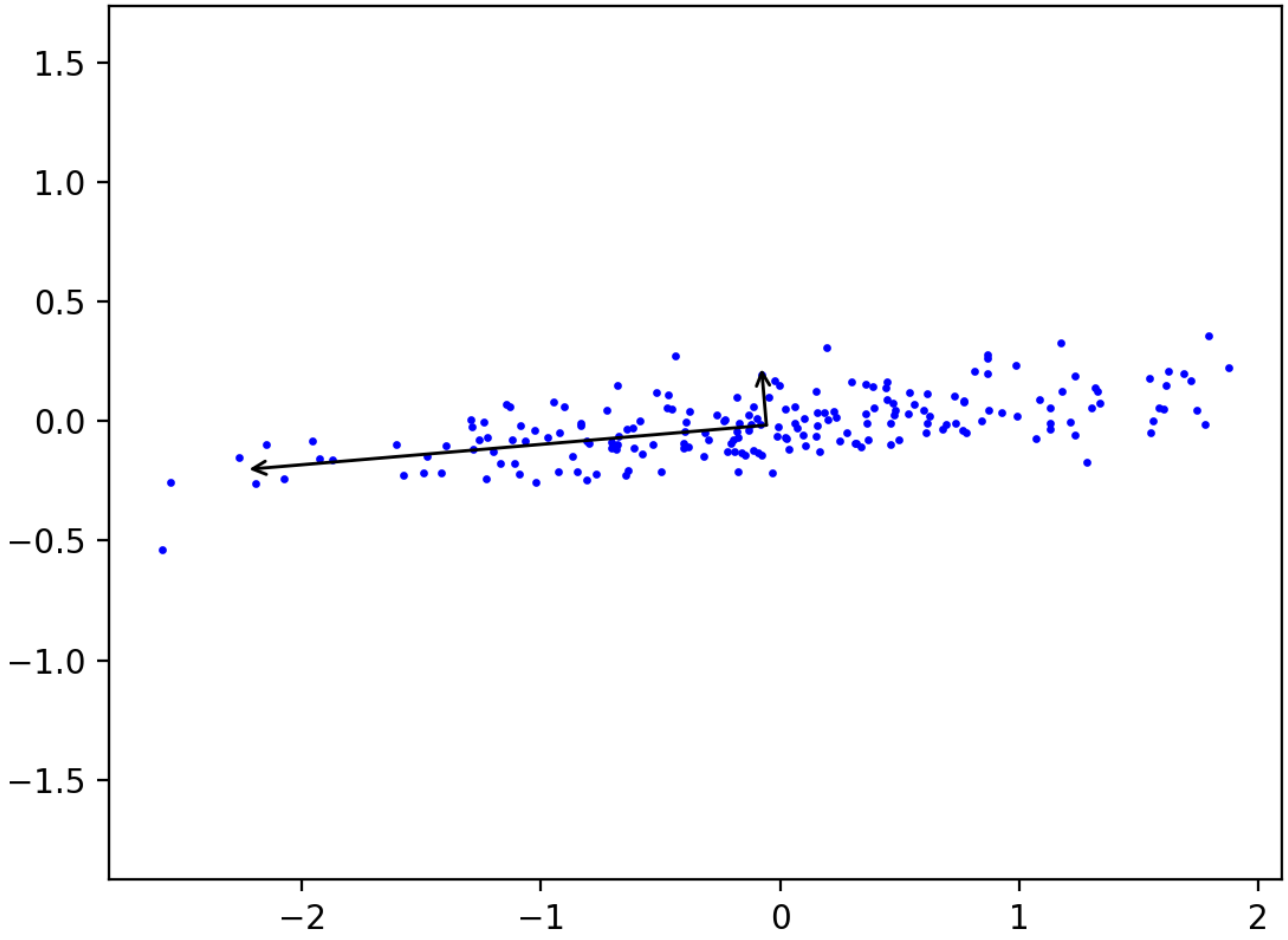
```
[[-0.99638832 -0.08491358]
 [-0.08491358  0.99638832]]
[0.89143208 0.01057402]
```

PCA in Python

- Draw everything:
-

```
plt.scatter(X[:, 0], X[:, 1], s=2, c='blue')
for length, vector in zip(pca.explained_variance_,
pca.components_):
    v = vector * 2.3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)

plt.axis('equal')
plt.show()
```

