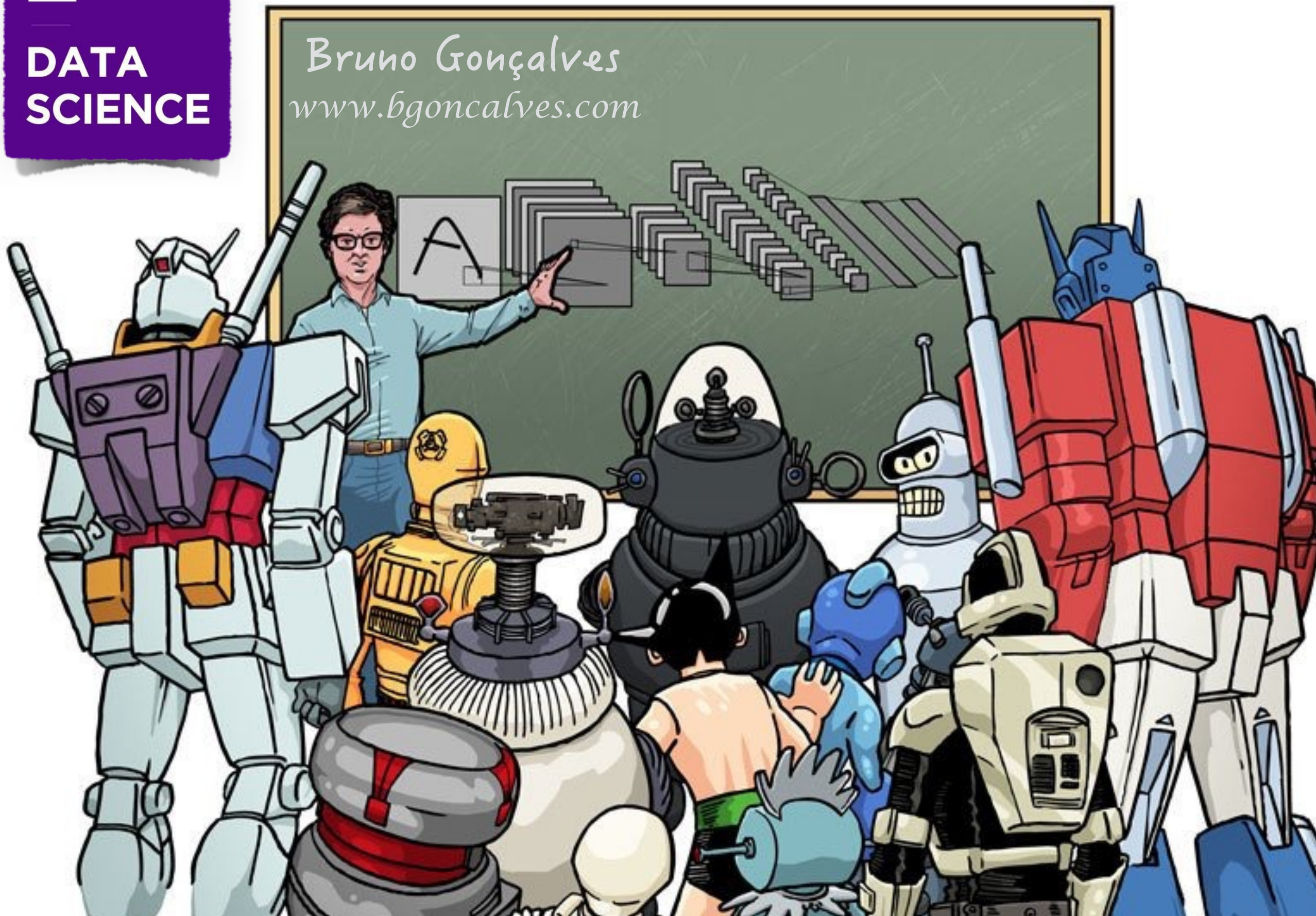
NYU

DATA SCIENCE

Machine(s) Learning and Data Science

Bruno Gonçalves
www.bgoncalves.com



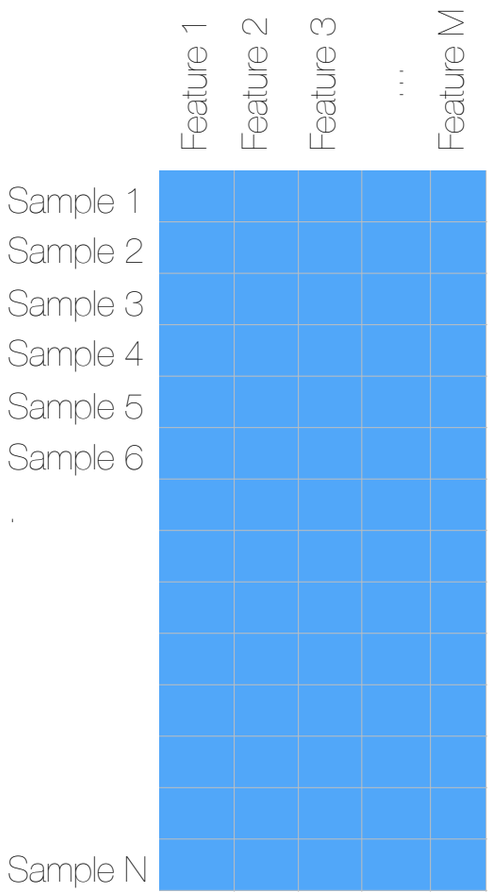
3 Types of Machine Learning

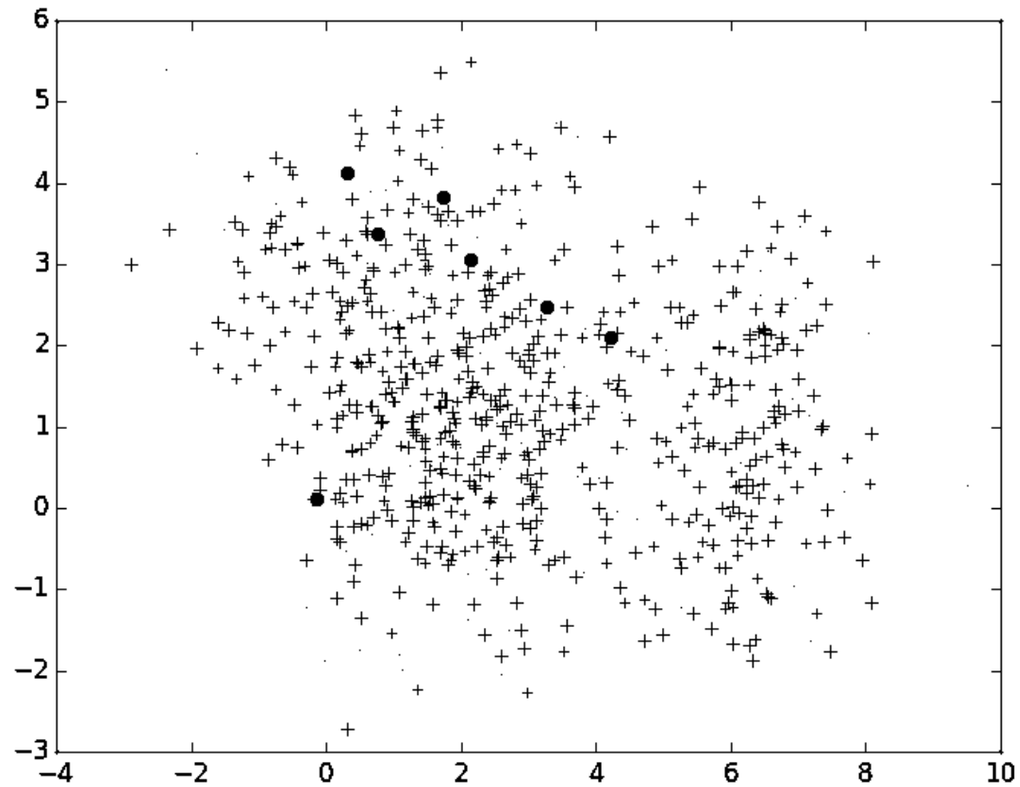
- Unsupervised Learning
  - Autonomously learn an good representation of the dataset
  - Find clusters in input
- Supervised Learning
