Computer Science

# CSC380: Principles of Data Science

## Dimensionality Reduction

Kyoungseok Jang

## Announcement

- Reading quiz next Tuesday (April 25th)
  - I will specify the questions and reading materials for the reading quiz and announce them to you.

- HW6 grade posted.

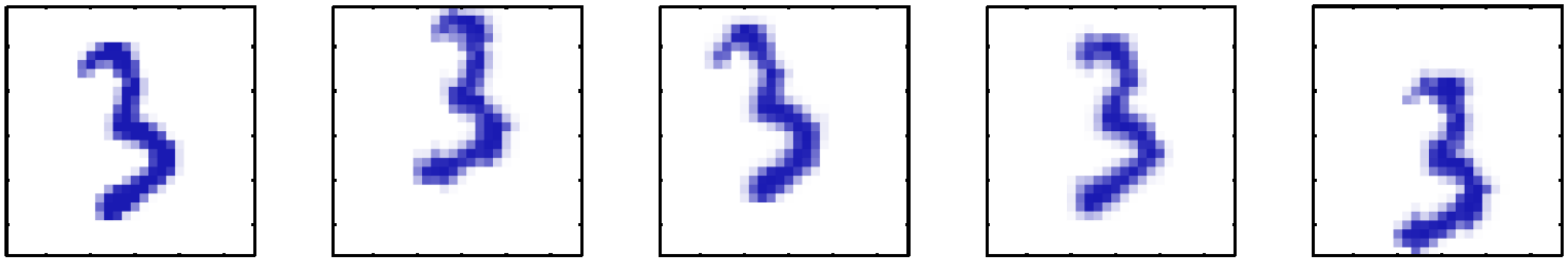
## Announcements

- Today will be the end of the scheduled lecture material!
  - Apr 25th: interesting machine learning algorithms or more detailed explanations on math stuff like Bayesian, MAP, … (topics that will not be included in your grading)
  - Apr 27th: wrap-up 1
  - May 2nd: wrap-up 2

- Maybe it's because…
  - I spoke too fastly.
  - Removed too much math that requires calculus/linear algebra knowledge.

## Motivation

*Data often have a lot of redundant information…*



**Example** A dataset consisting of a hand-drawn 3 at random locations and rotations in a 100x100 pixel image.

**Data Dimension** 100 x 100 = 10,000

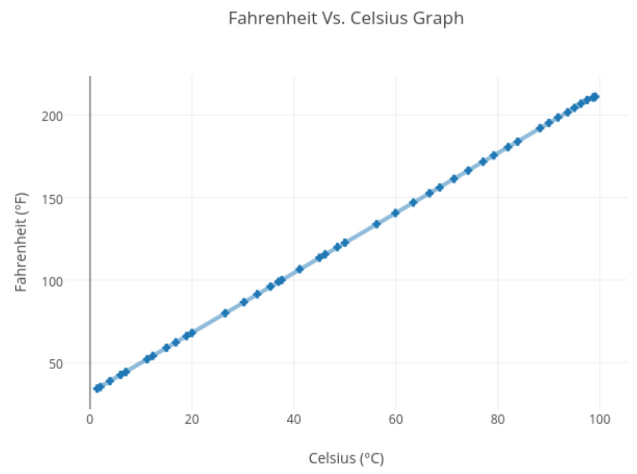
**Intrinsic Dimension** 3 (X-position, Y-position, Rotation)

[ Source: Bishop, C. ]

## Motivation

*…or data have strongly dependent features…*

| Fahrenheit | Celsius |
|---|---|
| 3.1 | -16.1 |
| 100.5 | 38.1 |
| 27.3 | -2.6 |
| 18.1 | -7.7 |
| 18.9 | -7.3 |
| 21.7 | -5.7 |
| … | … |

