Machine learning

Dimensionality reduction

Morteza khorsand

Dimensionality reduction

Dimensionality reduction refers to techniques that reduce the number of input variables in a dataset.

curse of dimensionality
The curse of dimensionality basically refers to the difficulties a machine learning algorithm faces when working with data in higher dimensions.

High dimensionality might mean hundreds, thousands, or even millions of input variables.

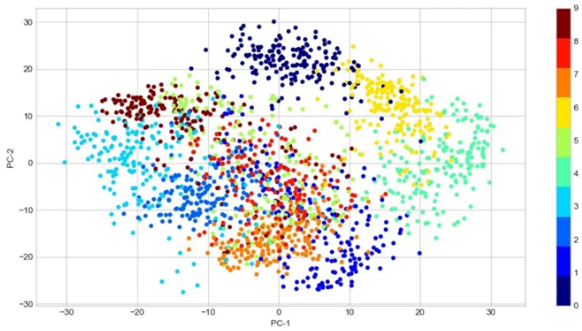
Goal

Visualize Data
Data compression

•Fewer features mean less complexity.
•You will need less storage space because you have fewer data.
•Fewer features require less computation time.
•Model accuracy improves due to less misleading data.

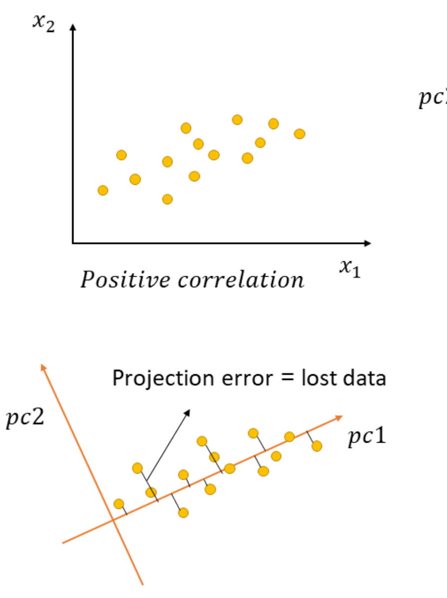
Dimensionality Reduction Techniques

- **Principal component analysis (PCA)**
- Missing value ratio.
- Backward feature elimination.
- Forward feature selection.
- Random forest.
- Factor analysis.
- Independent component analysis (ICA).
- Low variance filter.

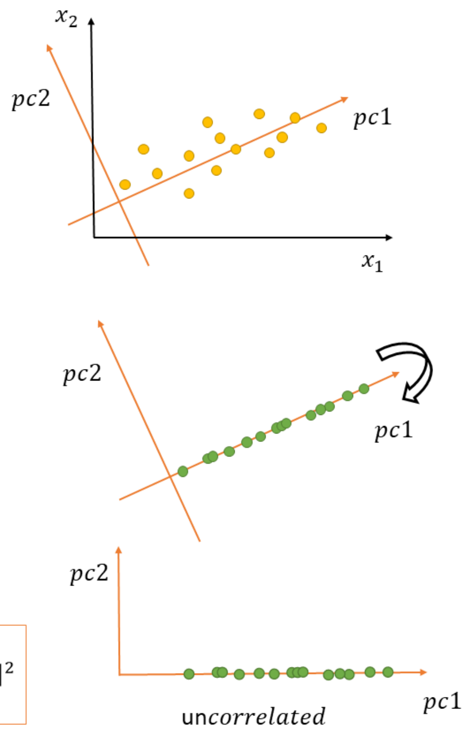


PCA

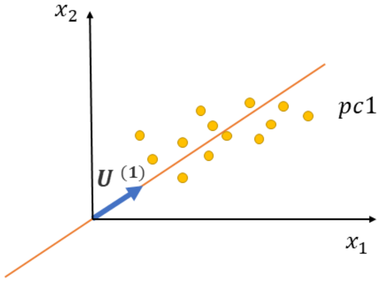
X_1	X_2
1	1.5
1.25	1.6
1.36	1.7
2	1.8
2.01	1.8
2.99	1.9
3	2
3.1	2.01
3.2	2.03
4	2.03
4.1	2.5
4.3	1.87
4.8	2.3
5	2.2
R=4	R=1



