Unsupervised Learning

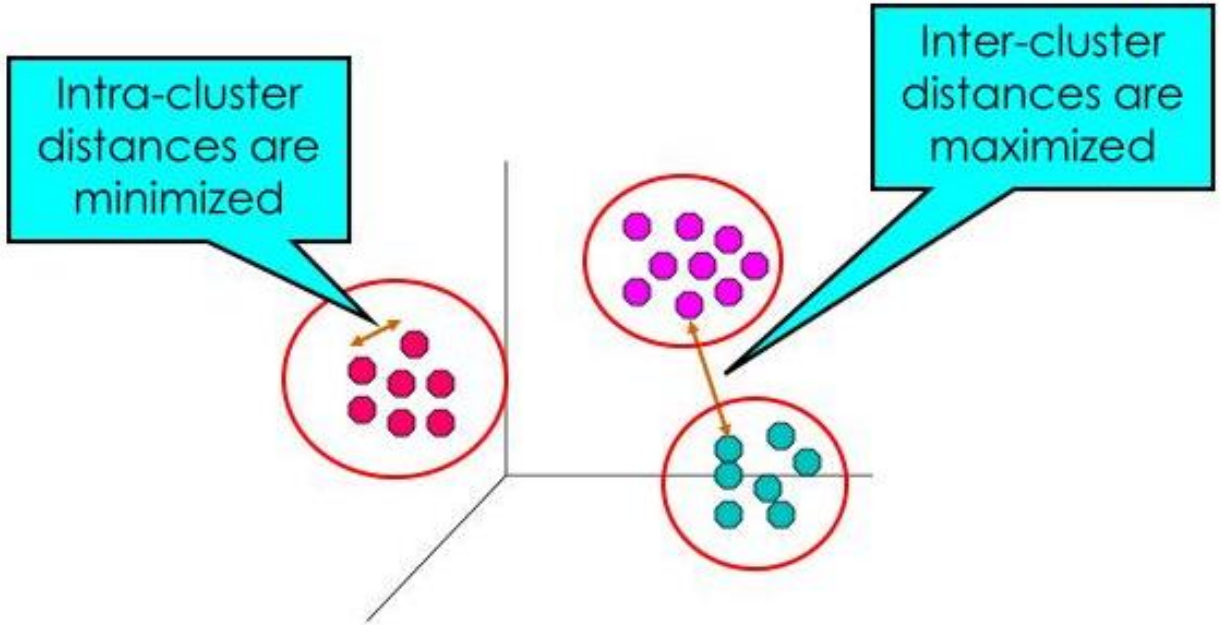
Supervised learning: discover patterns in the data that relate data attributes with a target (class) attribute. These patterns are then utilized to predict the values of the target attribute in future data instances.

Unsupervised learning: The data have no target attribute. We want to explore the data to find some intrinsic structures in them. The goal of unsupervised learning is to find hidden patterns in unlabeled data .

Supervise Learning		Unsupervised Learning	
Regression	Classification	Clustering	Dimensionality reduction

Clustering

Clustering is a technique for finding similarity groups in data, called **clusters**. It finds groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in the other groups

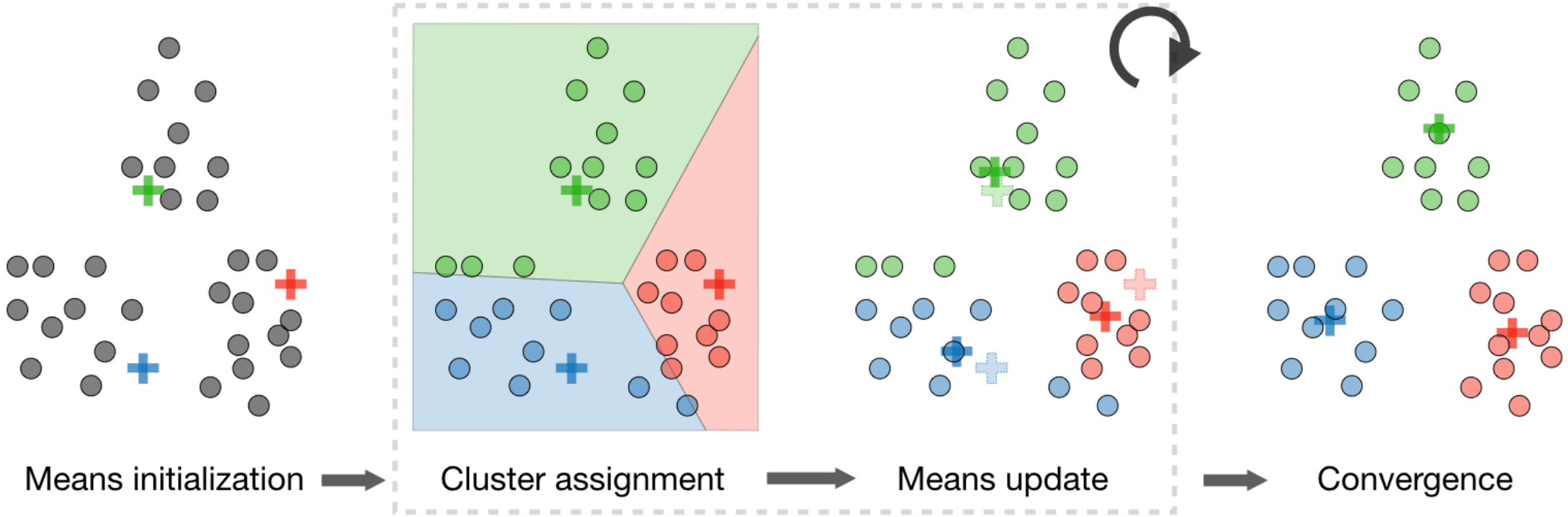


- Given a collection of text documents, we want to organize them according to their content similarities
- In marketing, segment customers according to their similarities to do targeted marketing.

