Lecture 13 Dimension Reduction: Principal Component Analysis

Starting from this lecture, we are goting to talk about **unsupervised machine learning**. The fundamental difference with supervised learning is that in unsupervised learning problems, there is no label (response) y to be predicted. All we have is the data matrix $X \in \mathbb{R}^{n\times p}$, and the general task is to explore the "pattern" of data.

Important Remark: For simplicity, below we assume that all variables (features) of X has mean zero. For arbitrary X, we can pre-process it by substracting the mean of each variable for the corresponding column.

One classical type of problem in unsupervised learning is **Dimension Reduction** (another type of problem is clustering).

(unrigorous) Mathematical Description: Given high-dimensional data observation $\mathbf{x} \in \mathbb{R}^{1\times p}$ (imagine it as one sample), find a "reasonable" projection function

$$\mathbf{t} = \mathbf{h}(\mathbf{x}) : \mathbb{R}^{1\times p} \to \mathbb{R}^{1\times k}, k << p$$

that "preserves" the high-dimensional information.

- A naive solution is to randomly pick k component of $\mathbf{x}$-- of course there is huge waste of information.
- Another simple yet reasonable assumption is that $\mathbf{h}$ is linear transformation -- of course, the linear coefficients should depend on the "structure" of dataset. In other words, the "new coordinates" are the linear combination of "old coordinates".

  $$\mathbf{t} = \mathbf{h}(\mathbf{x}) = \mathbf{x}\mathbf{V}_k, \mathbf{V}_k \in \mathbb{R}^{p\times k}.$$

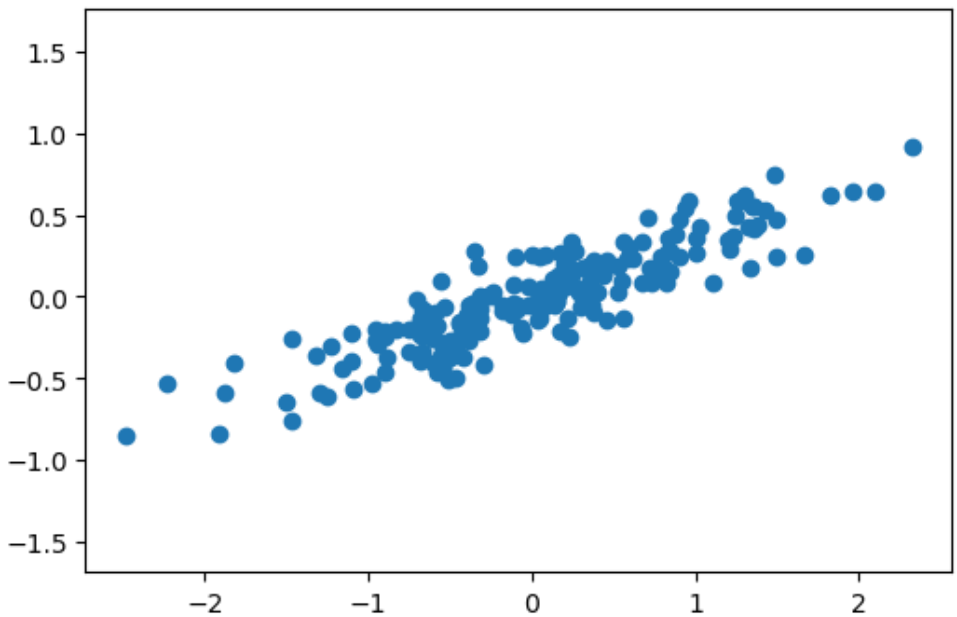
  In data matrix form (n samples), we have

  $$\mathbf{T}_k = \mathbf{X}\mathbf{V}_k \in \mathbb{R}^{n\times k}$$

Principal Component Analysis (PCA) is one typical linear dimension reduction method. Write $\mathbf{V}_k = [\mathbf{v}_1\, \mathbf{v}_2\, \cdots\, \mathbf{v}_k]$, then the column vectors $\mathbf{v}_j (1 \le j \le k)$ are called the first k **Principal Components (PCs)** of the dataset, and T_k is called the score matrix -- each row represents the k scores of one sample in k PCs -- they are the representation of the sample in R^k space.

Now the central question becomes: how to find the PCs based on the dataset?