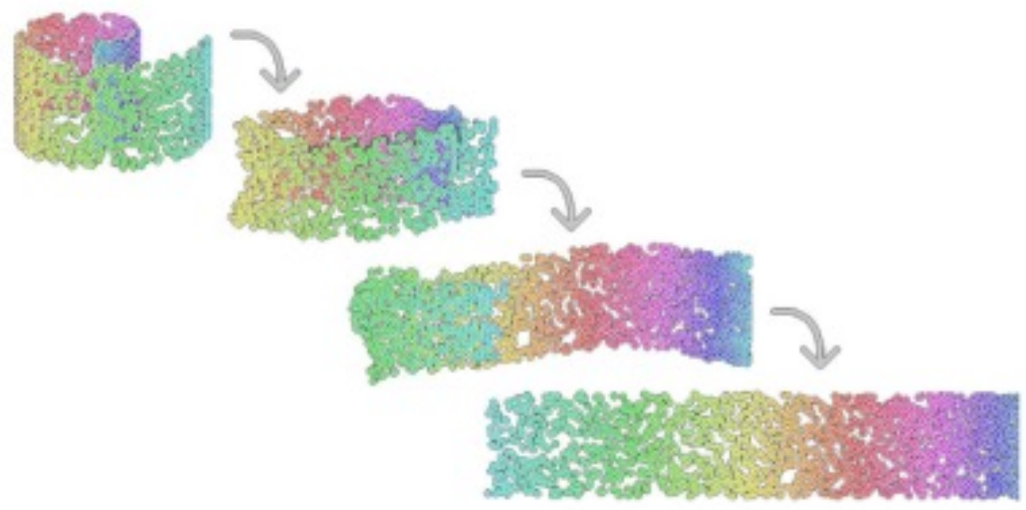


Linear Function

$$F = 1.8C + 32$$

## Motivation

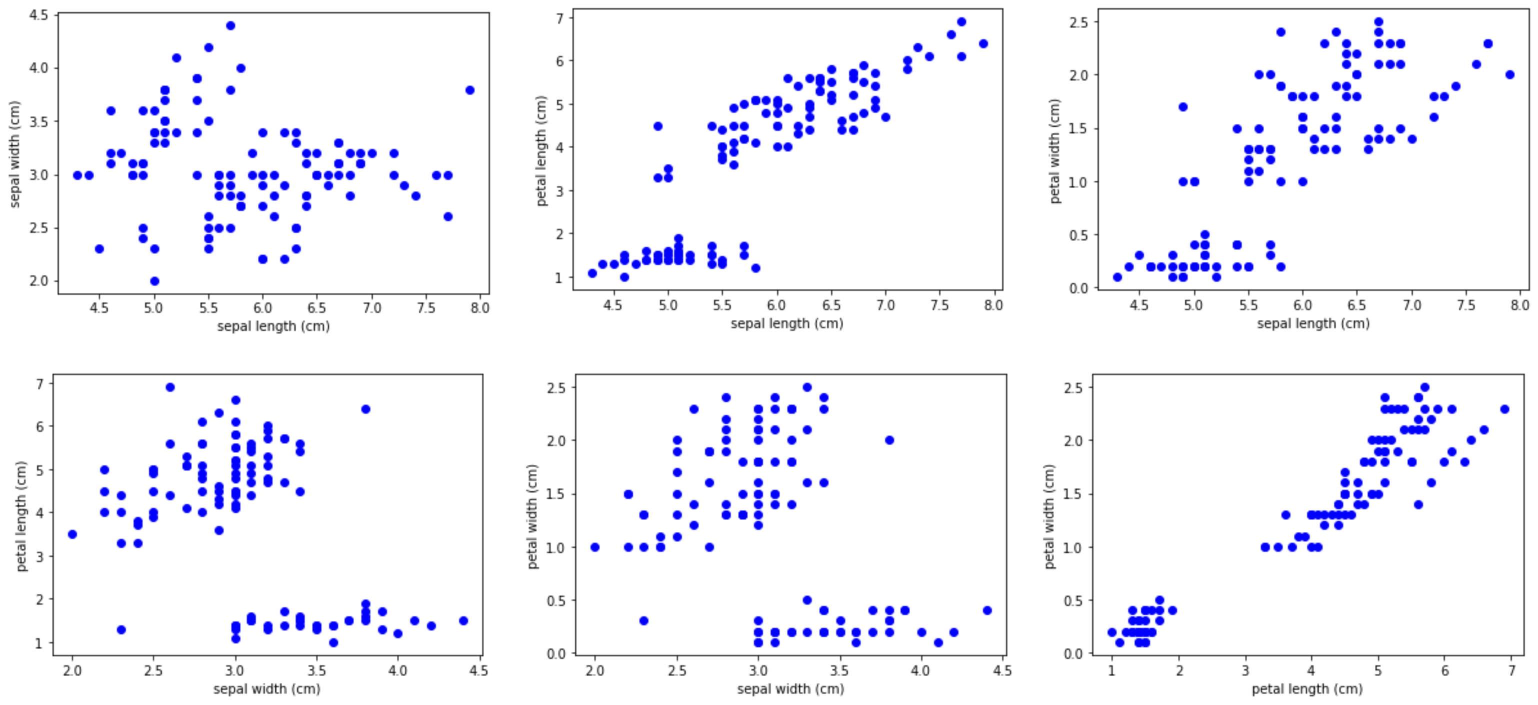
…or data are high-dimensional and hard to visualize…



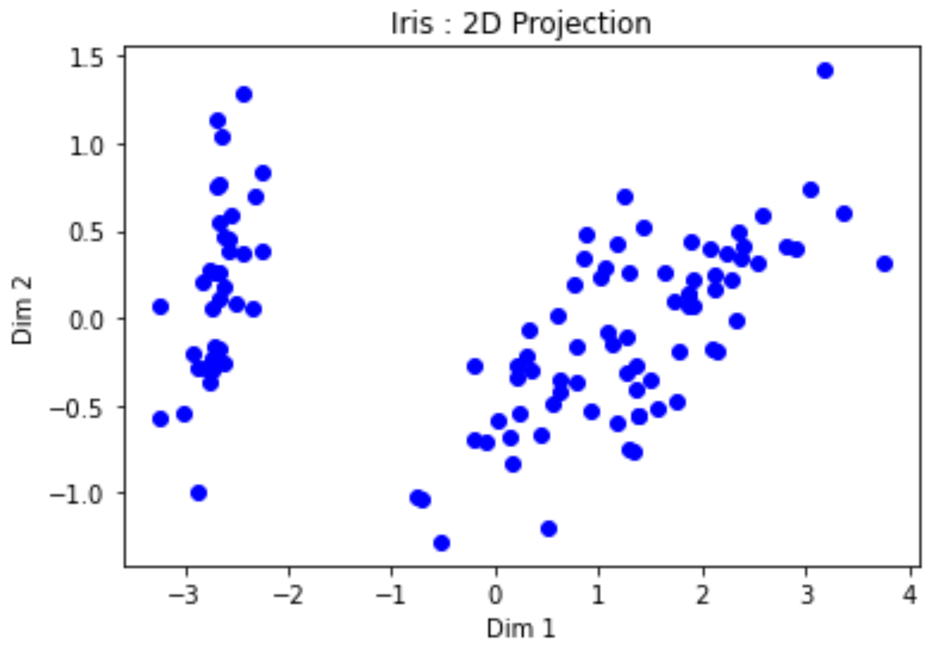
…in all cases finding lower *intrinsic dimension* is useful

## Example : Iris Dataset

*Recall that the Iris dataset has 4 features:*
*sepal length / width, petal length / width…*



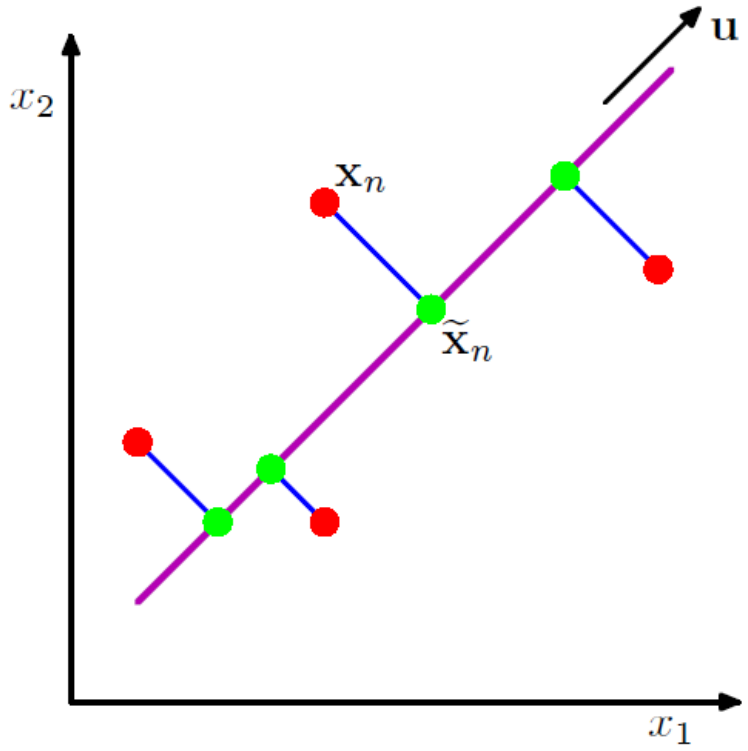
## Example : Iris Dataset



Data still cluster in a two-dimensional subspace

We can fit model in 2D to reduce complexity, visualize results, etc.