K-means Clustering

K-means Clustering

K clusters

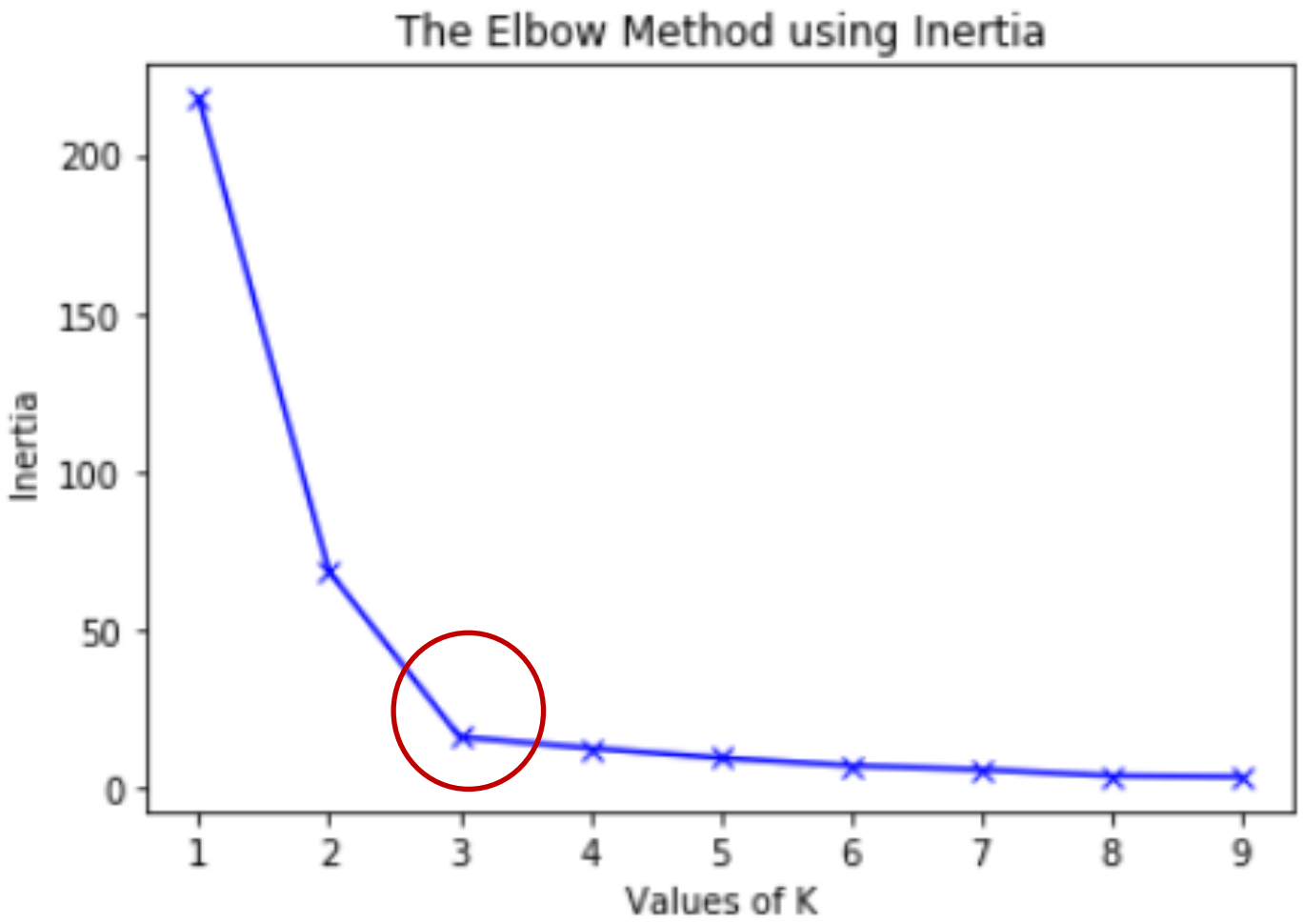


The K-means algorithm aims to choose centroids that minimize the inertia, or within-cluster sum-of-squares criterion:

$$J(c,\mu) = \frac{1}{M}\sum_{i=1}^{M}\left\| x^{(i)} - \mu_{c^{(i)}} \right\|^2$$

centroids

Selecting number of clusters with Elbow plot



$$J(c,\mu) = \frac{1}{M}\sum_{i=1}^{M}\left\| x^{(i)} - \mu_{c^{(i)}} \right\|^2$$

Selecting number of clusters with silhouette analysis

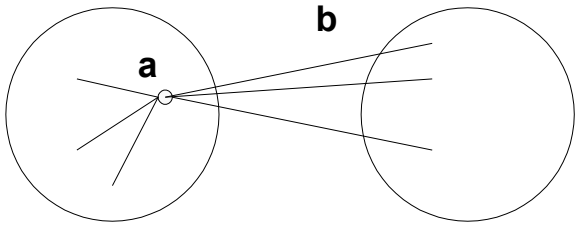
Silhouette Coefficient combines ideas of both cohesion and separation

For an individual point *i*

a = average distance of *i* to the points in the same cluster

b = average distance of *i* to points in nearest cluster

silhouette coefficient of i:



s = (b – a)/max{a, b}, for each i

Typically between -1 and 1. The closer to 1 the better.

sklearn.metrics.silhouette_score	compute the mean Silhouette Coef of all samples
sklearn.metrics.silhouette_samples	compute the Silhouette Coef for each sample

K-means algorithm in Sklearn

sklearn.cluster.**KMeans**(*n_clusters=8, init='k-means++’, n_init=10, max_iter=300, tol=0.0001, verbose=0, random_state=None*)

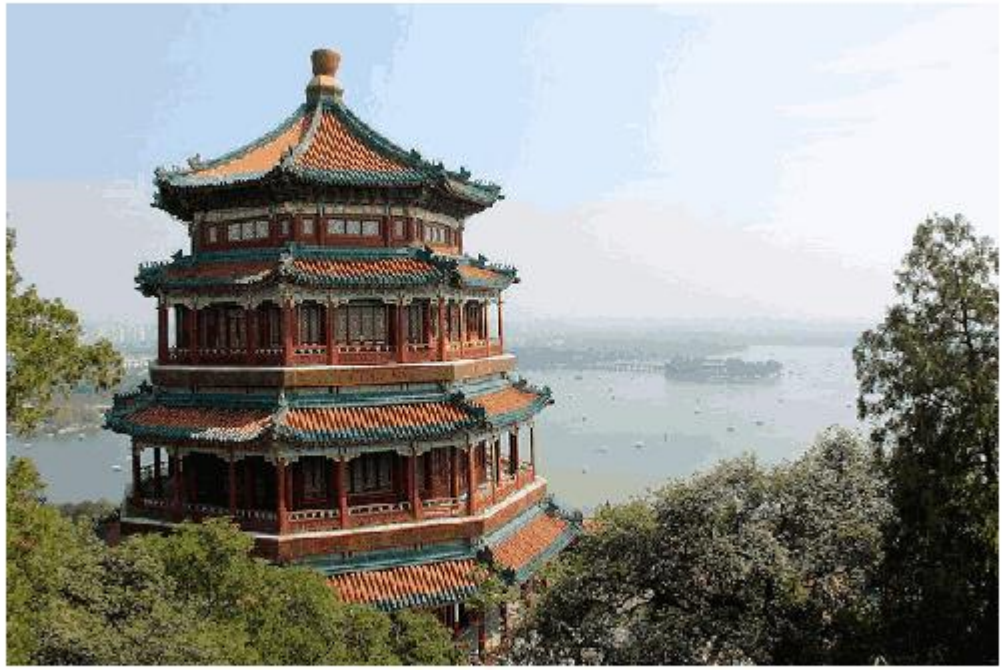
MiniBatchKMeans

Alternative online implementation that does incremental updates of the centers positions using mini-batches. For large scale learning (say n_samples > 10k) MiniBatchKMeans is probably much faster than the default batch implementation.

K-means Example: Color Quantization

Original image (96,615 colors)

Quantized image (64 colors, K-Means)

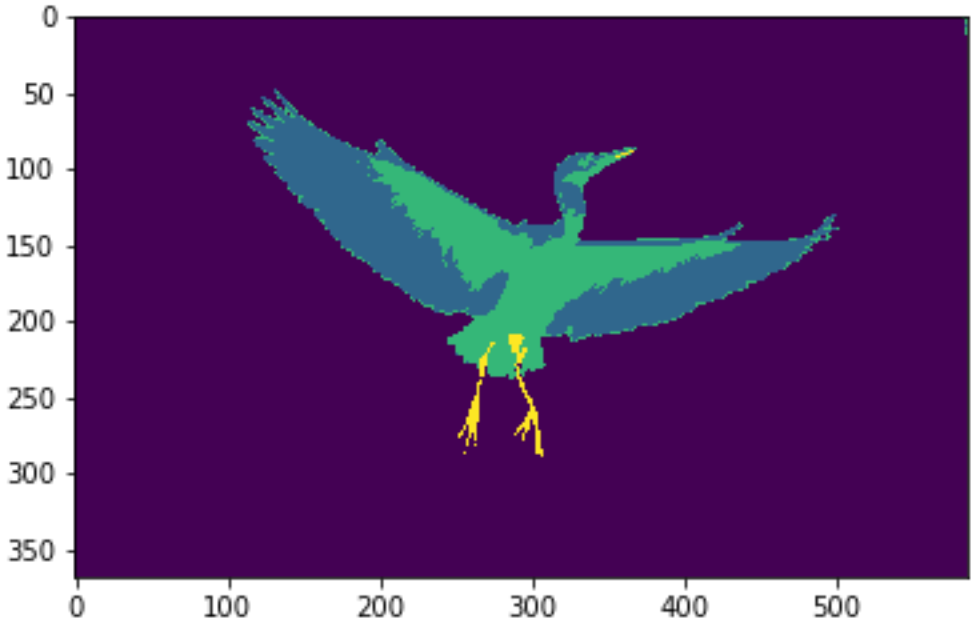


Performs a pixel-wise Vector Quantization (VQ) of an image of the summer palace (China), reducing the number of colors required to show the image from 96,615 unique colors to 64, while preserving the overall appearance quality.