PCA in Python

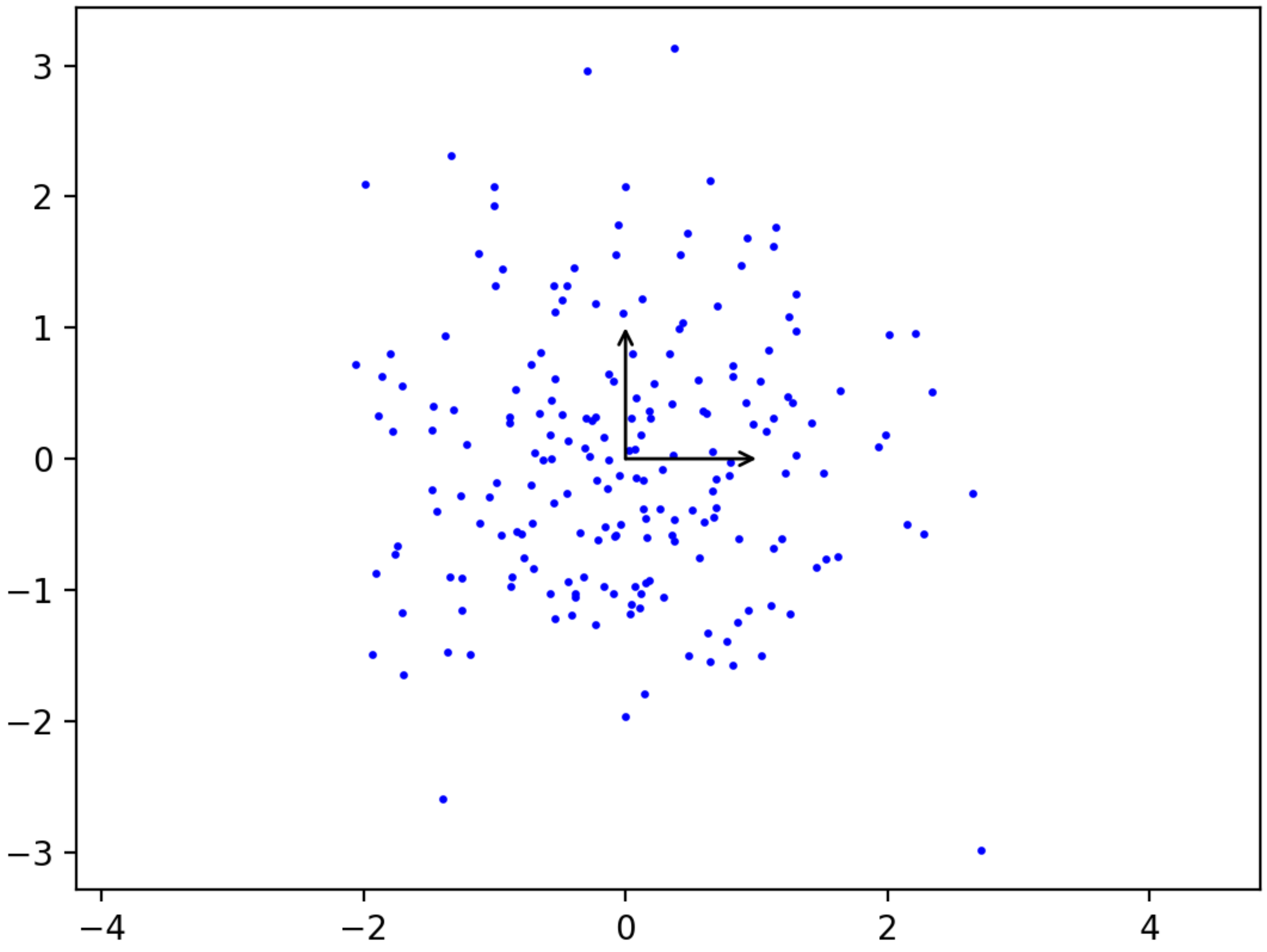


PCA in Python

- Can express data points in the new coordinates:
-

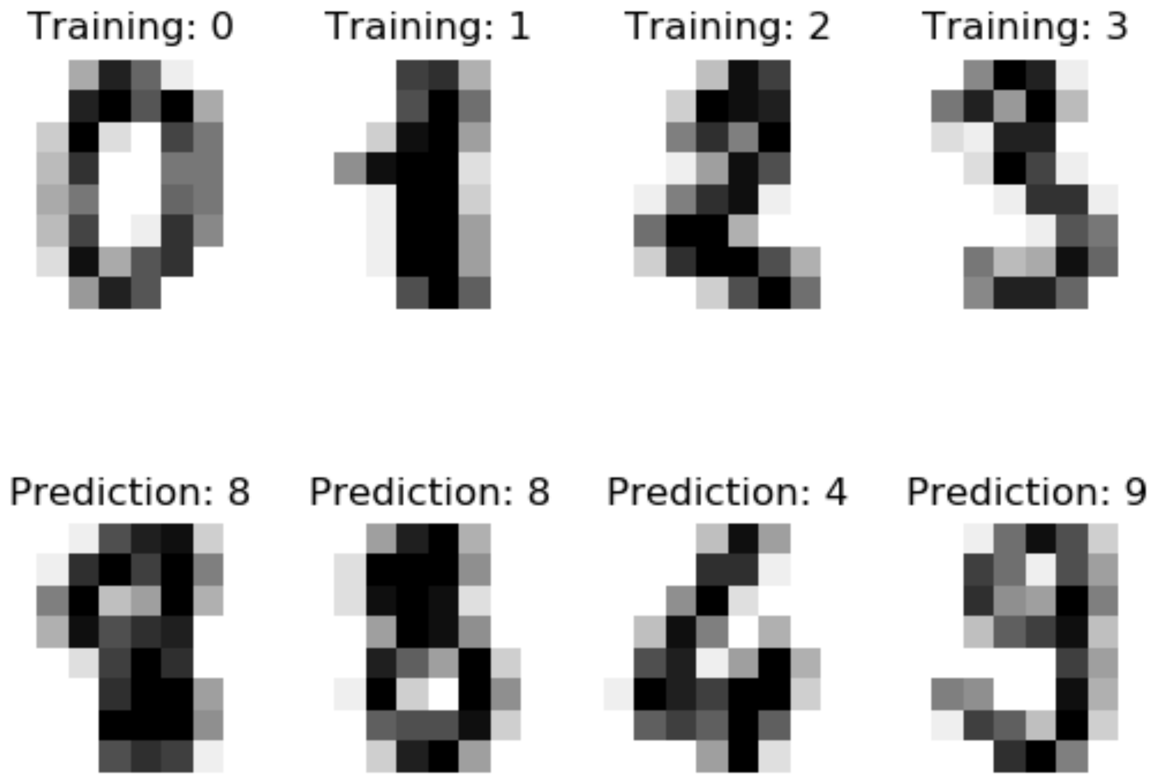
```
pca = PCA(n_components=2, whiten=True)
pca.fit(X)

X_pca = pca.transform(X)
```

PCA in Python



PCA in Python

- Sklearn has the digit data-set
    - Used for learning how to recognize digits for post-office automation, etc



PCA in Python

- Images have 64 pixels with gray values

```
from sklearn.datasets import load_digits

digits = load_digits()
```

```
>>> digits.data.shape
(1797, 64)
```

PCA in Python

- Can use PCA to lower dimension to two

```
pca = PCA(2)
projected = pca.fit_transform(digits.data)
```