## Linear Dimensionality Reduction



*Project data onto a line or plane…*

*…one of the simplest dimensionality reduction approaches*

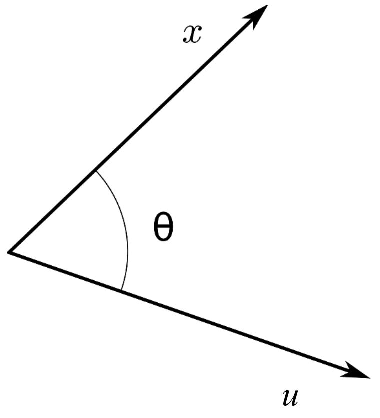
**First, let's review some linear algebra…**

[ Source: Bishop, C. ]

## Inner Products

Recall the definition of an *inner product*:

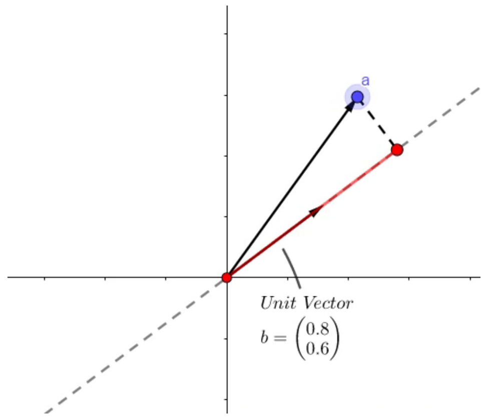
$$u^T x = u_1 x_1 + u_2 x_2 + \ldots + u_D x_D$$
$$= \sum_{d=1}^{D} u_d x_d$$



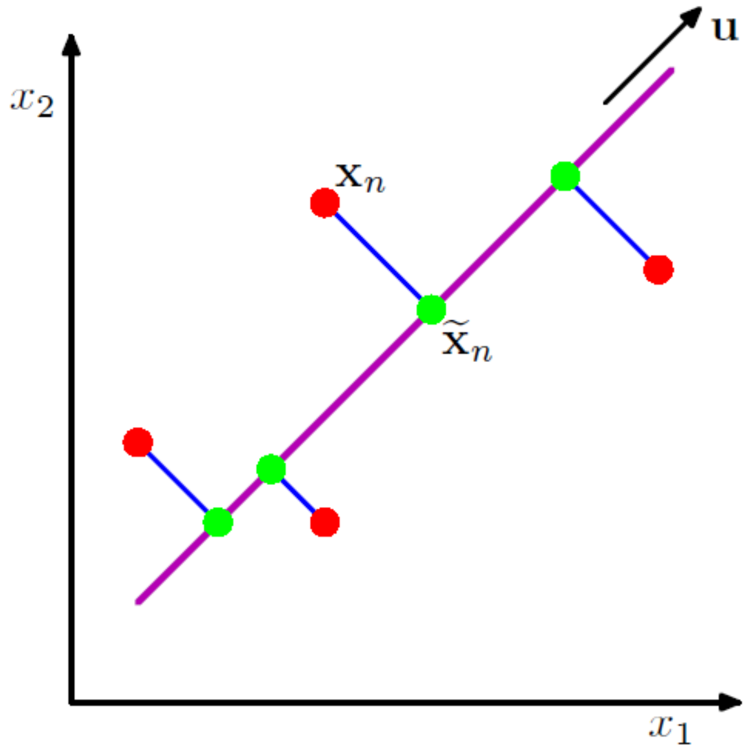
Equivalently, projection of one vector onto another,

$u^T x = |u||x| \cos\theta$ where $|x| = \sqrt{\sum_d x_d^2}$

**Vector Norm**



## Linear Dimensionality Reduction



*Projecting data onto a vector is a simple inner product,*

$$\widetilde{x}_n = u^T x_n$$

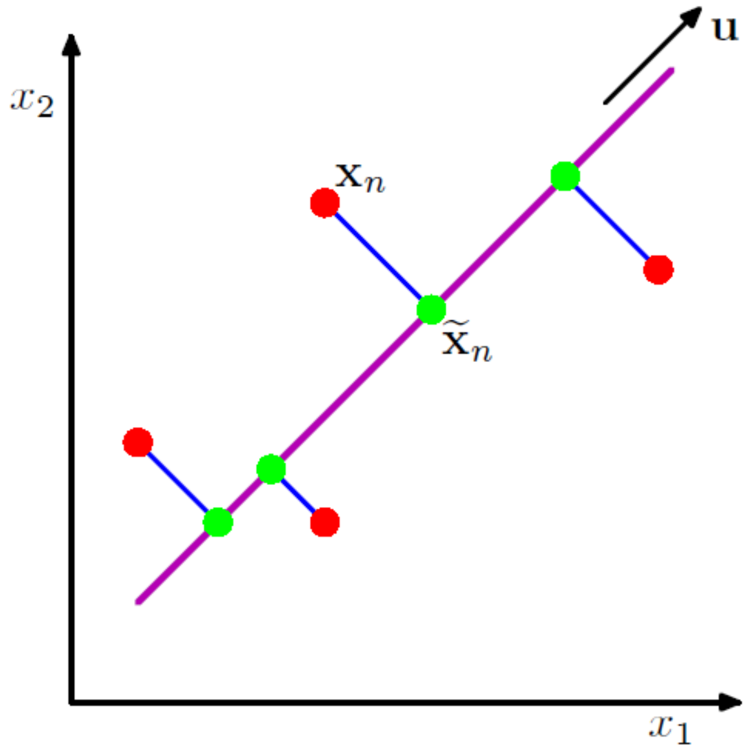
We call $u$ the *linear subspace*

(usually, you take u such that ||u|| = 1)

**Question** Why would dimensionality reduction be better than feature selection (e.g. choose 1-D features X1 or X2)?

[ Source: Bishop, C. ]

# Linear Dimensionality Reduction



*Projecting data onto a vector is a simple inner product,*

$$\widetilde{x}_n = u^T x_n$$

We call $u$ the *linear subspace*

(usually, you take u such that ||u|| = 1)

**Answer** No features are discarded (uses all the data),

$$\widetilde{x}_n = u_1 x_{n1} + u_2 x_{n2}$$

[ Source: Bishop, C. ]

## Linear Dimensionality Reduction

- E.g.) Shoe size
- There are several features that determine ‘largeness’: shoe length, width, height, texture…

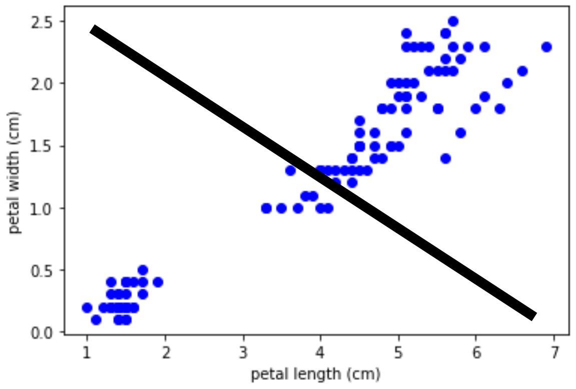
- Customer: too many things to learn… just give me one.
  - Feature selection: shoe length is the most important. Forget the rest.
  - Linear dimensionality reduction: let’s define

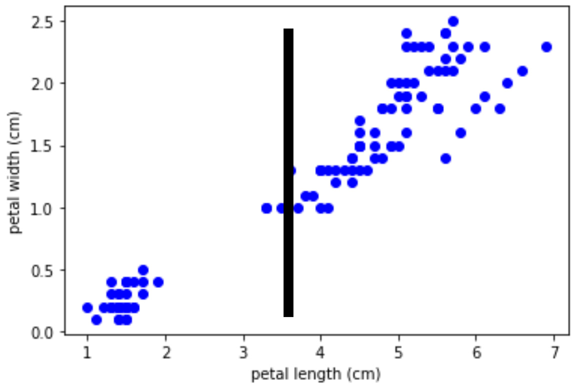
$$Largeness = 0.8 * length + 0.2 * width + 0.1 * height$$