https://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html

K-means Example: Image Segmentation

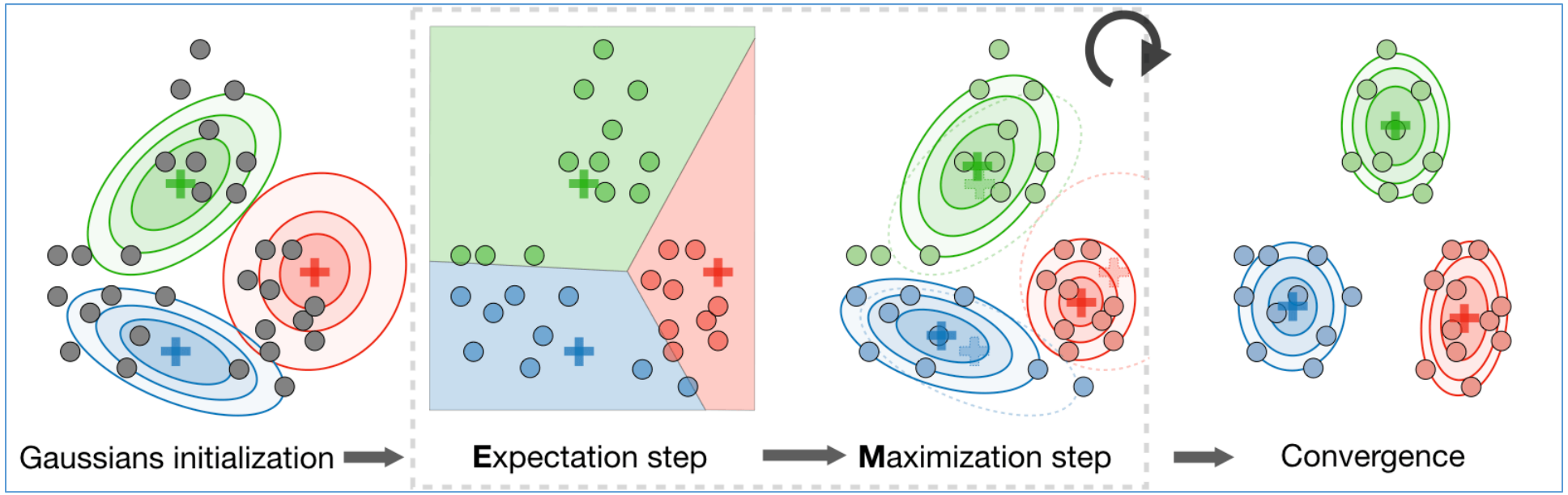
Original Image



sklearn.cluster.KMeans(n_clusters = 4)

Expectation Maximization (EM) algorithm

EM algorithm intuition



- <u>Initialization</u>:

  Initialize K clusters: $C_1, \ldots, C_K$ - (μ_j, Σ_j) and $P(C_j)$ for each cluster j.

- <u>Iteration Steps</u>:

  Expectation step: Estimate the cluster of each data $p(C_j \mid x_i)$

  Maximization step: Re-estimate the cluster parameters

  $$(\mu_j, \Sigma_j), p(C_j) \text{ For each cluster } j$$

Input: $\{x_i\}_{i=1}^{M}$

Output:

cluster parameters (μ_j, Σ_j) and $P(C_j)$ for each cluster j and clustering result $p(C_j/x_i)$

EM algorithm in Sklearn

sklearn.mixture.**GaussianMixture**(*n_components=1, covariance_type='full', init_params='kmeans'*)

covariance_type : *{'full' (default), 'tied', 'diag', 'spherical'}*

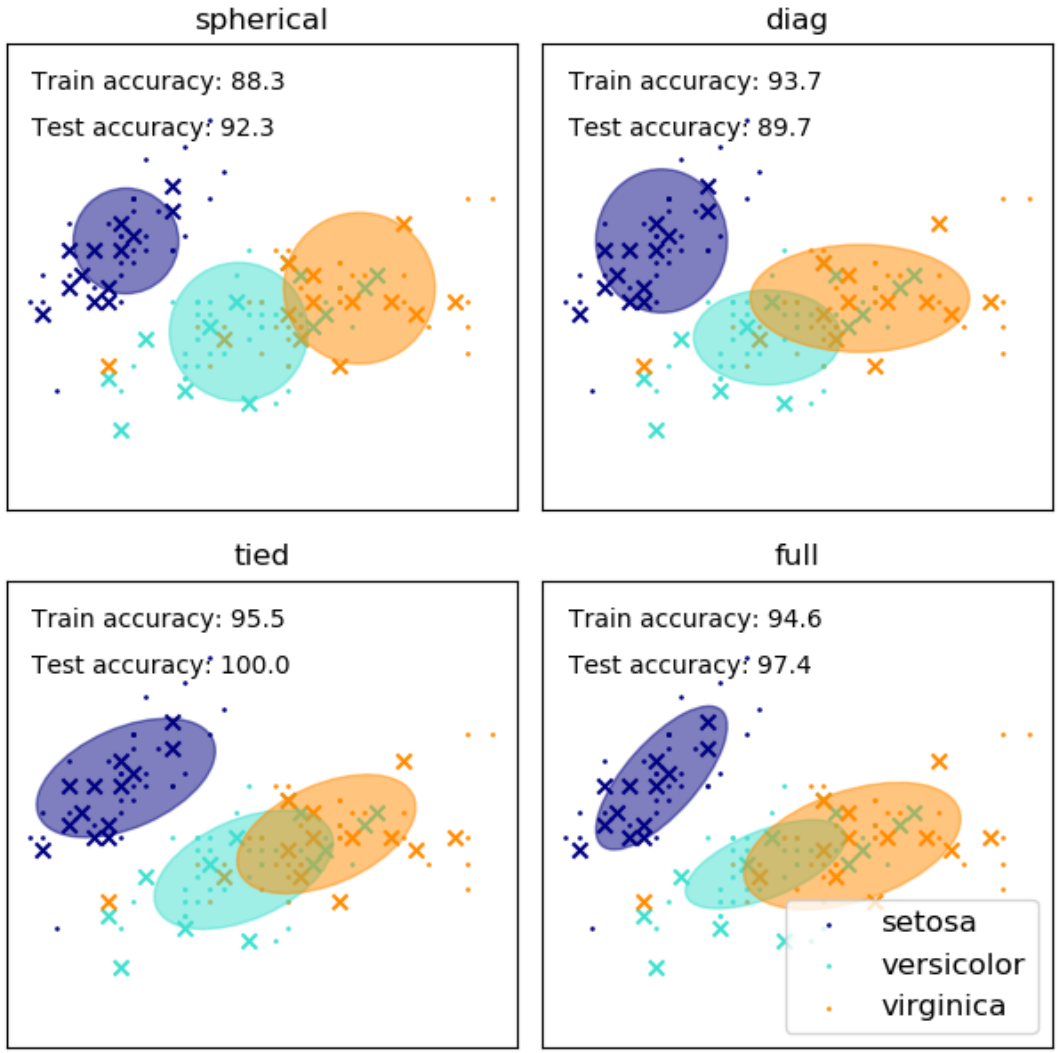
String describing the type of covariance parameters to use. Must be one of:

'full' - each component has its own general covariance matrix

'tied' - all components share the same general covariance matrix

'diag' - each component has its own diagonal covariance matrix

'spherical' - each component has its own single variance



https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_covariances.html#sphx-glr-auto-examples-mixture-plot-gmm-covariances-py