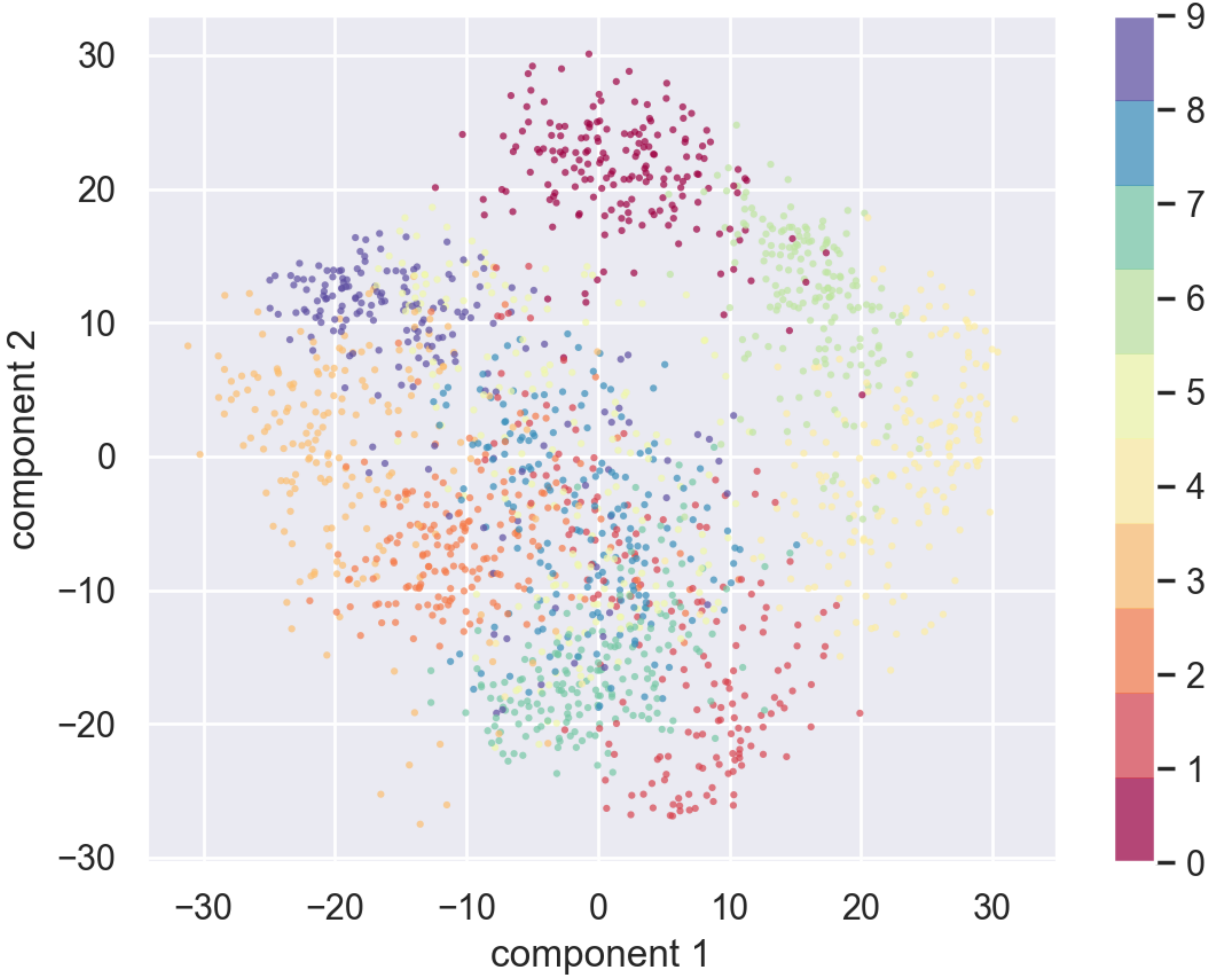
PCA in Python

- And display with the Spectral colormap

```
plt.scatter(projected[:, 0],
            projected[:, 1],
            s=5,
            c=digits.target,
            edgecolor='none',
            alpha=0.7,