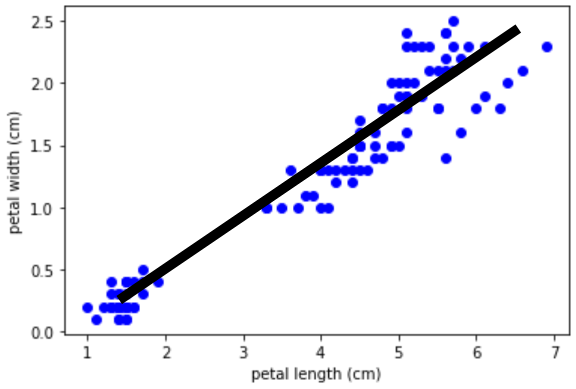
  and show it to the customer.

# Linear Dimensionality Reduction

*Which choice of subspace is best?  And why?*

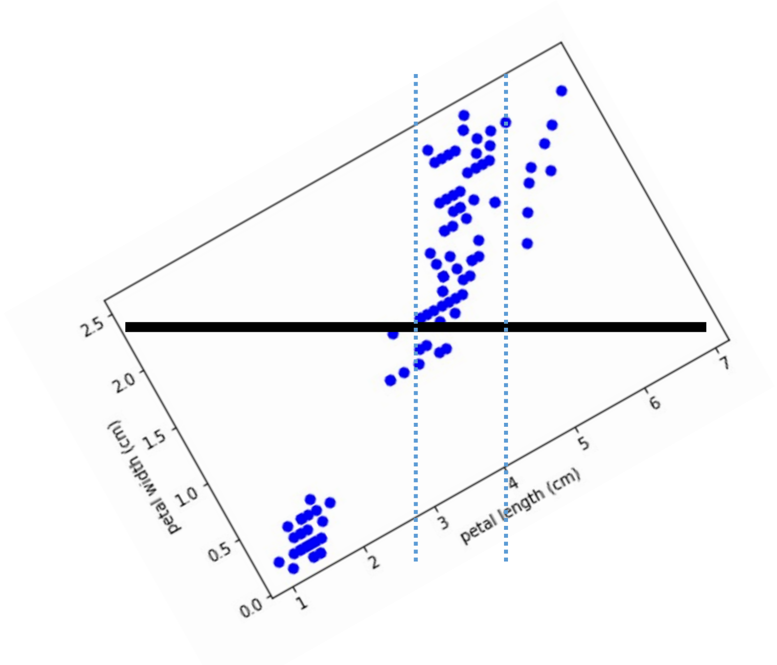




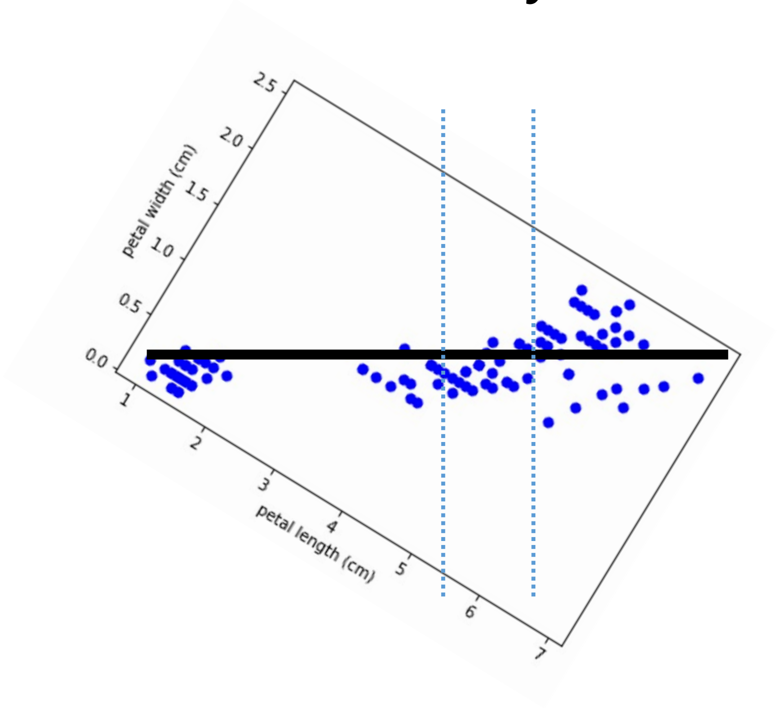


## Linear Dimensionality Reduction

*Which choice of subspace is best?  And why?*



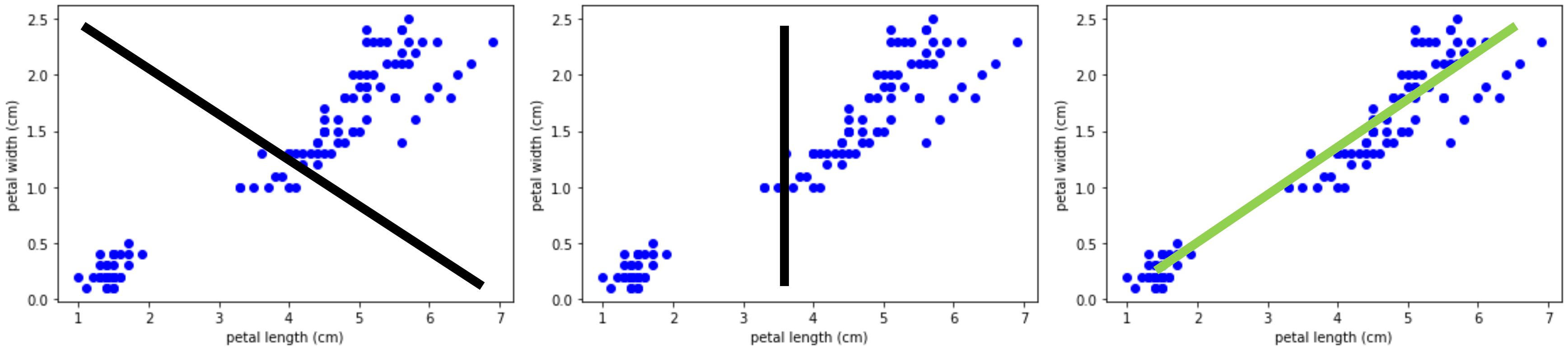
Too many samples are within a similar range…
Does not ‘explain’ your data enough!



Captures large variations of ‘size’
You can understand its largeness appropriately.