EM algorithm in Sklearn

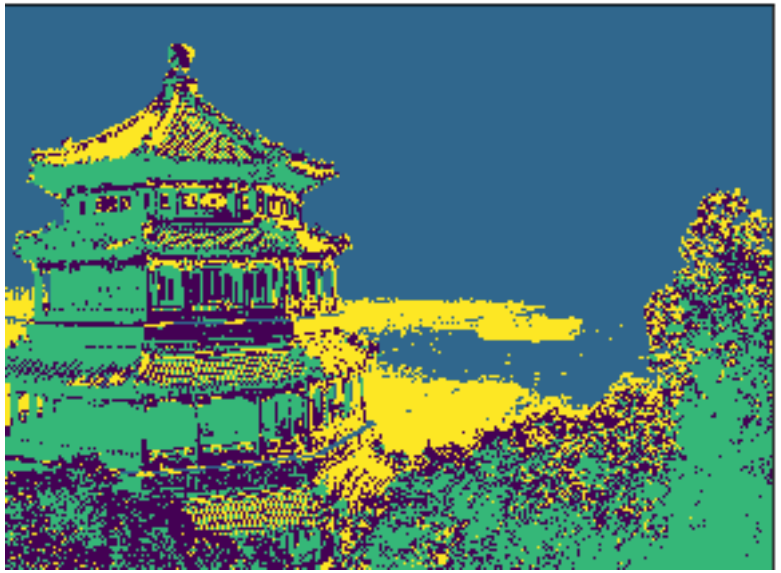
Attributes:

weights_ : ***array-like, shape (n_components,)***
The weights of each mixture components.
means_ : ***array-like, shape (n_components, n_features)***
The mean of each mixture component.
covariances_ : ***array-like***
The covariance of each mixture component.

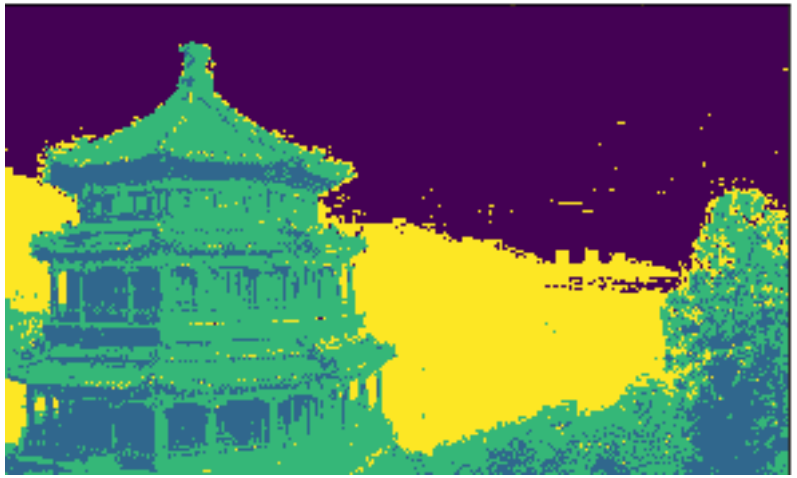
EM algorithm example

Original image

sklearn.cluster.
KMeans(n_clusters = 4)



sklearn.mixture.
GaussianMixture
(n_components=4)

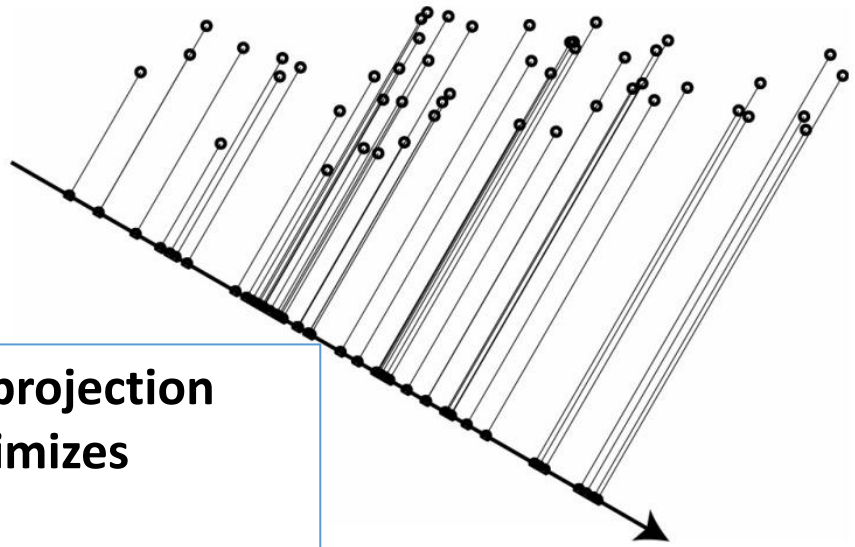


Dimensionality Reduction

Principal Component Analysis (PCA)

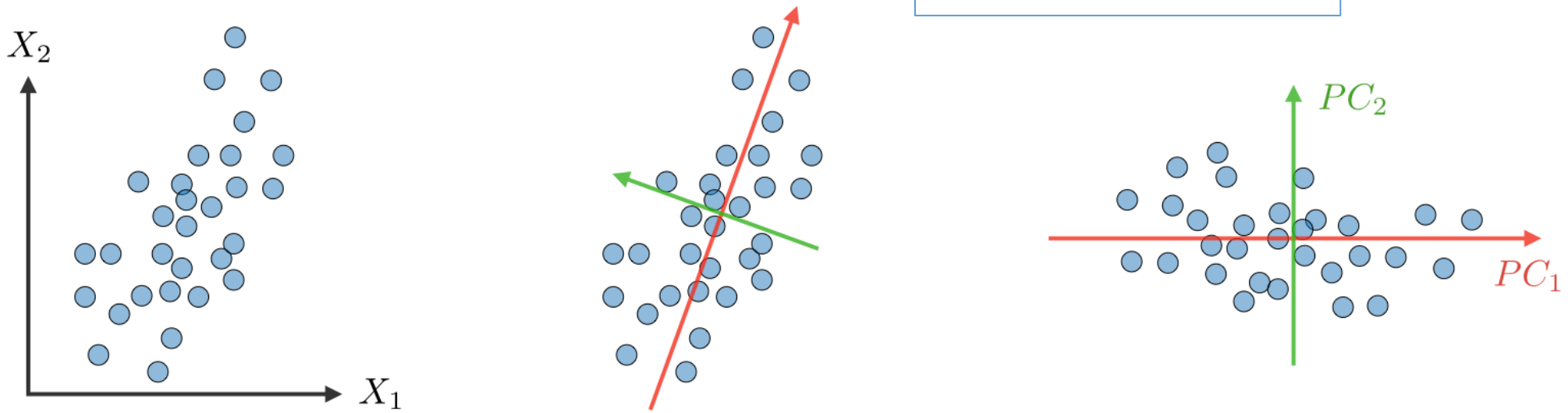
Principal Component Analysis (PCA): 2D case, intuition

PCA is the most popular dimensionality reduction algorithm. First it identifies the hyperplane that lies closest to the data, and then it projects the data onto it.