            cmap=plt.cm.get_cmap('Spectral', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();

plt.show()
```

PCA in Python

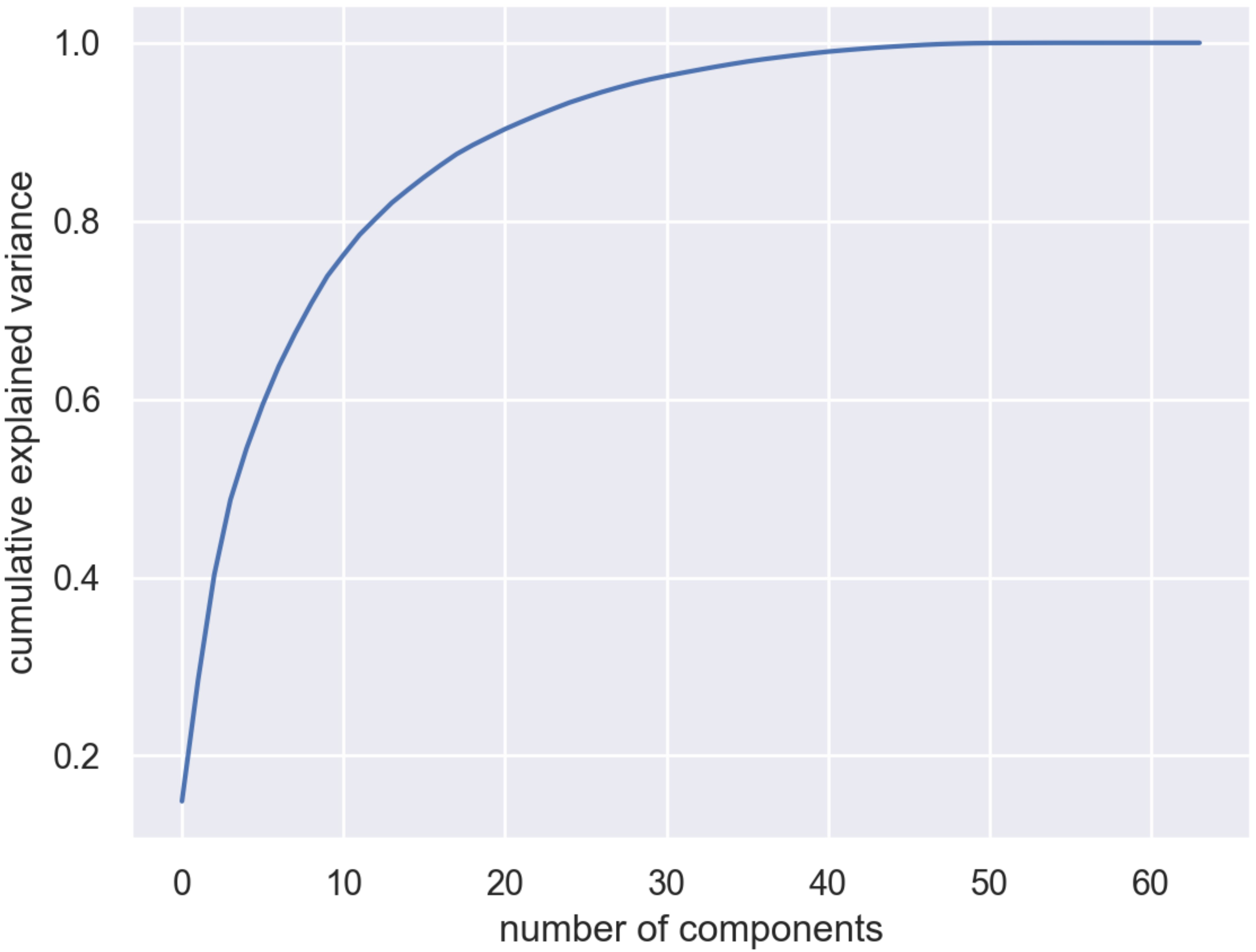
- Result shows that two features already give a decent classification:


component 1
component 2
−30
−20
−10
0
10
20
30
0
1
2
3
4
5
6
7
8
9

PCA in Python

- We can calculate the complete orthonormal base
  - And decide how many features we might need by looking at the total explained variance

```
pca = PCA().fit(digits.data)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')

plt.show()
```

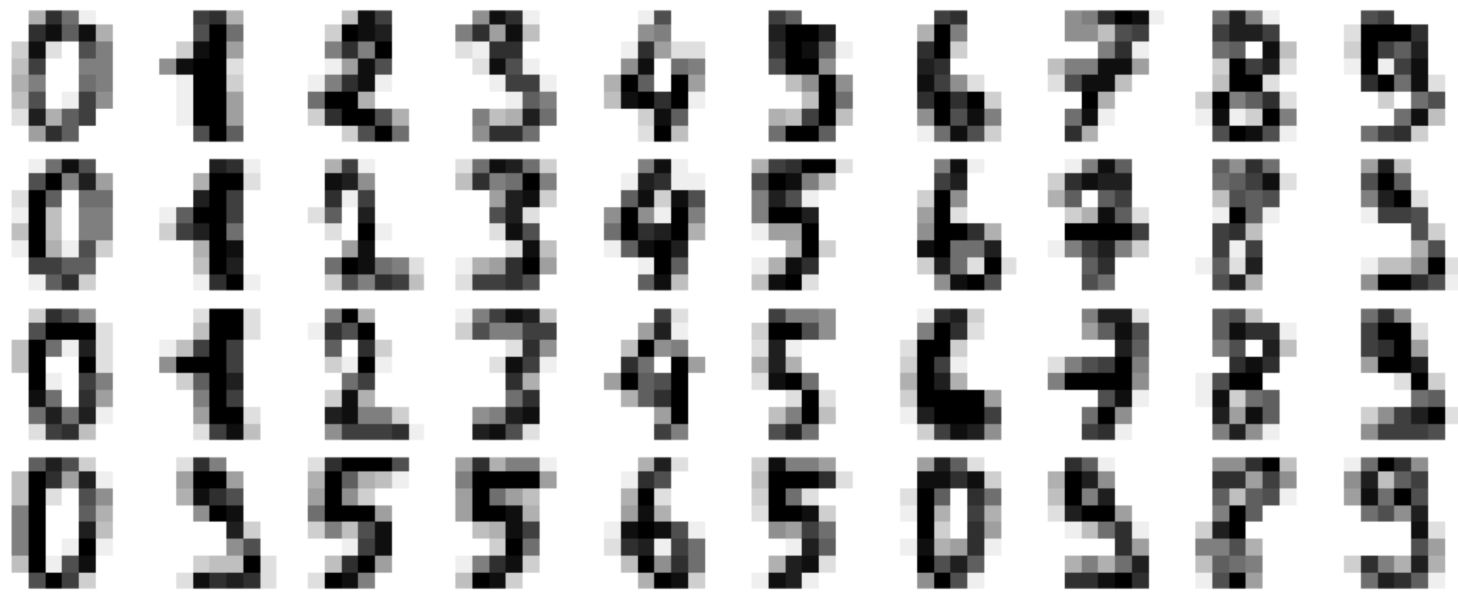

cumulative explained variance
number of components
1.0
0.8
0.6
0.4
0.2
0
10
20
30
40
50
60

PCA in Python

- Can also use this to filter noise:
  - Data will live primarily in the most important components