minimize $\frac{1}{m}\sum_{i}^{m} \|projection\ error\|^2$

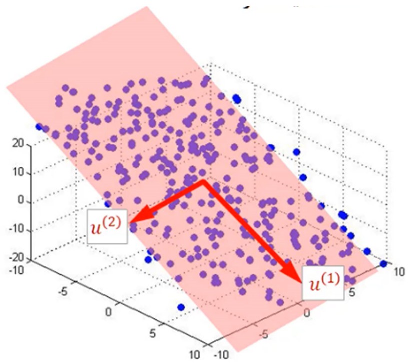


Dimensionality reduction



- **Reduction dimension from 2 to 1**

  Finding a direction like $U^{(1)} \in R^2$ to minimize So that by depicting the points in that direction, the sum of the squares of the error is minimized.



- Dimension reduction from n to k (k<=n))

  finding k orthogonal vectors such as $U^{(1)}, U^{(2)}, U^{(3)} \ldots.\ U^{(k)} \in R^n$ so that by depicting the points in those directions, the sum of squared errors is minimized.

Dimensionality reduction

- **PCA**

  Decorrelation
  - 1. rotate the samples to be aligned with axes
  - 2. shift data samples so they have zero mean

  Reduce dimension

- **Pre processing**

  - Remove average (zero means)
  - subtract the average data of the first column from all the data of the first column and so on.

$$\mu_j = \frac{1}{m} \sum_{i}^{m} x^{(i)}{}_j \qquad x^{(i)}{}_j = x^{(i)}{}_j - \mu_j$$

mu = X.mean (axis = 0)
s_j = X.std(axis=0)
X_norm = (X – mu) /s

  - Scaling (if needed)

$$x^{(i)}{}_j = \frac{x^{(i)}{}_j - \mu_j}{s_j}$$

In [ ]:

In [ ]:

PCA Implementation

In [1]:
```
import sys
import numpy as np
np.set_printoptions(precision=2 )
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
X = np.array([[1 , 1, 1, 0, 0],
              [2,  2, 2, 0, 0],
              [1,  1, 1 ,0 ,0],
              [5 , 5, 5, 0, 0],
              [1  ,1 ,0 ,2 ,2],
              [0  ,0 ,0 ,3 ,3],
              [0  ,0 ,0 ,1 ,1]])
```

In [3]:
```
data= pd.DataFrame(X)
data

data.corr()
```

Out[3]:

	0	1	2	3	4
0	1.000000	1.000000	0.977965	-0.524628	-0.524628
1	1.000000	1.000000	0.977965	-0.524628	-0.524628
2	0.977965	0.977965	1.000000	-0.588069	-0.588069
3	-0.524628	-0.524628	-0.588069	1.000000	1.000000
4	-0.524628	-0.524628	-0.588069	1.000000	1.000000

STEP 1: Zero-center data

preprocessing

In [4]:
```
#preprocessing
mu = X.mean (axis = 0)

print(mu)

print("--------------------------------------------")

X_norm =(X - mu)
print(X_norm )
```

```
[1.43 1.43 1.29 0.86 0.86]
--------------------------------------------
[[-0.43 -0.43 -0.29 -0.86 -0.86]
 [ 0.57  0.57  0.71 -0.86 -0.86]
 [-0.43 -0.43 -0.29 -0.86 -0.86]
 [ 3.57  3.57  3.71 -0.86 -0.86]
 [-0.43 -0.43 -1.29  1.14  1.14]
 [-1.43 -1.43 -1.29  2.14  2.14]
 [-1.43 -1.43 -1.29  0.14  0.14]]
```

Dimension reduction from n to k

1. Covariance matrix

$$\boxed{\sum = \frac{1}{m} X^T X = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} (x^{(i)})^T}$$