It is essential rotating the coordinate axes so higher-variance come first

Find the projection that maximizes variance



Data in feature space → Find principal components → Data in **p**rincipal **c**omponents space

PCA

Covariance matrix:

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \dots & \sigma_{1n} \\ \dots & \dots & \\ \sigma_{1n} & & \sigma_n^2 \end{pmatrix}$$

$$\sigma_{jk} = \frac{1}{m-1}\sum_{i=1}^{m}\left(x_j^{(i)} - \underbrace{\mu_j}_{0}\right)\left(x_k^{(i)} - \underbrace{\mu_k}_{0}\right)$$

For standardized data:

$$\boxed{\Sigma = \frac{1}{m-1} X^T X}$$

Principal axes are eigenvectors of matrix Σ

$$\Sigma\vec{v} = \lambda\vec{v}, \;\; \det(\Sigma - \lambda E) = 0$$

λ are eigenvalues

$\vec{v}$ are eigenvectors

Proof: https://en.wikipedia.org/wiki/Rayleigh_quotient

PCA

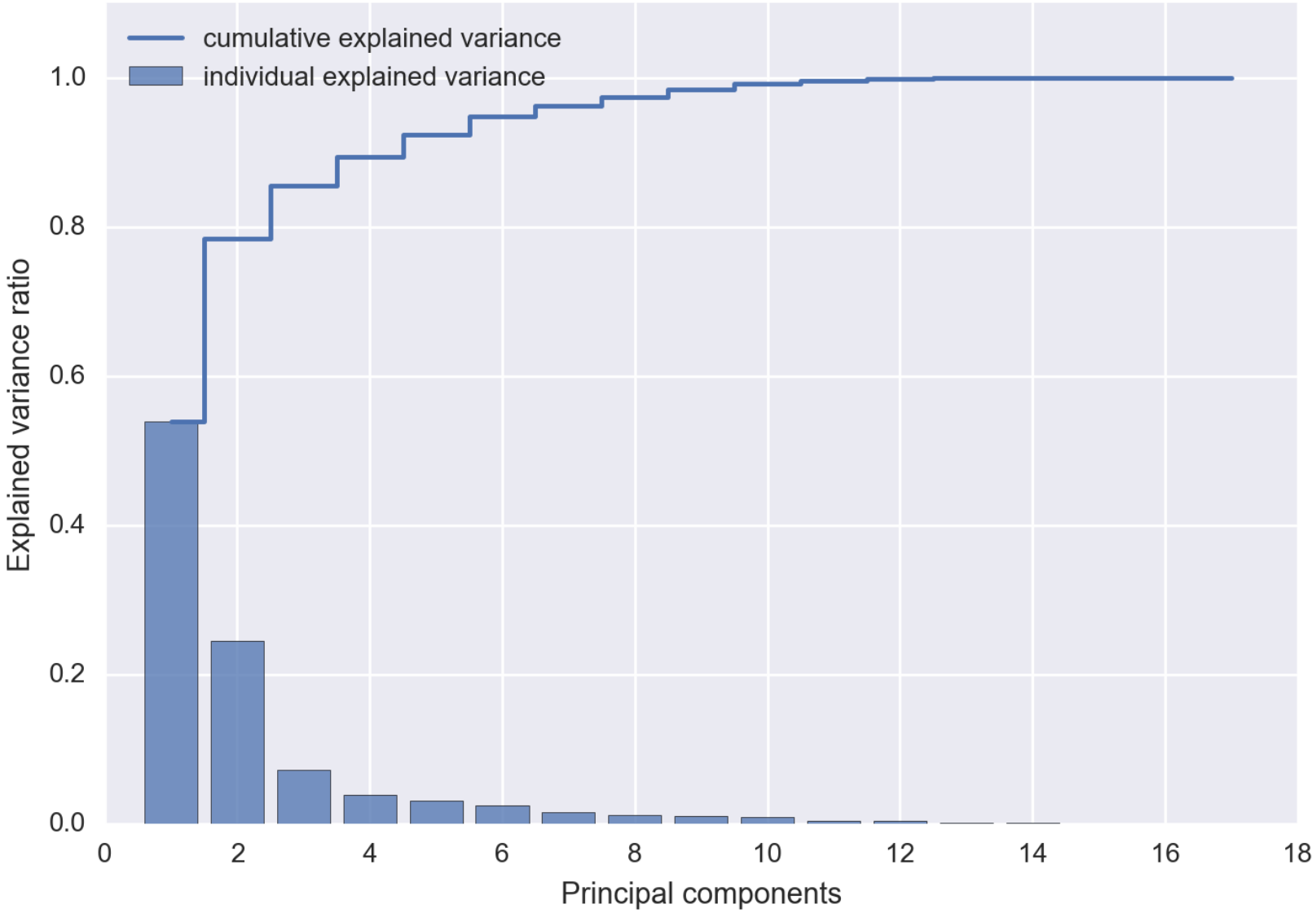
- eigenvectors are uncorrelated (orthogonal)
- the eigenvalues, λ_1, λ_2, ... λ_n are the variances of the coordinates on each principal component axis
- the sum of all *n* eigenvalues equals to the sum of the variances of the original variables
- each eigenvector represents the "contribution" of each variable to the principal component axis

$$\frac{\lambda_j}{\sum_{j=1}^{n} \lambda_j}$$

- explained variance ratio (it indicates the proportion of the dataset's variance that lies along the axis of each principal component)

PCA: Choosing the right number of dimensions

It is generally preferred to choose the number of dimensions that add up to a sufficiently large portion of variance (e.g. 95%).



Note: for data visualization k = 2 or 3

PCA: Build the new reduced dataset

Sort eigenvalues $\lambda_1, \lambda_2, \ldots \lambda_n$ in descending order

Choose first k eigenvalues; compose the matrix of chosen k eigenvectors coordinates.

$$W = \begin{pmatrix} | & \cdots & | \\ \vec{v}_1 & \cdots & \vec{v}_k \\ | & \cdots & | \end{pmatrix}$$

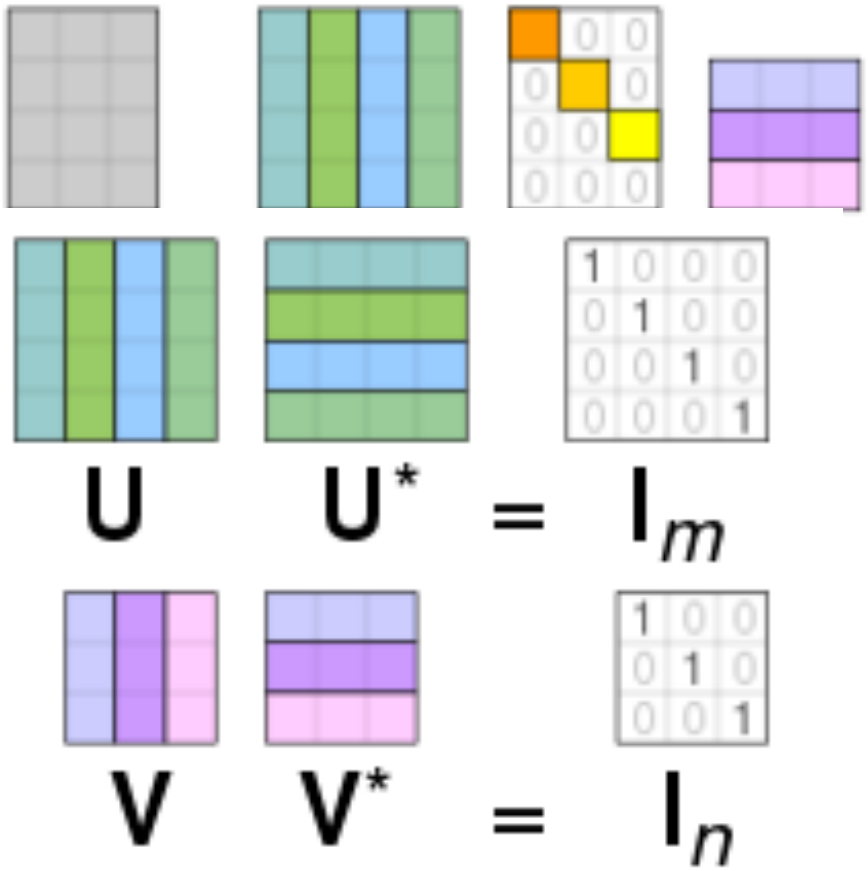
Projection the training set down to k dimension $\tilde{X}_{mxk} = X_{mxn} \cdot W_{nxk}$

Initial data recovering $X_{mxn} = \tilde{X}_{mxk} \cdot W^T$

PCA and SVD decomposition

Singular value decomposition (SVD) is standard matrix factorization technique.