PCA in Python

- Example:
  - Use some digits from the data set

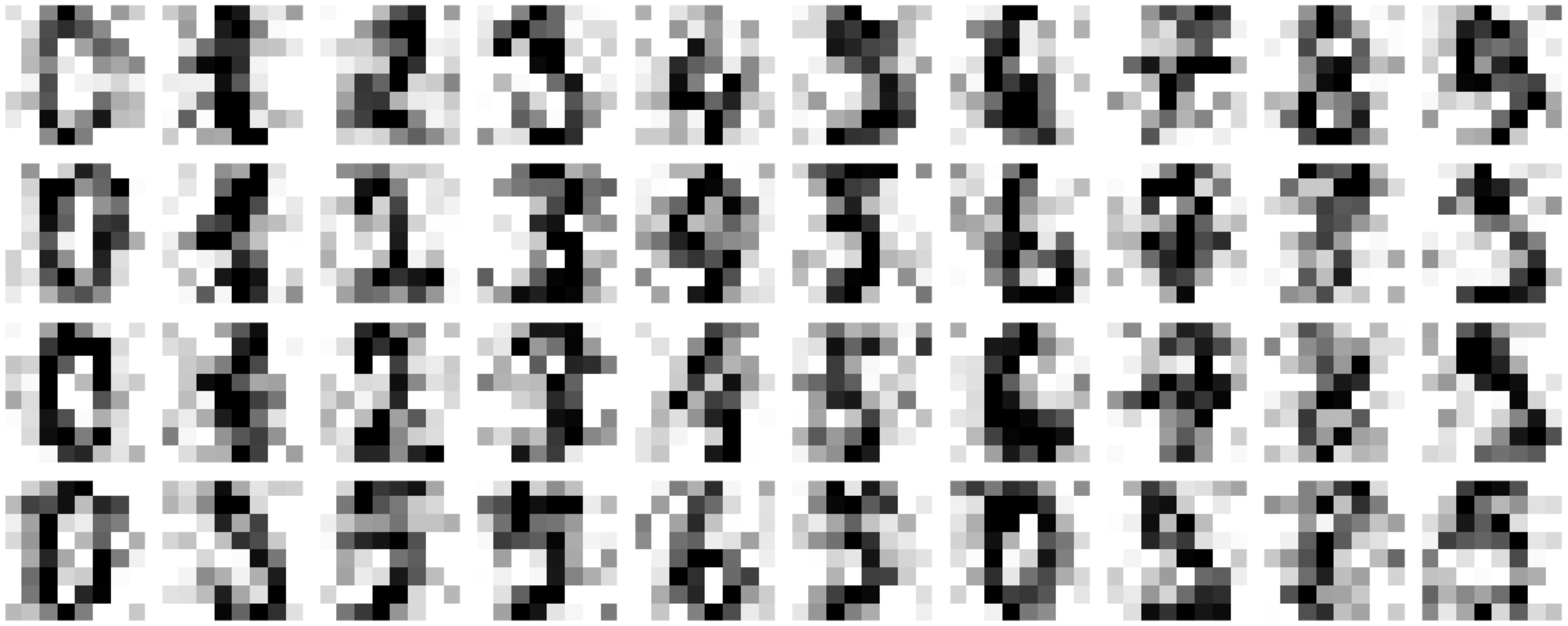


PCA in Python

- Now add some noise

```
np.random.seed(42)
noisy = np.random.normal(digits.data, 4)
plot_digits(noisy)
```

PCA in Python



PCA in Python

- Take the noisy set
  - Use enough components to obtain 50% explained variance

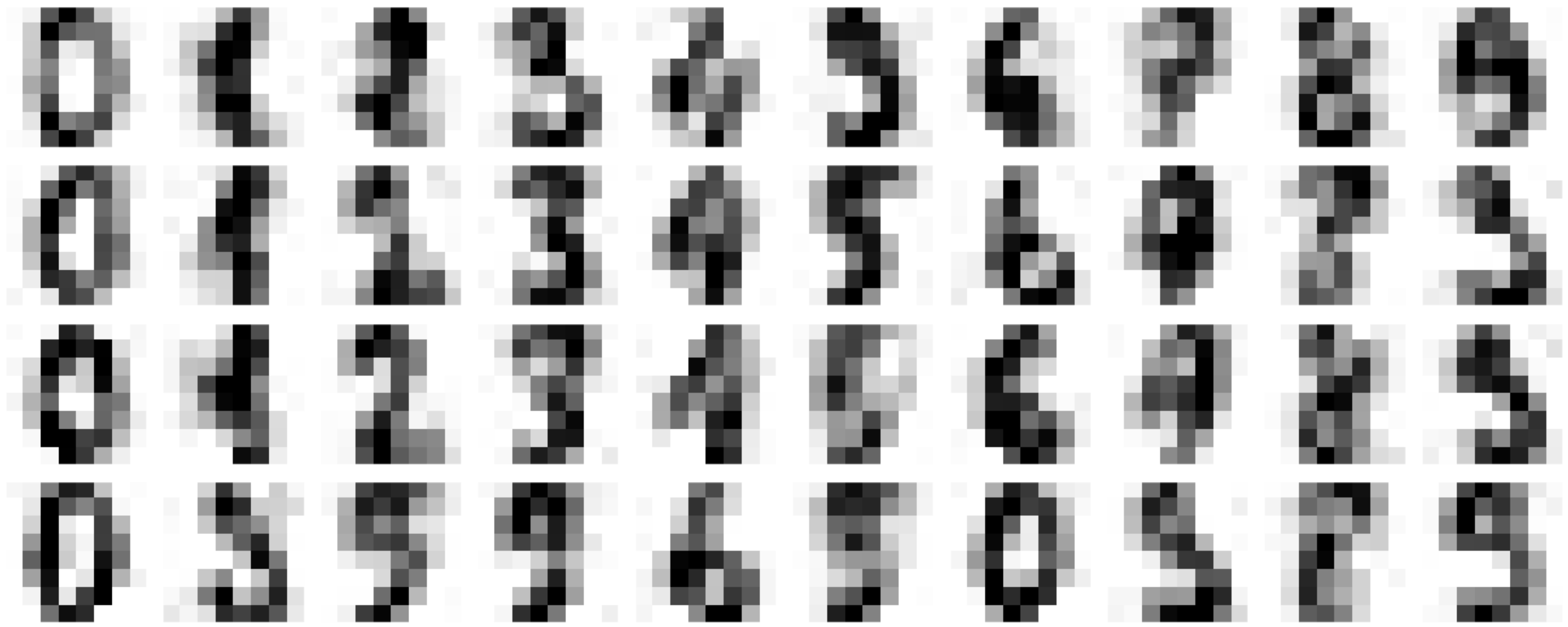
```
pca = PCA(0.50).fit(noisy)
print(pca.n_components_)
```