2. Covariance matrix decomposition (SVD : singular value decomposition)

$$\boxed{\sum = U \times S \times V \qquad [U \times S \times V] = \text{svd}\,(\sum)}$$

$$U = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ U^{(1)} & U^{(2)} & U^{(3)} & \vdots & U^{(n)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{n \times n} \qquad S = \begin{bmatrix} s_{11} & 0 & 0 & \cdot & 0 \\ 0 & s_{22} & 0 & & 0 \\ \cdot & \cdot & s_{33} & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & 0 & \cdot & s_{nn} \end{bmatrix} \qquad s_{11} > s_{22} > s_{33} \ldots s_{nn}$$

3. Select first k matrices from U matrix

$$U_{reduced} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ U^{(1)} & U^{(2)} & U^{(3)} & \vdots & U^{(k)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}_{n \times k}$$

STEP 2: Compute covariance matrix

In [5]:
```
m=X.shape[0]
sigma= (1/m)* X.T@X
print(sigma)
```

```
[[4.57 4.57 4.43 0.29 0.29]
 [4.57 4.57 4.43 0.29 0.29]
 [4.43 4.43 4.43 0.   0.  ]
 [0.29 0.29 0.   2.   2.  ]
 [0.29 0.29 0.   2.   2.  ]]
```

STEP 3: Singular Value Decomposition

In [6]:
```
#decomposition (svd)
U, S, _ = np.linalg.svd(sigma)

print(U)

print("---------------------------")

print(S)
```

```
[[-5.81e-01 -4.61e-03  4.03e-01 -7.07e-01  1.08e-16]
 [-5.81e-01 -4.61e-03  4.03e-01  7.07e-01 -9.53e-17]
 [-5.67e-01  9.62e-02 -8.18e-01 -9.22e-15 -1.31e-17]
 [-3.50e-02 -7.04e-01 -5.85e-02 -6.14e-16 -7.07e-01]
 [-3.50e-02 -7.04e-01 -5.85e-02 -7.25e-16  7.07e-01]]
---------------------------
[1.35e+01 4.00e+00 6.69e-02 5.87e-16 6.06e-48]
```

In [7]:
```python
U_reduced = U[ : , :2 ]

print(U_reduced)
```

```
[[-0.58 -0.  ]
 [-0.58 -0.  ]
 [-0.57  0.1 ]
 [-0.03 -0.7 ]
 [-0.03 -0.7 ]]
```

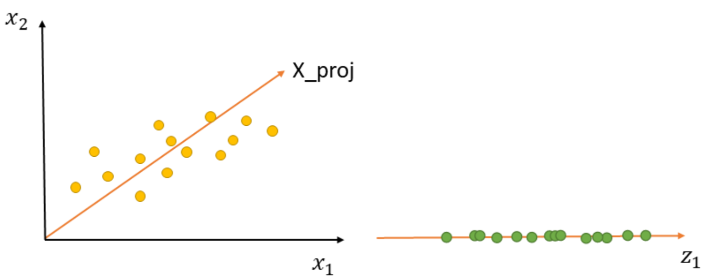
In [8]:
```python
X_proj = X_norm@ U_reduced

print(X_proj)
```

```
[[ 0.72  1.18]
 [-1.01  1.27]
 [ 0.72  1.18]
 [-6.2   1.53]
 [ 1.15 -1.73]
 [ 2.24 -3.13]
 [ 2.38 -0.31]]
```

Dimensionality reduction 8

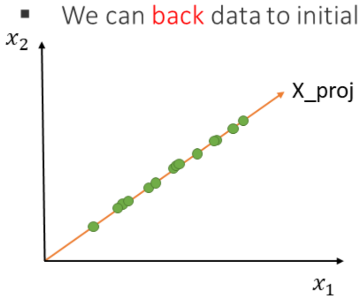
4. Project data on new vectors

$$[U_{reduced}]^T{}_{k\times n} \times x^{(i)}{}_{n\times 1} = [\text{X\_proj}]_{k\times 1}$$



Initial data (two dimensional)   New data (one dimensional)