## Linear Dimensionality Reduction

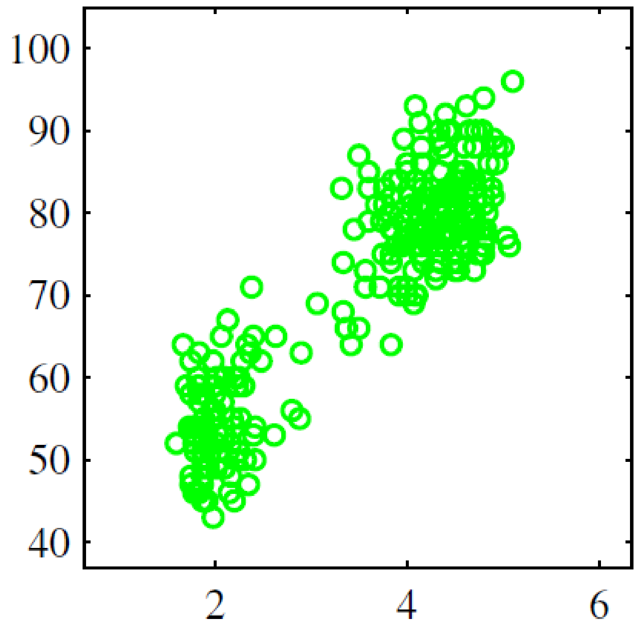
*Which choice of subspace is best? And why?*

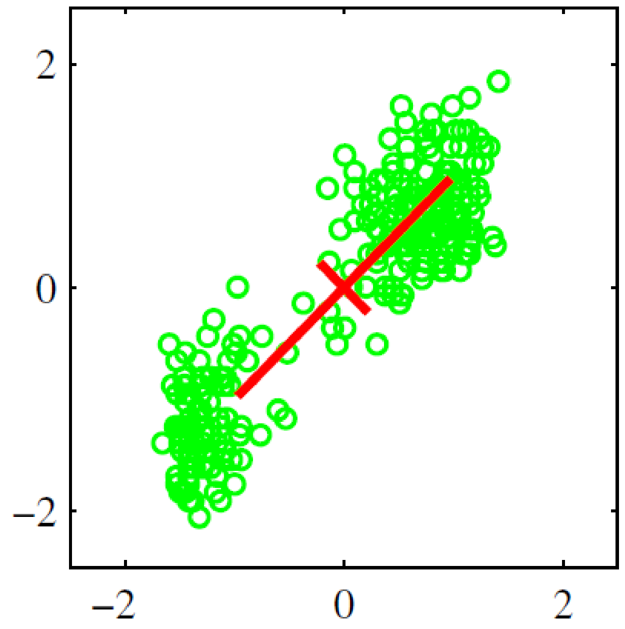


**Idea** Choose the subspace that captures the most variation in the original data

## Principal Component Analysis (PCA)

Identify directions of *maximum variation* as subspaces…





…we call each direction a *principal component*

[ Source: Bishop, C. ]

## Principal Component Analysis (PCA)

First, center the data by subtracting the sample mean,

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

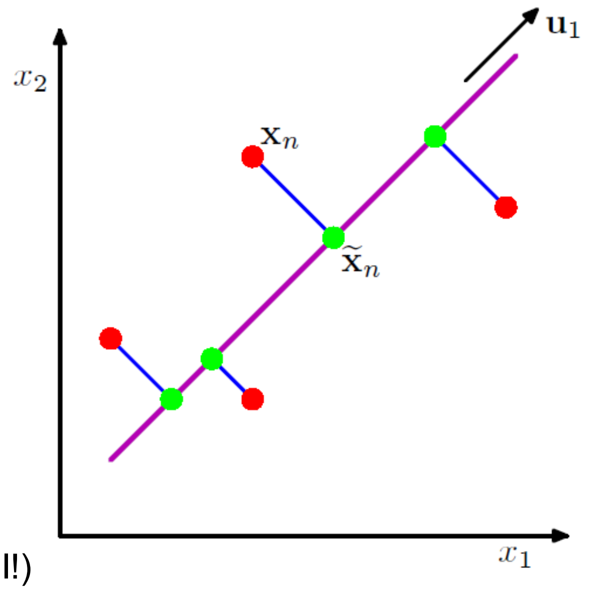
Variance of projected subspace,

$$\frac{1}{N}\sum_{n=1}^{N}\left(\underbrace{u^T x_n}_{\text{Projection of nth data point}} - \underbrace{u^T \bar{x}}_{\text{Projection of mean}}\right)^2$$

**Projection of nth data point**

**Projection of mean**

(=mean of the projection as well!)



## Maximum Variance Formulation

A little algebra…

$$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{u^T(x_n - \bar{x})\right\}^2$$ Pull out u

Quadratic form $$= \frac{1}{N}\sum_{n=1}^{N} u^T (x_n - \bar{x})(x_n - \bar{x})^T u$$

Define: $S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T$

Then: $$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 = u^T S u$$

**This is what we will optimize over u**

## Maximum Variance Formulation

*Find u so that projected variance is maximal…*

$$\max_u u^T S u$$

Don't want to *cheat* with large magnitude u, so we add a constraint $||u|| = 1$

Long story short, the solution can be obtained via so-called **eigen decomposition**.

Eigen decomposition of $S \in \mathbb{R}^d$ gives $\{(\underbrace{\lambda_i}_{\text{eigenvalue}} \in \mathbb{R}, \underbrace{v_i}_{\text{eigenvector}} \in \mathbb{R}^D)\}_{i=1}^D$ where $\lambda_i \geq 0$ and $||v_i|| = 1$. The solution u will be the top-1 eigen vector.

Terminology: "take the top-k eigen vectors": choose top-k $\lambda_i$ value and then pick their corresponding eigenvectors.

## Minimum Variance Formulation

*Find u so that projected variance is maximal…*

$$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 \qquad\qquad \max_u u^T S u$$

- This gives us a 1-dimensional projection: perform $x_n' \leftarrow u^\top x_n$

- What if we want to obtain a M-dimensional subspace?

- $\max_{U\in\mathbb{R}^{D\times M}:\|U_{\cdot,i}\|=1} \frac{1}{N}\sum_{n=1}^{N}(U^\top(x_n-\bar{x}))^2$

  ↑ i-th column of U

  How would this be related to the eigen decomposition?

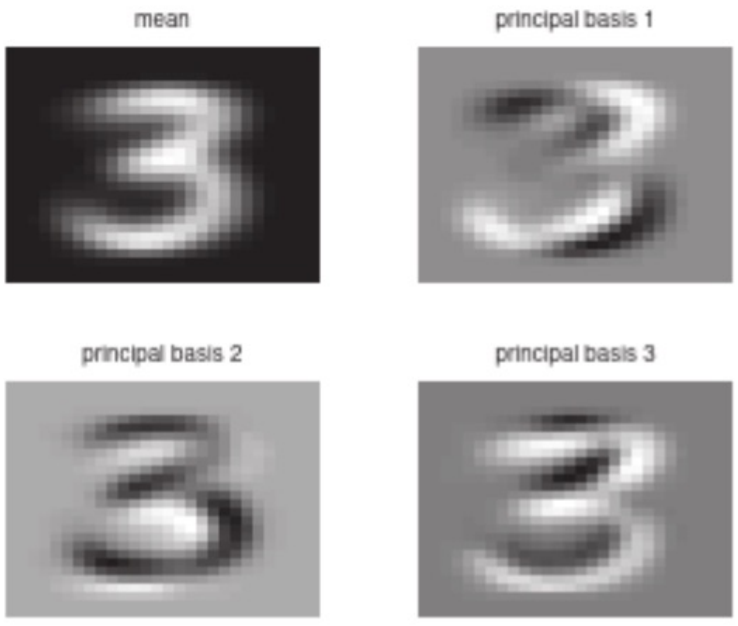
turns out, choosing $U_{\cdot,i}$ to be the $i$-th largest eigenvector of S will do the job!