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
rng = np.random.RandomState(1)
X = np.matmul(rng.rand(2, 2), rng.randn(2, 200)).T
fig = plt.figure(dpi=100)
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');
```

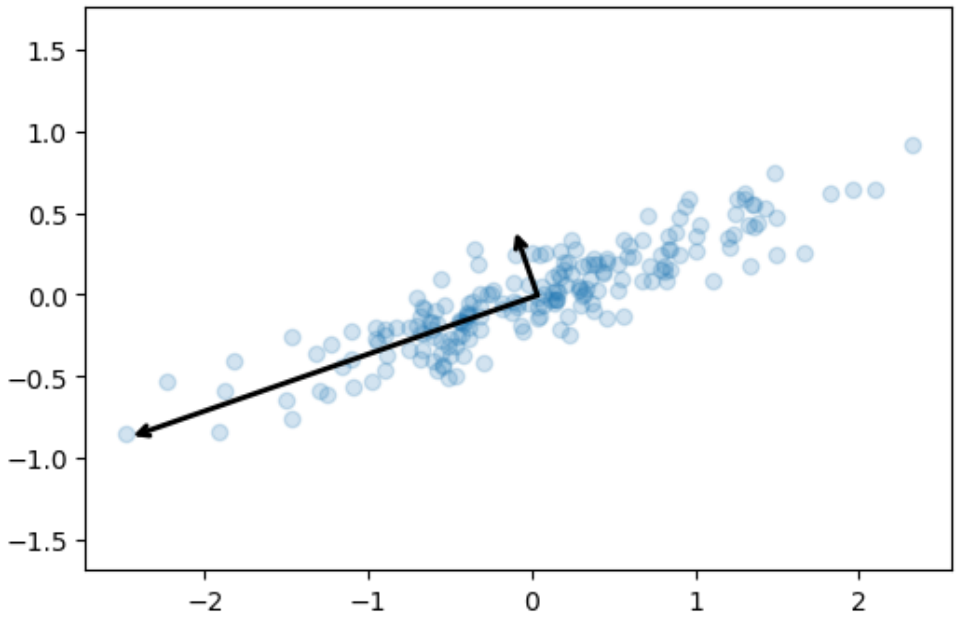


The data is in 2D space. If I "force" you to believe that the data can be further reduced to one-dimension , you may (reluctantly) admit that data is generated along one line (the long axis of "ellipse" sketched by the data), and the other short axis merely corresponds "noise".

In [2]:
```
from sklearn.decomposition import PCA
# run pca from sklearn
pca = PCA(n_components=2)
pca.fit(X)

def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
fig = plt.figure(dpi=100)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



Therefore we may imagine the major "axis" of ellipse are the principal components of the data, and the reduced coordinate are the projections on such direction.

How do we determine the direction of "ellipse axis"? A staightfoward way is from **covariance matrix** of the data (to fully understand this, you need some knowledge about quadratic forms in linear algebra and multivariate Gaussian distribution in probability, although it's optional for the basic requirements).

PCA from Covariance Matrix (https://en.wikipedia.org/wiki/Covariance)

- **Step 0**: Center the data matrix, making it column mean zero.
- **Step 1**: Calculate the covariance matrix
$$\Sigma = \frac{1}{n-1} X^\top X \in \mathbb{R}^{p\times p}.$$
The element Σ_{ij} denotes the correlation between variable (feature) i and j in the data.
- **Step 2**: Eigen-decomposition of symmetrix covariance matrix Σ,
$$\Sigma = V\Lambda V^\top,$$
where $V \in \mathbb{R}^{p\times p}$ is orthogonal matrix whose columns are unit eigen-vectors and Λ is the diagonal matrix of eigen-values. We further arrange the λ_j in descending orders.
- **Step 3**: Principal Components are just the first k columns of V, denoting as V_k. Indeed, they are the eigen-vectors corresponding to the top k eigen-values.
- **Step 4**: Compute the score matrix
$$T_k = XV_k.$$
Then each row of T_k is the coordinate of the sample in $\mathbb{R}^k$ space.

(Optional) Remark: The covariance matrix of T_k is $\frac{1}{n-1}T_k^\top T_k^\top = V_k^\top \Sigma V_k = \Lambda_k$, which is defined as the (1:k,1:k) submatrix of Λ. This means that different PCs are independent from each other, and λ_j is indeed the variance of j-th PC.