- We can back data to initial space to see how much data is lost



$$X_{recoverd} = \text{X\_proj} \times u^T{}_{reduced} + means$$

STEP 4: Recover

In [9]:
```python
X_approx = X_proj @ U_reduced.T + mu

print(X_approx)
```

```
[[ 1.00e+00  1.00e+00  9.91e-01 -6.57e-04 -6.57e-04]
 [ 2.01e+00  2.01e+00  1.98e+00 -1.38e-03 -1.38e-03]
 [ 1.00e+00  1.00e+00  9.91e-01 -6.57e-04 -6.57e-04]
 [ 5.02e+00  5.02e+00  4.95e+00 -3.55e-03 -3.55e-03]
 [ 7.69e-01  7.69e-01  4.68e-01  2.03e+00  2.03e+00]
 [ 1.41e-01  1.41e-01 -2.86e-01  2.98e+00  2.98e+00]
 [ 4.67e-02  4.67e-02 -9.48e-02  9.93e-01  9.93e-01]]
```


- **Select "k"**

1. The mean of the sum of squared projection errors.

$$\boxed{\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x^{(i)}{}_{approx}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\le 0.01}$$

K=0
Repeat
{
K=k+1
Try Pca(X) with K components

Compute $U_{reduced}\ Z^{(1)}, Z^{(2)}, Z^{(3)}, \dots Z^{(n)}\ , x_{aprox}{}^{(1)}, x_{aprox}{}^{(2)}, \dots x_{aprox}{}^{(m)}$

Until $$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x^{(i)}{}_{approx}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\le 0.01$$
}

Choosing number of principal components

In [10]:
```
m, n = X.shape
X= X - X.mean(axis=0)
sigma= (X.T @X) / m

U , S , V = np.linalg.svd(sigma)

for k in range(1 , n+1):
    total_var = np.sum(S[ : k]) / np.sum(S)
    if total_var >= 0.99:
        print(k)
        break
```

2

In [ ]:

In [ ]:

PCA class

In [11]:

```
class Pca:
    def __init__ (self , X , variance):
        self.X = X
        self.variance = variance

    def covar(self):
        m,n = self.X.shape
        mu=self.X .min(axis=0)
        X= (self.X-mu)

        sigma = self.X.T @ self.X
        u, s, _ = np.linalg.svd(sigma)

        last_k=0
        for k in range(n+1):
            total_var = np.sum(s[ : k]) / np.sum(s)
            if total_var >= self.variance:
                break

        last_k=k
        u_reduced = u[: , : last_k]
        x_proj = self.X@u_reduced

        X_recoverd= x_proj @ u_reduced.T + mu

        return  sigma , u , s , u_reduced  , x_proj  , X_recoverd
```

Example

In [13]:

```
import pandas as pd
```

In [14]:

```
import pandas as pd
seed=pd.read_csv("seeds-width-vs-length.csv")
data=np.array(seed)
d = {'width': data[: , 0], "length" : data[: , 1]}
df=pd.DataFrame(d)
df
```