## Summary

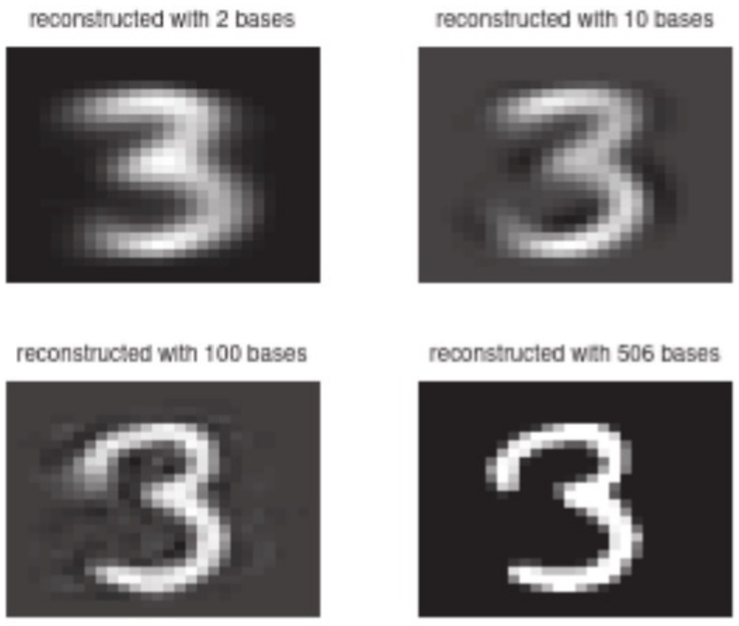
- Algorithm PCA:
  - Input: $M$: the target dimensionality (should be < D)
  - Compute $S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^\top$ and then compute the eigen decomposition
    ($aa^\top$ =np.outer(a,a))
  - Choose the top-M eigenvectors: $v_1, \ldots, v_M$
    (these are called *principal components*)
  - Return $x_n' = \left(v_1^\top(x_n - \bar{x}), \ldots, v_M^\top(x_n - \bar{x})\right)^\top \in \mathbb{R}^M, \forall n = 1, \ldots, N$

- Time complexity: there are algorithms that can find the top-M eigenvectors O(M$D^2$) time

## Visual example: how PCA works



(a)


reconstructed with 2 bases
reconstructed with 10 bases
reconstructed with 100 bases
reconstructed with 506 bases

(b)

## `sklearn.decomposition`.PCA

## Parameters

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

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

**copy : *bool, default=True***

If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

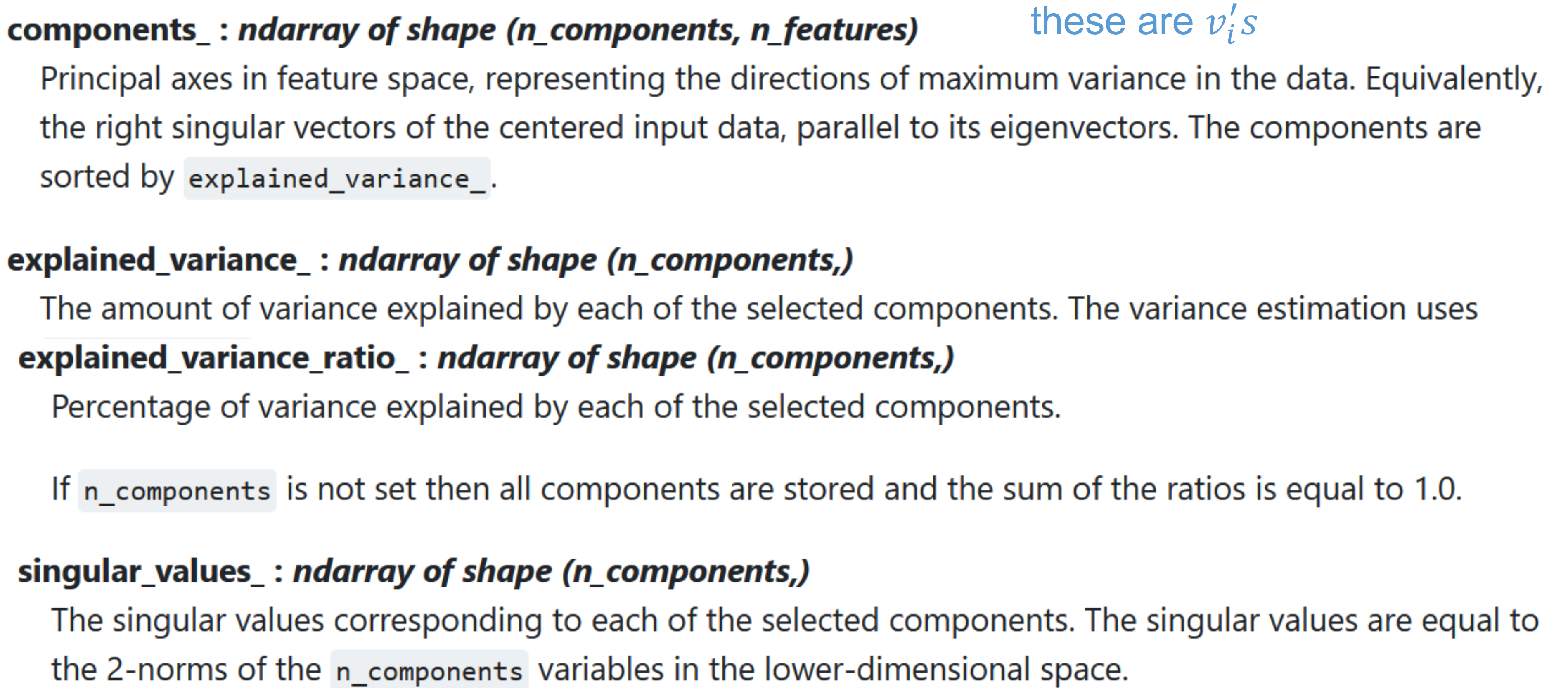
**whiten : *bool, default=False***

When True (False by default) the `components_` vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

## sklearn.decomposition.PCA

### Attributes

these are $v_i's$



**components_ : *ndarray of shape (n_components, n_features)***

Principal axes in feature space, representing the directions of maximum variance in the data. Equivalently, the right singular vectors of the centered input data, parallel to its eigenvectors. 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. The variance estimation uses

**explained_variance_ratio_ : *ndarray of shape (n_components,)***

Percentage of variance explained by each of the selected components.