$$X_{mxn} = U_{mxn} \cdot D_{nxn} \cdot V^T_{nxm}$$



Columns of U matrix are eigenvectors of XX^T

Columns of V matrix are eigenvectors of X^TX

Diagonal elements of D matrix are known as singular values of X

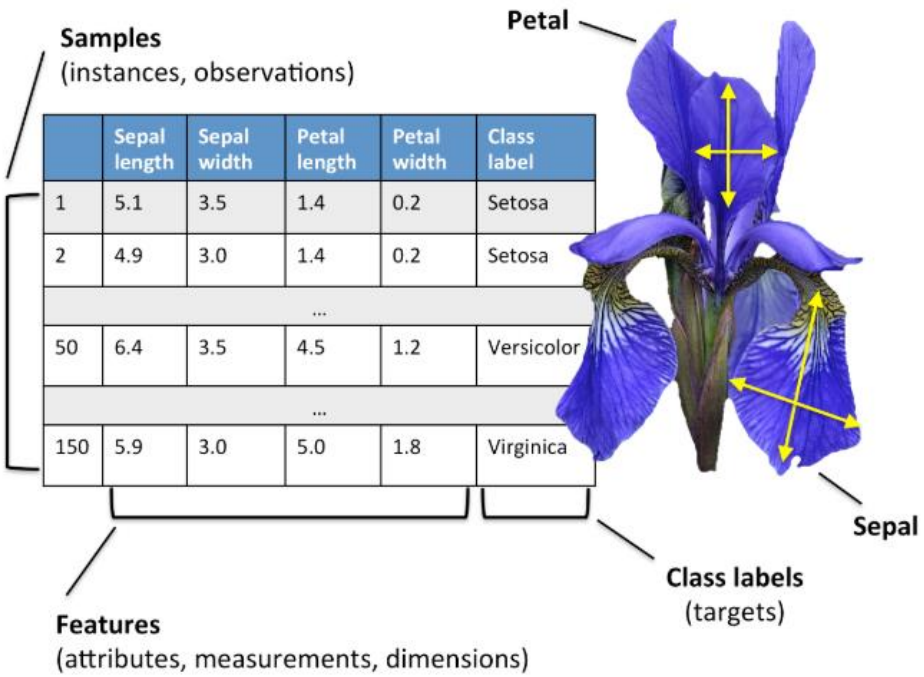
$$D = \begin{pmatrix} s_1 & 0 & \\ 0 & s_2 & \\ & & \dots \end{pmatrix}, s_1 \geq s_2 \dots$$

https://en.wikipedia.org/wiki/Singular_value_decomposition

PCA in Python (Iris dataset)

```
from sklearn.preprocessing import StandardScaler
from sklearn import datasets

iris = datasets.load_iris()
X = iris.data
X_std = StandardScaler().fit_transform(X)
cov_mat = X_std.T.dot(X_std) / (X_std.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

```
eig_vecs
[[ 0.5223  -0.3723  -0.7210 0 .2619]
[-0.2633   -0.9255   0.2420  -0.1241]
[ 0.5812   -0.02109 0.1408  -0.8011]
 [ 0.5656  -0.0654   0.6338  0.5235]]
eig_vals
 [2.9303   0.9274  0.14834  0.0207]
```

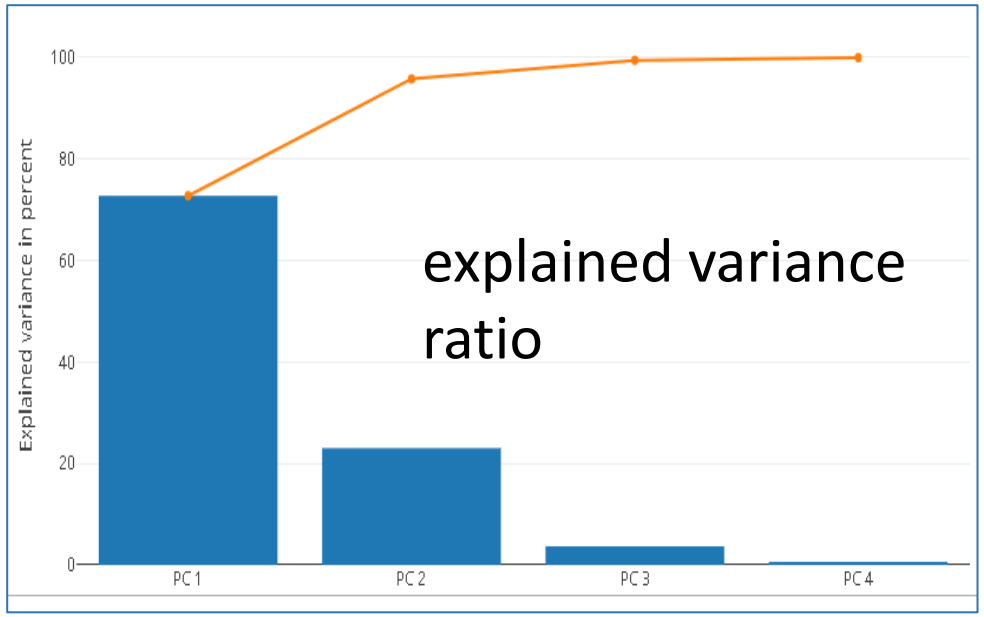
```
u,s,v = np.linalg.svd(X_std)
print(v.T)
```

s://plot.ly/python/v3/ipython-notebooks/principal-component-analysis/

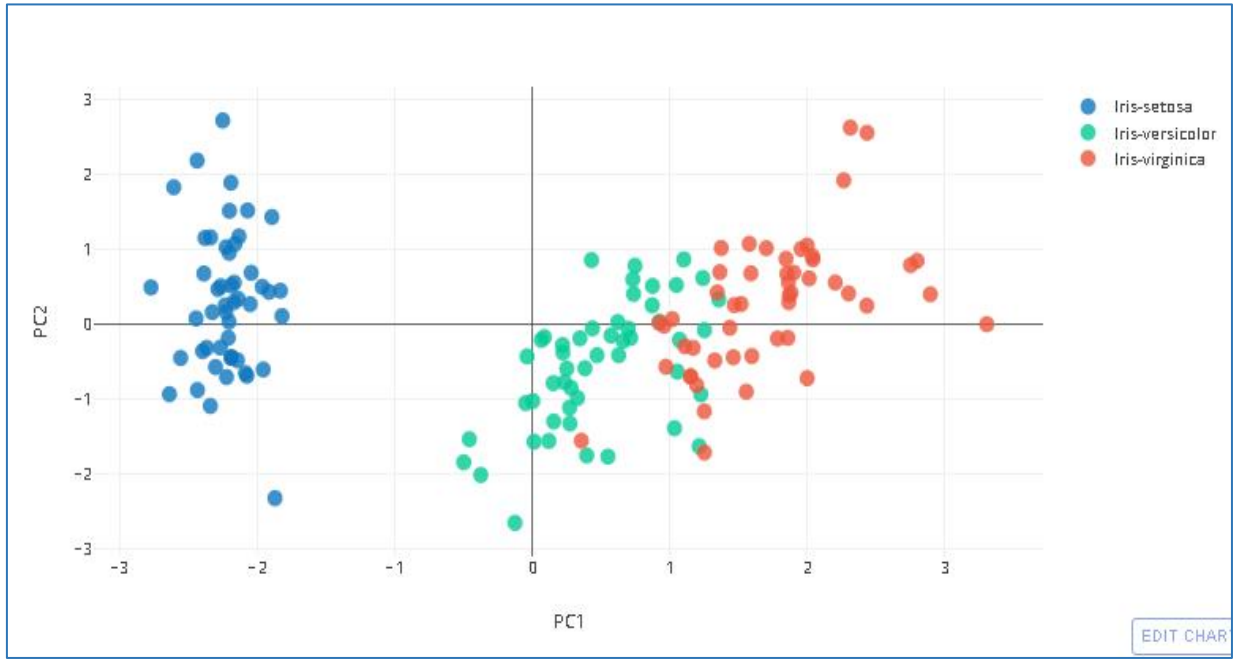
PCA in Python (Iris dataset)

Matrix_w:
[[0.5223 -0.3723]
 [-0.2633 -0.9255]
[0.5812 -0.0210]
 [0.5656 -0.0654]]