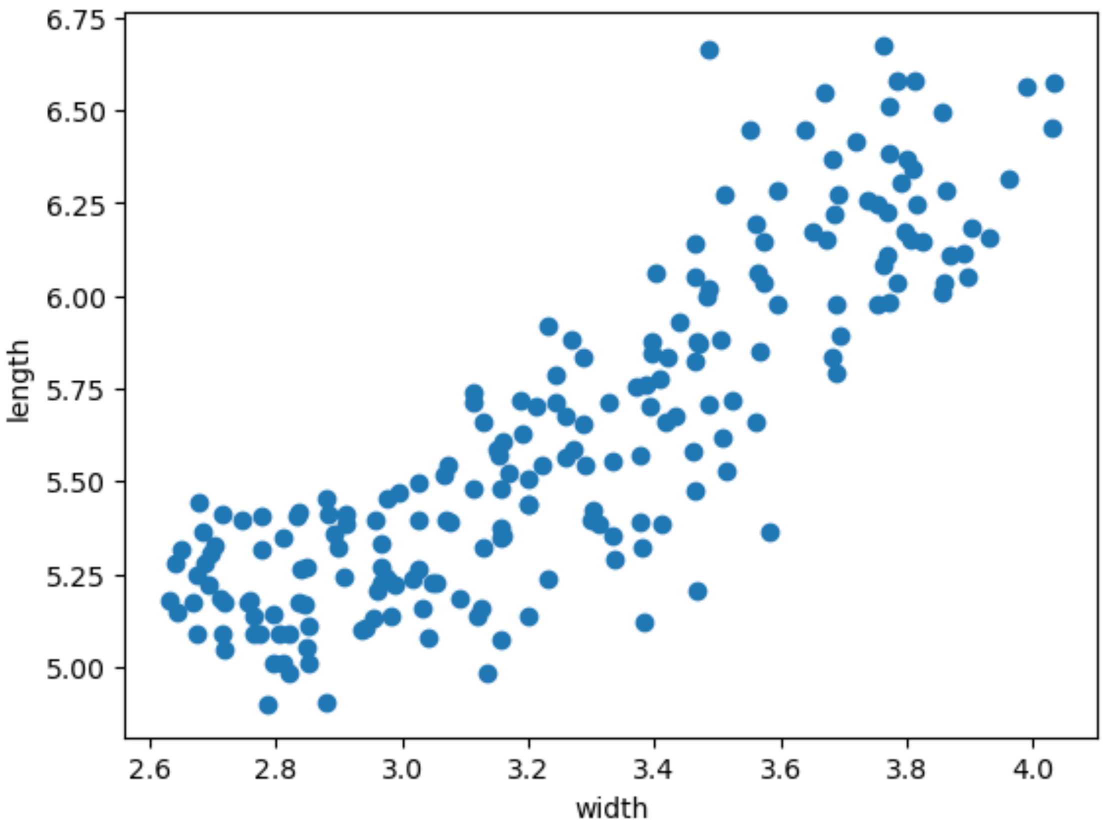
Out[14]:

	width	length
0	3.333	5.554
1	3.337	5.291
2	3.379	5.324
3	3.562	5.658
4	3.312	5.386
...	...	...
204	2.981	5.137
205	2.795	5.140
206	3.232	5.236
207	2.836	5.175
208	2.974	5.243

209 rows × 2 columns

In [15]:

```
plt.plot(df["width"] , df["length"] , "o") ;
plt.xlabel("width")
plt.ylabel("length")
plt.show()
```



In [16]:
```
A= Pca(data , 0.99)

sigma=A.covar()[0]
u=A.covar()[1]
s=A.covar()[2]
u_reduced=A.covar()[3]
x_proj=A.covar()[4]
X_recoverd=A.covar()[5]

print(sigma)
print("--------------")
print(u)
print("--------------")
print(s)
print("--------------")
print(u_reduced)
print("--------------")
```

```
[[2248.73 3862.65]
 [3862.65 6660.7 ]]
--------------
[[-0.5  -0.86]
 [-0.86  0.5 ]]
--------------
[8.90e+03 6.52e+00]
--------------
[[-0.5 ]
 [-0.86]]
--------------
```

Sklearn

In [17]:
```
from sklearn.decomposition import PCA
```

In [18]:
```
pca = PCA()
pca.fit(data)
transformed=pca.transform(data)
```

In [19]:
```
U = pca.components_          # Principal Components (directions)
S = pca.explained_variance_  # importance of ecah direction (variances)

print("1st Principal Component: {} ({:.2f})".format(U[0], S[0]))
print("2nd Principal Component: {} ({:.2f})".format(U[1], S[1]))
```

```
1st Principal Component: [0.64 0.77] (0.32)
2nd Principal Component: [-0.77  0.64] (0.02)
```

In [20]:
```
print(np.linalg.norm(U[0]))
print(np.linalg.norm(U[1]))
```

```
1.0
1.0
```