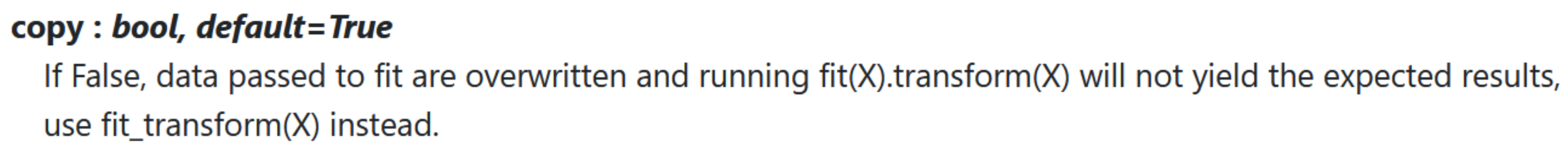
If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

**singular_values_ : *ndarray of shape (n_components,)***

The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

## Caution

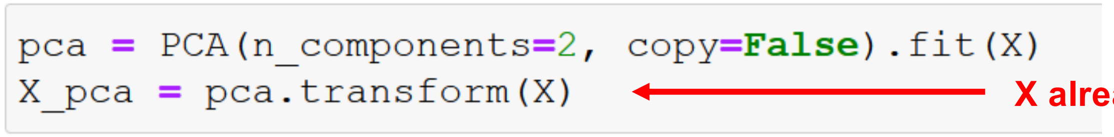
Careful with the following parameter,



**copy : *bool, default=True***

If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

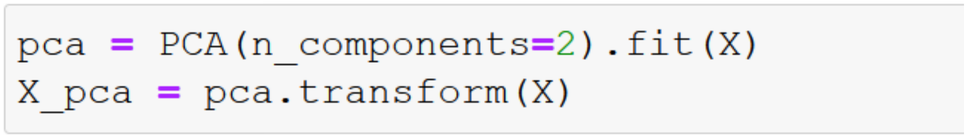
**Wrong**



```
pca = PCA(n_components=2, copy=False).fit(X)
X_pca = pca.transform(X)
```

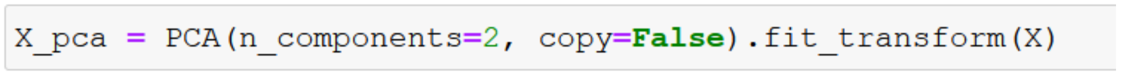
**X already modified**

**Right**



```
pca = PCA(n_components=2).fit(X)
X_pca = pca.transform(X)
```

**Right**



```
X_pca = PCA(n_components=2, copy=False).fit_transform(X)
```

## Example : PCA on Iris Data

Load Iris data without labels,

```
iris = datasets.load_iris(as_frame=True)
X = iris.data
```

Find PCA with 2 principal components,

```
pca = PCA(n_components=2).fit(X)
X_pca = pca.transform(X)
```

How much variance did we capture?

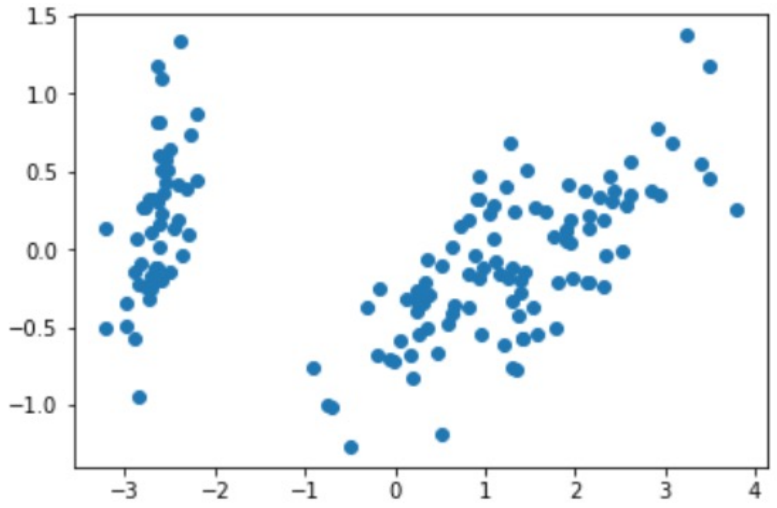
```
expvar = pca.explained_variance_ratio_
print('% Variance in 1st PC: ', expvar[0])
print('% Variance in 2nd PC: ', expvar[1])
print('Total explained variance: ', sum(expvar))
```

```
% Variance in 1st PC:  0.9246187232017271
% Variance in 2nd PC:  0.053066483117067804
Total explained variance:  0.977685206318795
```

## Example : PCA on Iris Data

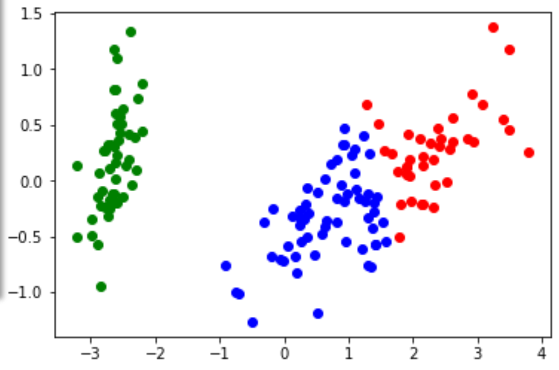
View data in 2-D subspace,

```
plt.scatter(X_pca[:,0], X_pca[:,1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

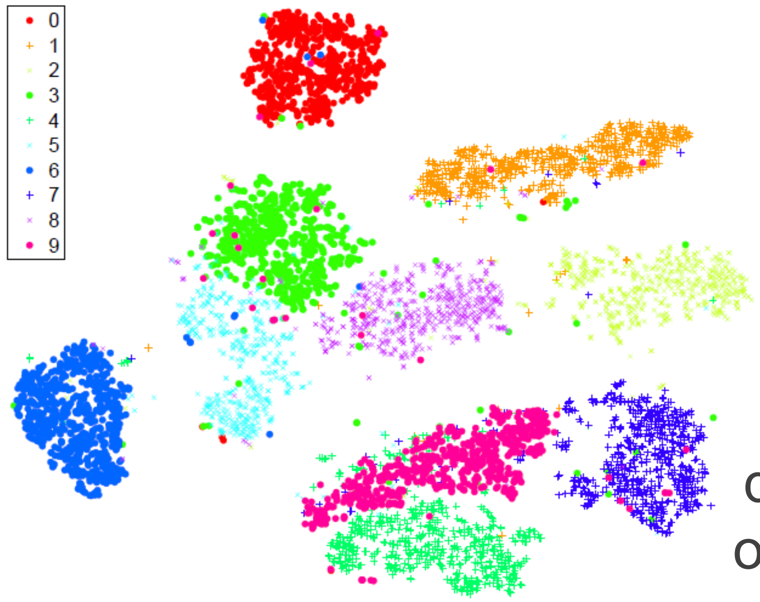


Do K-means clustering in 2-D subspace,