```
X_new = X_std.dot(matrix_w)
```



PCA 4d -> 2d



PCA in sklearn (Iris dataset)

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_sklearn = pca.fit_transform(X_std)
```

Attributes:

explained_variance_

The amount of variance explained by each of the selected components

explained_variance_ratio_

Percentage of variance explained by each of the selected components

```
pca = PCA(n_components=0.95)
```

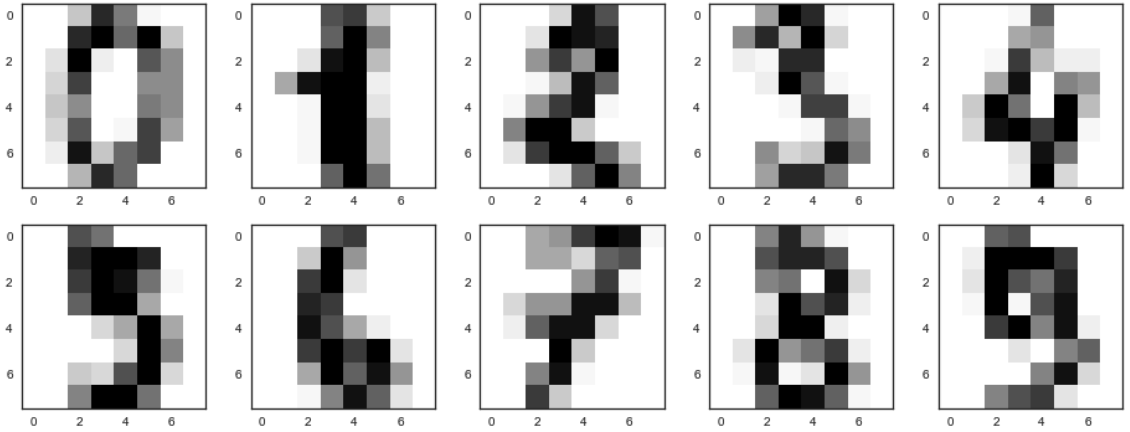
If 0 < n_components < 1, select the number of components such that the amount of variance is greater than the percentage specified by n_components.

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(max_depth=2)
clf.fit(X_train, y_train)
preds = clf.predict(X_test)
accuracy_score(y_test, preds)          0.88889
…
clf.fit(pca_X_train, y_train)  # 2 components
preds = clf.predict(pca_X_test)
accuracy_score(y_test, preds)          0.91111
```

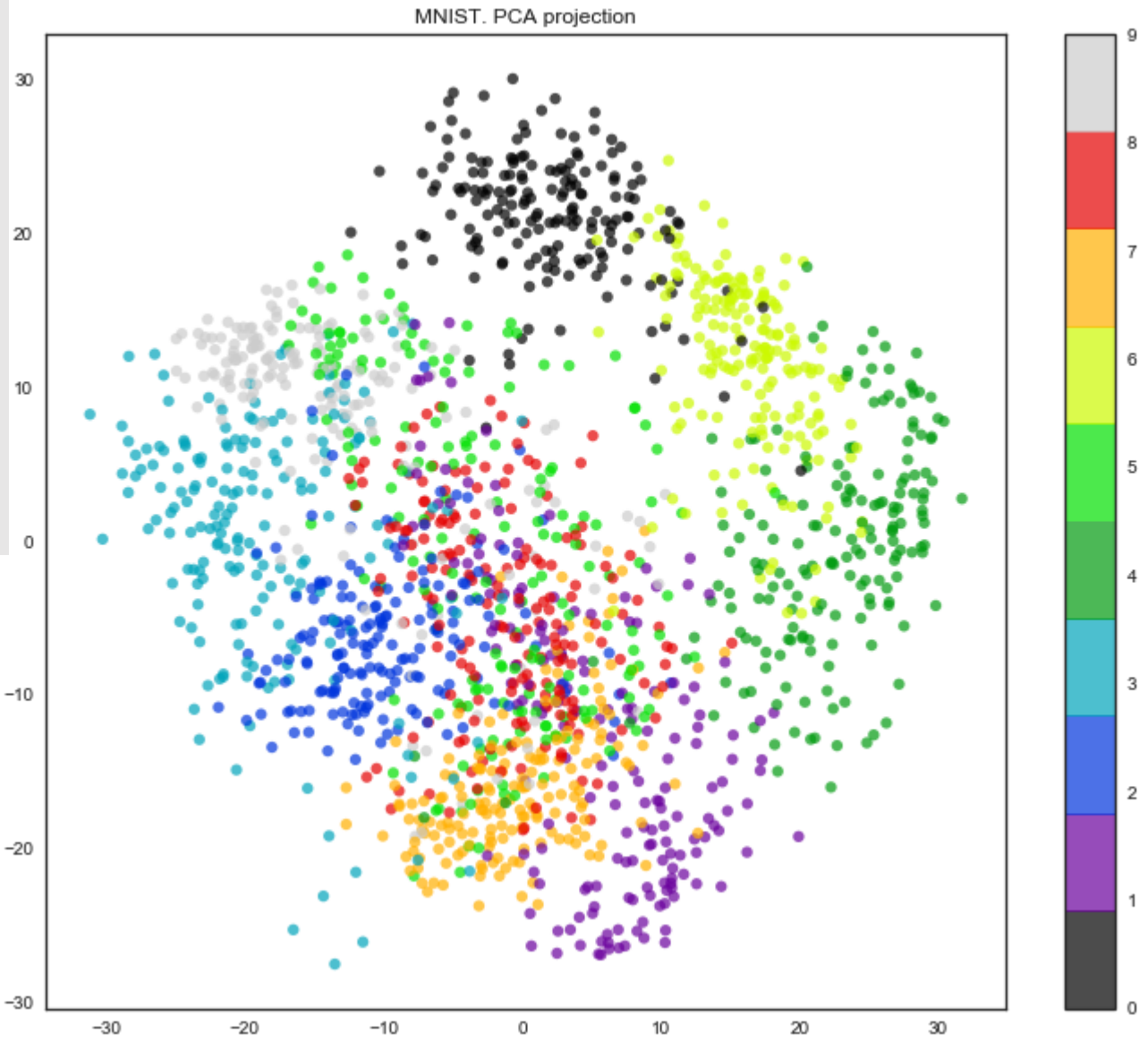
PCA in sklearn (MNIST dataset)

```
from sklearn import datasets
from sklearn.decomposition import PCA

digits = datasets.load_digits()
X = digits.data
y = digits.target
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```



8x8



Dimensionality reduction for data visualization

tSNE

t-distributed SNE (t-SNE)

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

$$\boxed{p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}}$$

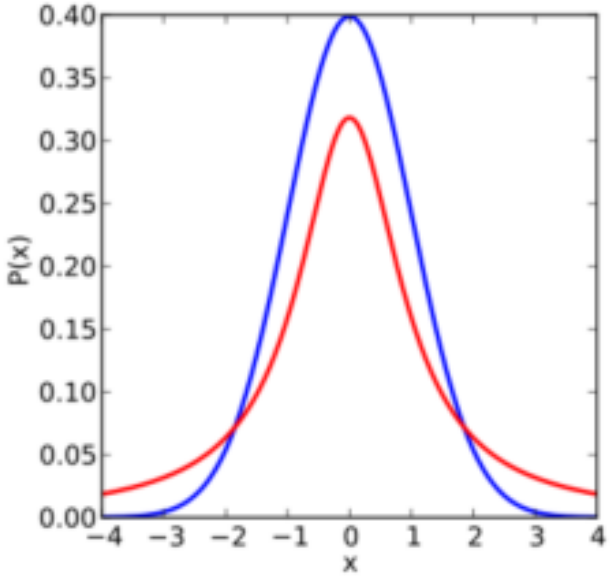
Probabilistic input neighborhood:
Probability to be picked as a neighbor in space X (input coordinates)

$$\boxed{q_{ij} = \frac{(1+||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1+||y_k - y_l||^2)^{-1}}}$$