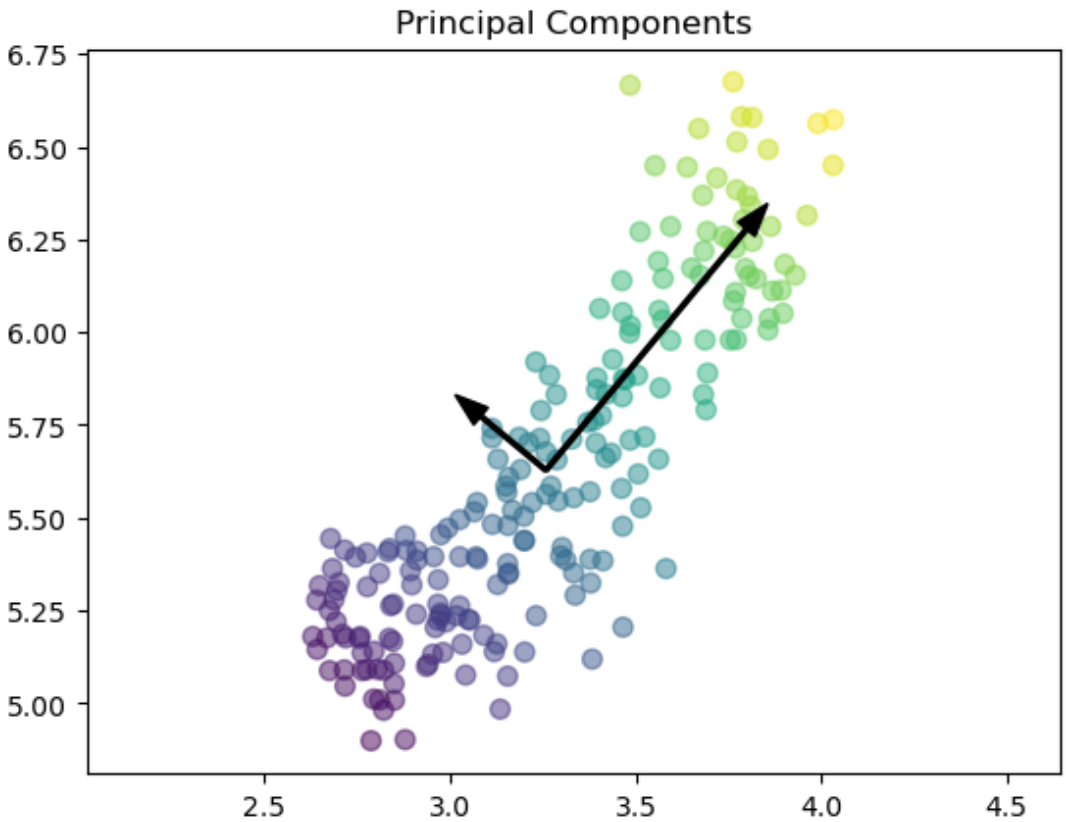
In [21]:
```
print(np.dot(U[0], U[1]))
```

```
0.0
```