  - Need 12 components in this case

PCA in Python

- Then display the data of only the highest 12 components

```
components = pca.transform(noisy)
filtered = pca.inverse_transform(components)
plot_digits(filtered)

plt.show()
```



PCA : Eigenfaces

- There is a set of faces of important people in sklearn

```
from sklearn.datasets import fetch_lfw_people
sns.set()


faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)

['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'Geo
W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi
'Tony Blair']
(1348, 62, 47)
```

PCA : Eigenfaces

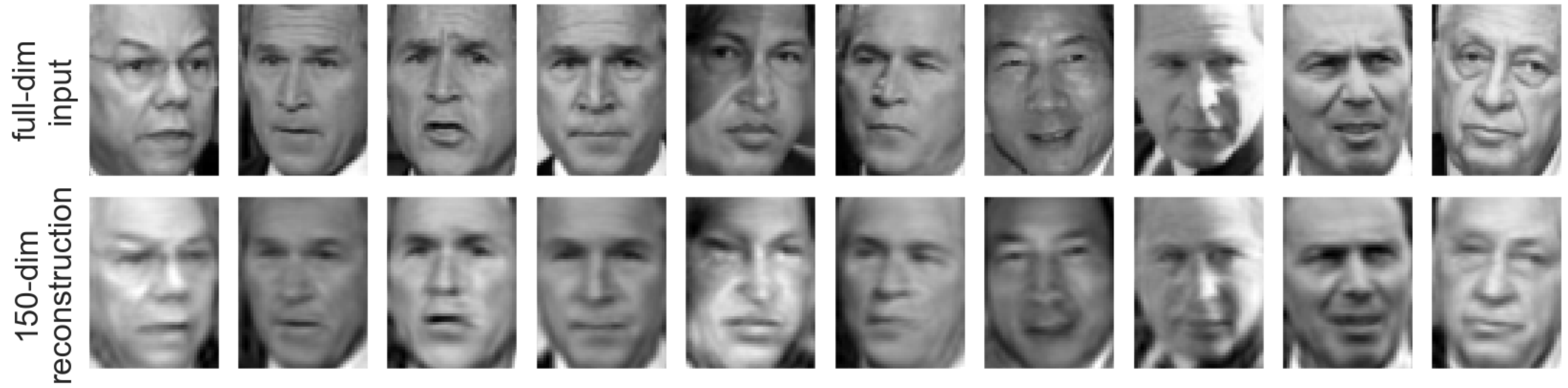
- There is a randomized version of PCA that approximates
  - This is necessary because of the size of the data set

```
pca = PCA(n_components=150,
          svd_solver = 'randomized',
          whiten=True
          )
pca.fit(faces.data)
```

```
pca = PCA(n_components=150, svd_solver = 'randomized',
whiten=True)
pca.fit(faces.data)
components = pca.transform(faces.data)
projected = pca.inverse_transform(components)

fig, ax = plt.subplots(2, 10, figsize=(10, 2.5),
    subplot_kw={'xticks':[], 'yticks':[]},
    gridspec_kw=dict(hspace=0.1, wspace=0.1))
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47),
cmap='binary_r')
    ax[1, i].imshow(projected[i].reshape(62, 47),
cmap='binary_r')

ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('150-dim\nreconstruction');

plt.show()
```

PCA : Eigenfaces

- With about 150 components, the features of the faces are retained



Linear Discriminant Analysis

- Idea:
  - Estimate mean and variance for each category
  - Assumes same covariances
  - Calculates (like PCA) an affine transformation