```
kmeans = KMeans(n_clusters=3).fit(X_pca)
labels = kmeans.labels_
fig, ax = plt.subplots()
ax.scatter(X_pca[:,0][labels == 0], X_pca[:,1][labels == 0], c='r')
ax.scatter(X_pca[:,0][labels == 1], X_pca[:,1][labels == 1], c='g')
ax.scatter(X_pca[:,0][labels == 2], X_pca[:,1][labels == 2], c='b')
plt.show()
```



# t-SNE



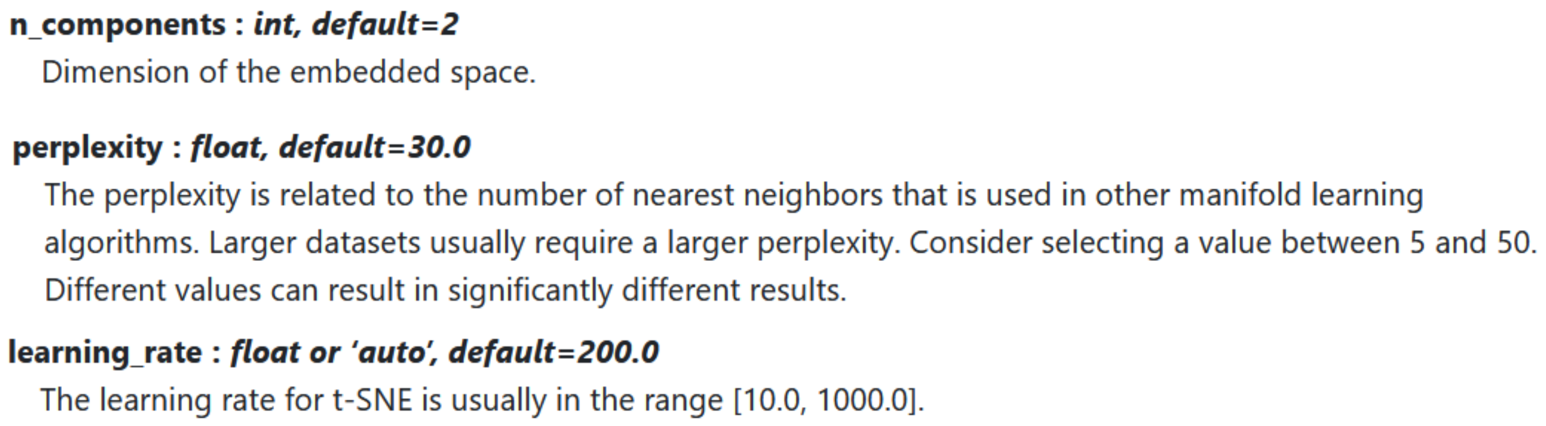
Nonlinear reduction can (potentially) amplify clustering properties

**t-Distributed Stochastic Neighbor Embedding (t-SNE)** Models similarity between data as a Student's-t distribution in high / low dimensions and optimizes reduction to preserve similarity

Visualization shows MNIST digits (recall from lecture on Neural Nets) projected to 2D and clustered

## `sklearn.manifold`.TSNE

### Parameters

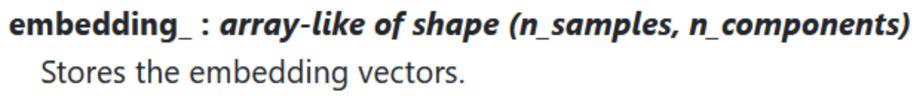


**n_components : *int, default=2***
Dimension of the embedded space.

**perplexity : *float, default=30.0***
The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results.

**learning_rate : *float or 'auto', default=200.0***
The learning rate for t-SNE is usually in the range [10.0, 1000.0].

### Attributes



**embedding_ : *array-like of shape (n_samples, n_components)***
Stores the embedding vectors.

## Example : t-SNE on Iris Dataset

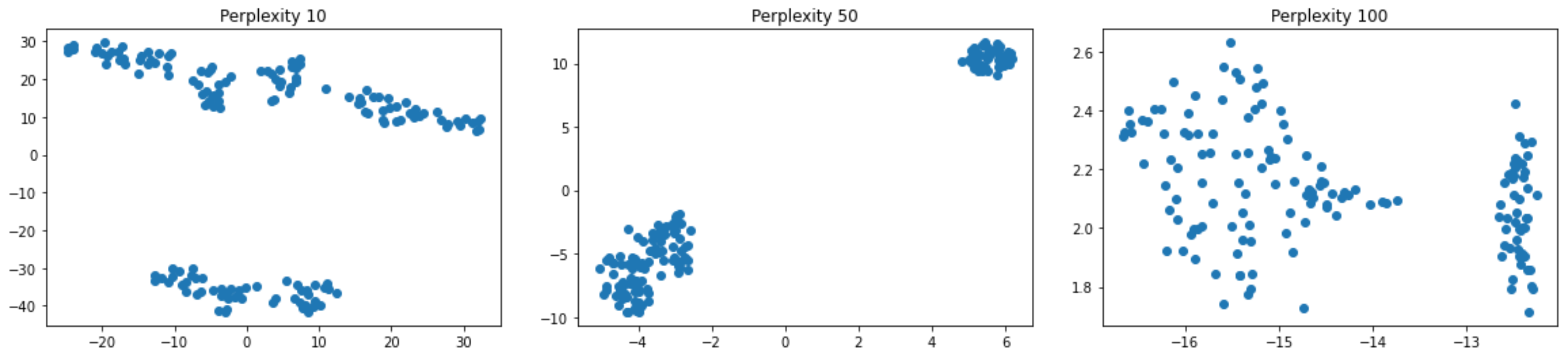
t-SNE can work surprisingly well…

```
from sklearn.manifold import TSNE
perplexity = [20, 50, 100]
for perp in perplexity:
    tsne = TSNE(n_components=2, perplexity=perp)
    X_tsne = tsne.fit_transform(X)
    fig, ax = plt.subplots()
    ax.scatter(X_tsne[:,0], X_tsne[:,1])
    ax.set_title('Perplexity %i' % perp)
plt.show()
```

**One advantage of PCA is that it has no parameters that need tuning (aside from number of PCs)**

**PCA is also much easier to interpret**

…but can be a bit fussy about parameters and unreliable



## Closing Comments

- Nonlinear methods in Scikit-Learn categorized under “manifold learning” in the *manifold* sub-package,
  - Isomap, Locally Linear Embedding, Spectral Embedding, Multidimensional scaling, and of course TSNE
- Other methods related to PCA (in *decomposition* sub-pkg):
  - Factor Analysis, Kernel PCA, Incremental PCA
- For multiple data sources, consider cross-decomposition
  - Canonical Correlation Analysis (CCA)
  - Learns same embedding for both spaces
  - Under *cross_decomposition* sub-package