In [22]:
```
mean=pca.mean_
mean
```

Out[22]:
```
array([3.26, 5.63])
```

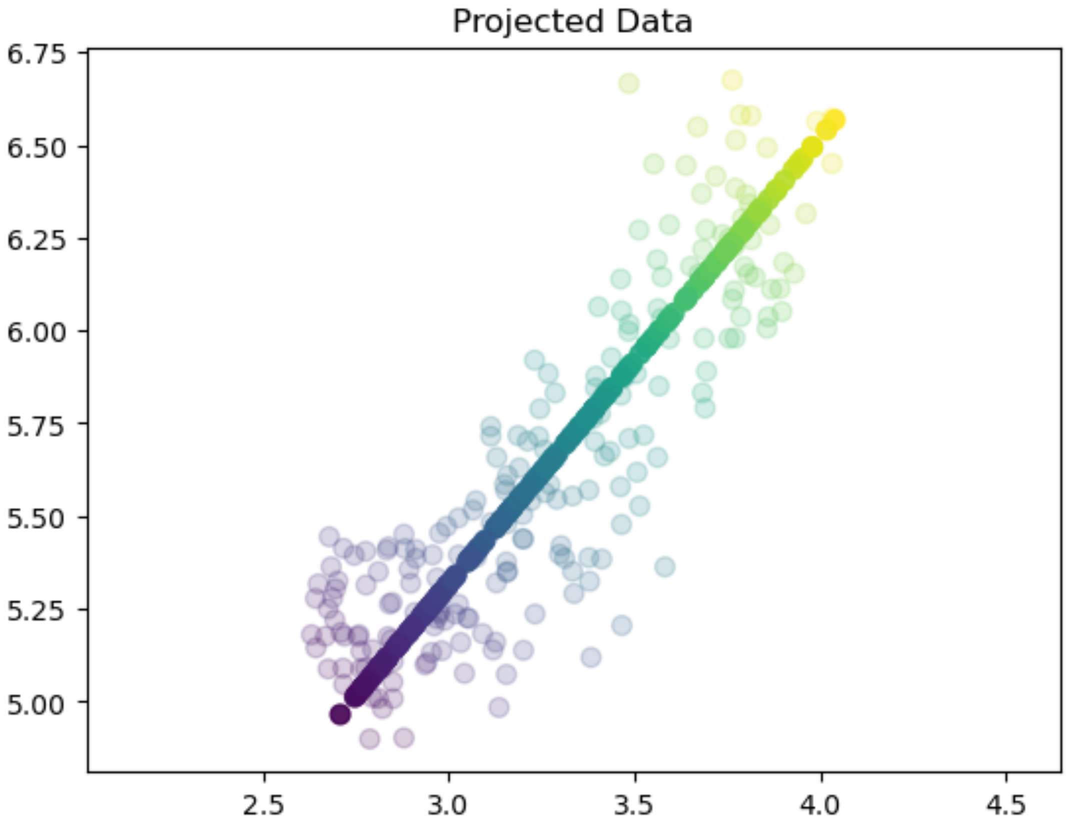
In [23]:
```
# plot data
c = transformed[ :, 0]          # colors
plt.scatter(data[:, 0], data[:, 1], s=50, c=c, cmap='viridis', alpha=0.5)

plt.arrow(mean[0] , mean[1],  1.5 * np.sqrt(S[0]) * U[0, 0], 1.5 * np.sqrt(S[0]) * U[0,
plt.arrow( mean[0] , mean[1], 1.5 * np.sqrt(S[1]) * U[1, 0], 1.5 * np.sqrt(S[1]) * U[1,

plt.title("Principal Components")
plt.axis('equal')
plt.show()
```



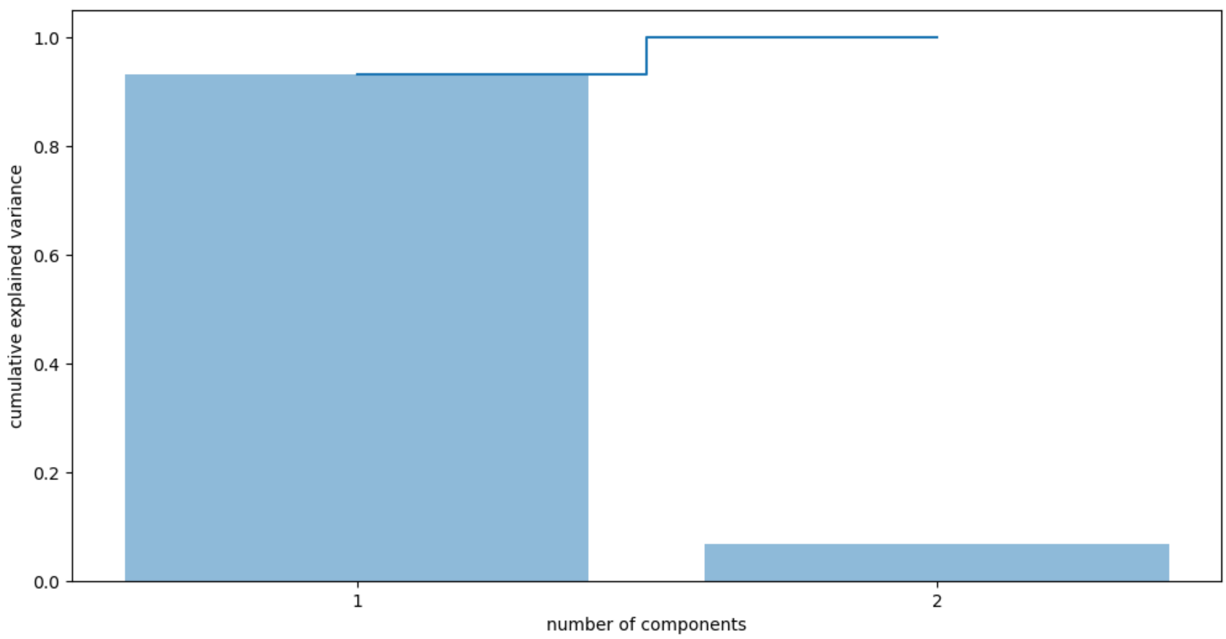
In [24]:
```
pca = PCA(0.93)                 # keep 93% of variance
X_proj = pca.fit_transform(data)

print(data.shape)
print(X_proj.shape)
```

```
(209, 2)
(209, 1)
```