In [ ]:
```
import numpy as np

class myPCA():
    '"write your document strings here"'

    def __init__(self, n_components = 2):
        '"write your document strings here"'
        self.n_c = n_components


    def fit(self,X):
        '"write your document strings here"'
        cov_mat = np.cov(X.T) # covariance matrix, the input matrix to this function
 does not need to be centered
        eig_val, eig_vec = np.linalg.eigh(cov_mat) #eigen-values and orthogonal eigen
-vectors --ascending order
        eig_val = np.flip(eig_val) # reverse the order --descending
        eig_vec = np.flip(eig_vec,axis=1) # reverse the order
        self.eig_values = eig_val[:self.n_c] # select the top eigen-vals
        self.principle_components = eig_vec[:,:self.n_c] # select the top eigen-vecs
        self.variance_ratio = self.eig_values/eig_val.sum() # variance explained by e
ach PC

    def transform(self,X):
        '"write your document strings here"'
        return np.matmul(X-X.mean(axis = 0),self.principle_components) #project the d
ata (centered) on PCs
```

In [ ]:
```
from sklearn.datasets import load_digits
X,y = load_digits(return_X_y = True)
```

In [ ]:
```
pca = myPCA(n_components = 5)
pca.fit(X)
X_pca = pca.transform(X)
```

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
figure = plt.figure(dpi=100)
plt.scatter(X_pca[:, 0], X_pca[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,cmap=plt.
cm.get_cmap('tab10', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```

In [ ]:
```
from numpy import random
figure = plt.figure(dpi=100)
rand_ind = random.randint(64, size=(2)) # pick up two random pixels from original dat
a
plt.scatter(X[:, rand_ind[0]], X[:, rand_ind[1]],c=y, s=15, edgecolor='none', alpha=
0.5,cmap=plt.cm.get_cmap('tab10', 10))
plt.xlabel('pixel 1')
plt.ylabel('pixel 2')
plt.colorbar();
```

In [ ]:
```
from sklearn.decomposition import PCA
pca_sklearn = PCA(n_components=2)
projected = pca_sklearn.fit_transform(X)

figure = plt.figure(dpi=100)
plt.scatter(projected[:, 0], projected[:, 1],c=y, s=15, edgecolor='none', alpha=0.5,c
map=plt.cm.get_cmap('tab10', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```

In [ ]:
```
pca.principle_components[:,1] # principle components (directions), in our code
print(pca_sklearn.components_[1,:]) # in skelarn, note the shape are different
```

In [ ]:
```
print(pca.variance_ratio) # ratio of variance explained by first few PCs
print(pca_sklearn.explained_variance_ratio_)
```

Another Algorithm: Singular Value Decomposition (SVD)

The PCA in [scikit-learn (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) is realized with SVD of data matrix, which sometimes can be more stable numerically than covariance matrix-based approach.

Let $X \in \mathbb{R}^{n\times p}$ be the centered data matrix (each feature is of mean zero).

> Singular Value Decomposition (SVD): any real matrix can be decomposed into the following form:
> $$X = USV^\top$$

Where $S \in \mathbb{R}^{n\times p}$ is a diagonal matrix whose diagonal entries are non-negative and in decreasing order. $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{p\times p}$ are orthogonal matrices (i.e. columns of V are orthogonal, same for U, $U^\top U = UU^\top = I$).

The columns of $V = [\mathbf{v}_1\, \mathbf{v}_2\, \cdots\, \mathbf{v}_p]$ are are known as the right singular vectors. They are indeed the **principal components** (important directions).

The columns of $U = [\mathbf{u}_1\, \mathbf{u}_2\, \cdots\, \mathbf{u}_n]$ are known as the left singular vectors.

Relation with covariance-based approach: Since X is centered, we have covariance matrix

$$\Sigma = \frac{1}{n-1}X^\top X = \frac{1}{n-1}VS^\top SV^\top,$$

Note that $S^\top S$ is the diagonal matrix, therefore V is the eigen-vector matrix of Σ -- i.e. principle components. The first k score (projection) matrix $T_k \in \mathbb{R}^{N\times k}$ of the data on first k-PCs are then calculated as $T_k = XV_k = U_kS_k$.