Probabilistic output neighborhood:
Probability to be picked as a neighbor in space Y (display coordinates)

$$\text{Cost} = KL(P \,||\, Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

Visualizing Data using t-SNE, 2008, L.Maaten&G.Hinton



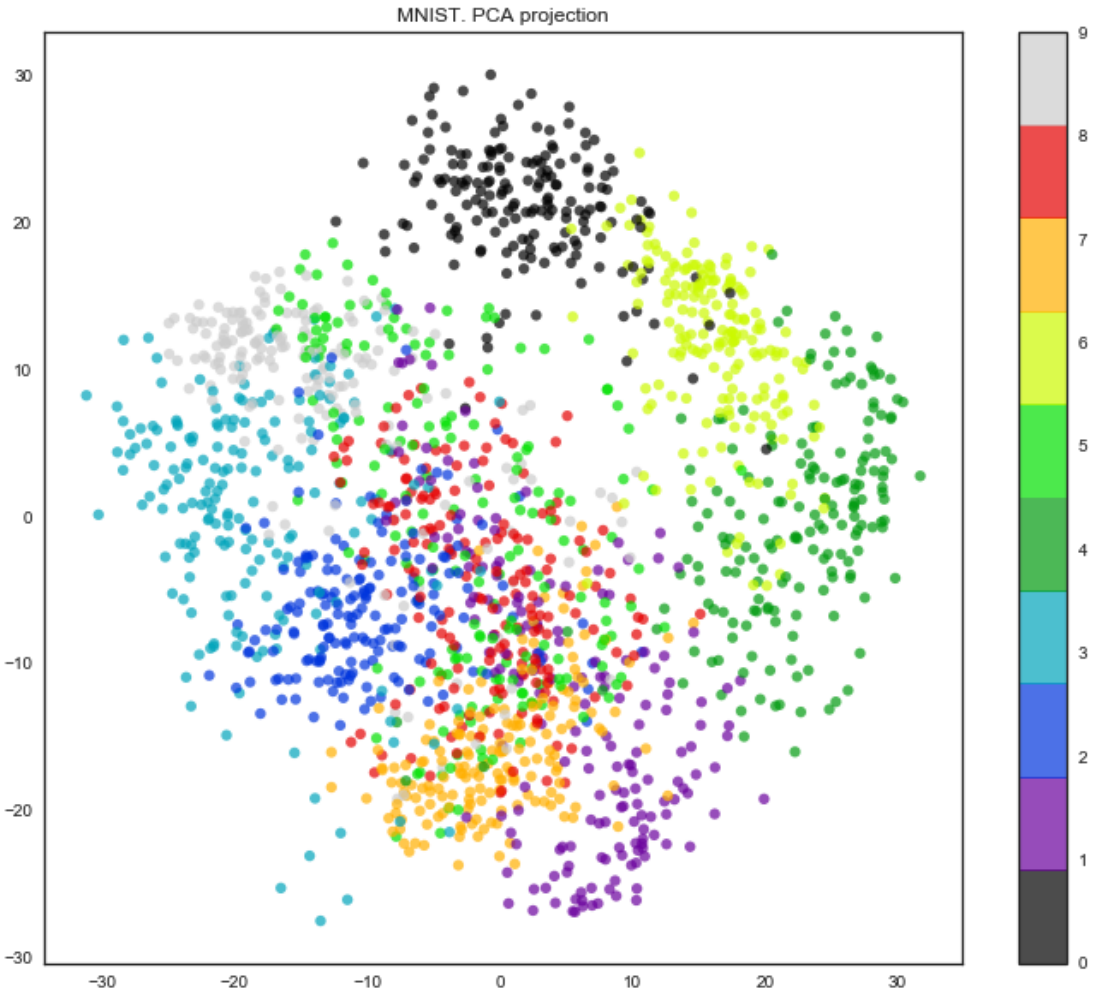
Blue: normal distribution
Red: t-distribution (heavy-tailed)

http://www.sci.utah.edu/~beiwang/teaching/cs6965-spring-2018/Lecture02-DR.pdf

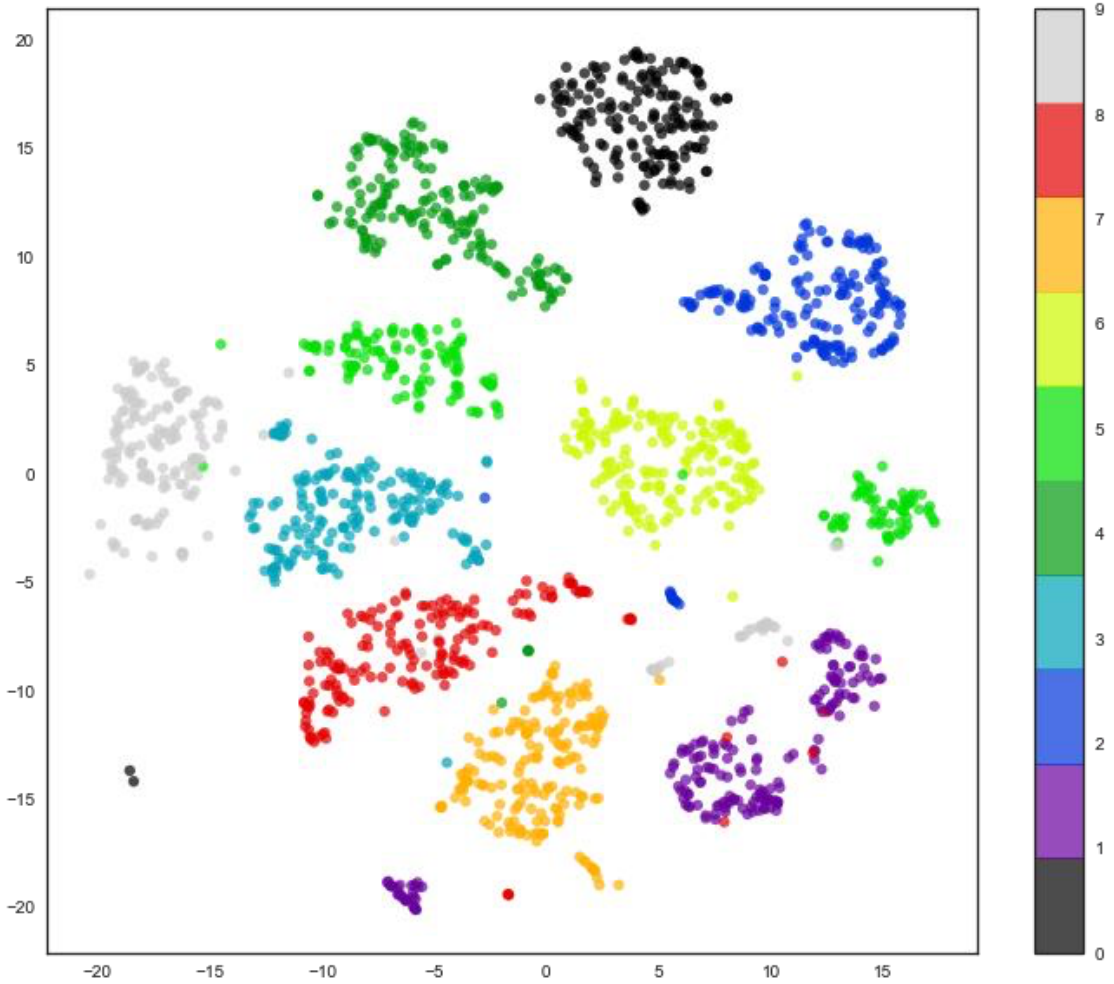
MNIST (8x8) Example

digits = sklearn.datasets.load_digits()

sklearn.manifold.**TSNE**(*n_components=2)*



PCA



t-SNE