In [25]:
```
X_approx = pca.inverse_transform(X_proj)

plt.scatter(data[:, 0], data[:, 1], s=50, c=c, cmap='viridis', alpha=0.2)
plt.scatter(X_approx[:, 0], X_approx[:, 1], s=50, c=c, cmap='viridis', alpha=0.9)  # plo
plt.title("Projected Data")
plt.axis('equal')
plt.show()
```



In [26]:
```
plt.figure(figsize=(12, 6))

pca = PCA().fit(data)  # Notice

plt.bar(range(len(pca.explained_variance_ratio_)),
        pca.explained_variance_ratio_,
        alpha=0.5,
        align='center')

plt.step(range(len(pca.explained_variance_ratio_)),
         np.cumsum(pca.explained_variance_ratio_),
         where='mid')
plt.xticks([0, 1], [1 ,2])
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```

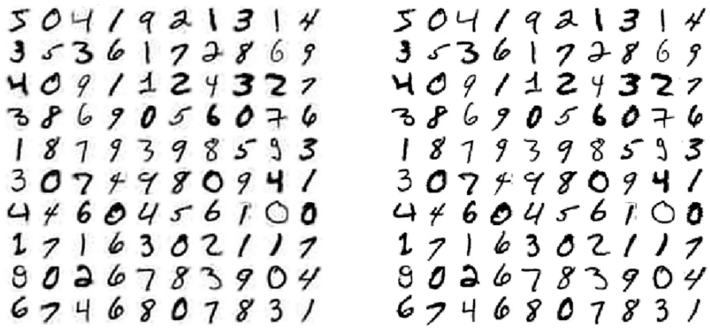


In [27]:
```python
total_var = np.cumsum(pca.explained_variance_ratio_)

for i in [0, 1]:
    print("Components: {:2d}, total explained variance: {:.2f}".format(i, total_var[i]))
```

```
Components:  0, total explained variance: 0.93
Components:  1, total explained variance: 1.00
```

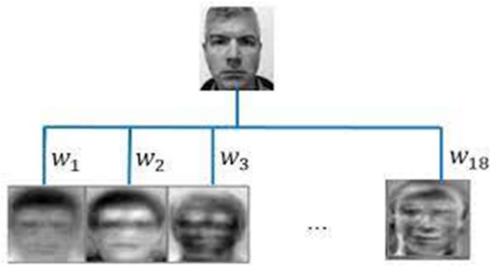
Dimensionality reduction

- **mnist dataset compression**



K = 100 Variance = 0.91 n= 4096 Original data

- **Image compression**



Dimensionality reduction

Original Faces

n= 4096

K = 50 Variance: 0.87

K = 100 Variance: 0.93

K = 150 Variance: 0.96

K = 200 Variance: 0.98