Linear Discriminant Analysis

- Import LDA:

```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA
```

- Read data & divide

```
iris = pd.read_csv('Iris.csv',
index_col=0).drop(columns='Species')
X_train, X_test, y_train, y_test = train_test_split(
                                   iris,
                                   50*[0]+50*[1]+50*[2],
                                   test_size=0.2,
                                   random_state=0)
```

Linear Discriminant Analysis

- Reset

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- Train with two dimensions:

```
lda = LDA(n_components=2)
lda.fit(X_train, y_train)

for i in range(len(X_test)):
    print(lda.predict([X_test[i]])[0], y_test[i])
```

Linear Discriminant Analysis

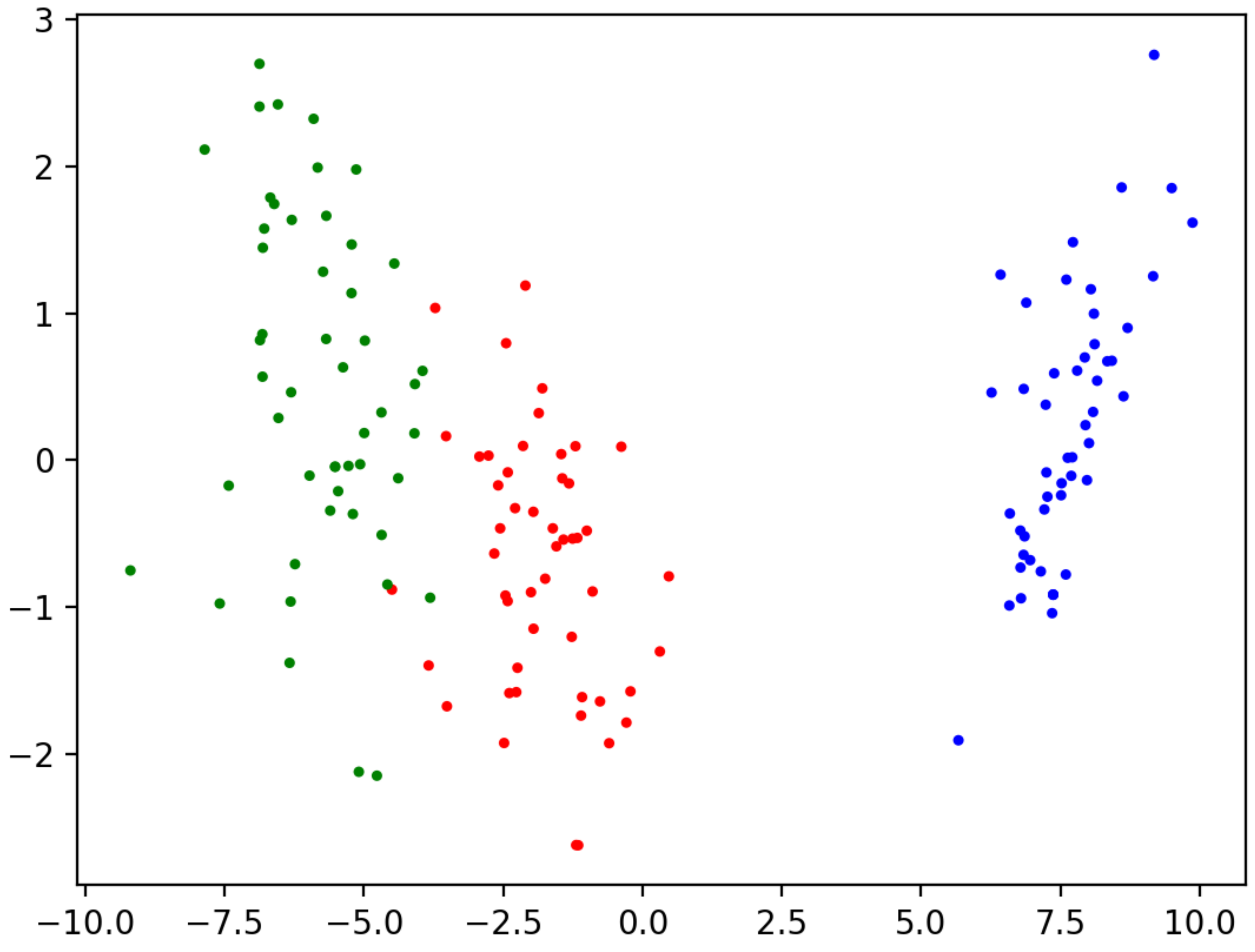
- Results is 100%

Linear Discriminant Analysis

- Show transformation for LDA:

```
transX = lda.fit_transform(iris, 50*[0]+50*[1]+50*[2])

cmap = colors.ListedColormap(['b','r','g'])
plt.scatter(transX[:, 0], transX[:, 1], s=3,
            c=50*[0]+50*[1]+50*[2], cmap = cmap )
plt.show()
```

Linear Discriminant Analysis



3
2
1
0
−1
−2
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0