  - Predict output given input
  - Training set of known inputs and outputs is provided
- ~~Reinforcement Learning~~
  - Learn sequence of actions to maximize payoff
  - Discount factor for delayed rewards

Unsupervised Learning

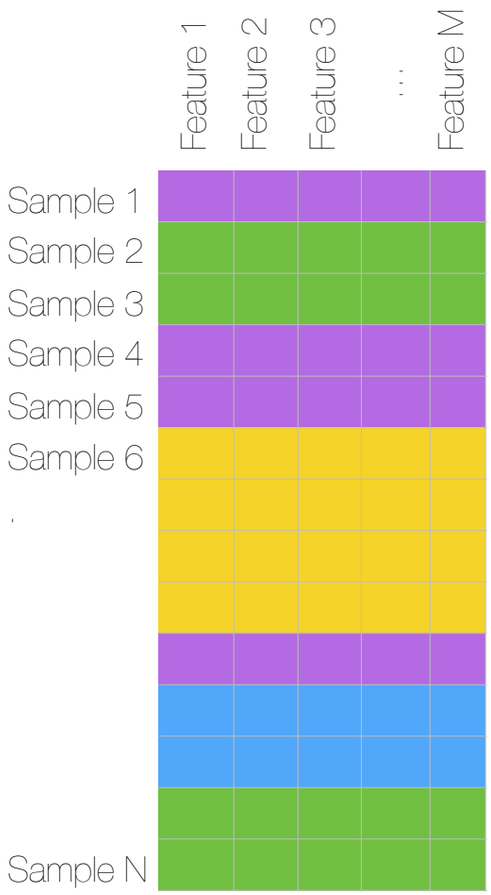
- Extracting patters from data
  - K-Means
  - Expectation Maximization
  - Gaussian Mixture Models
  - PCA

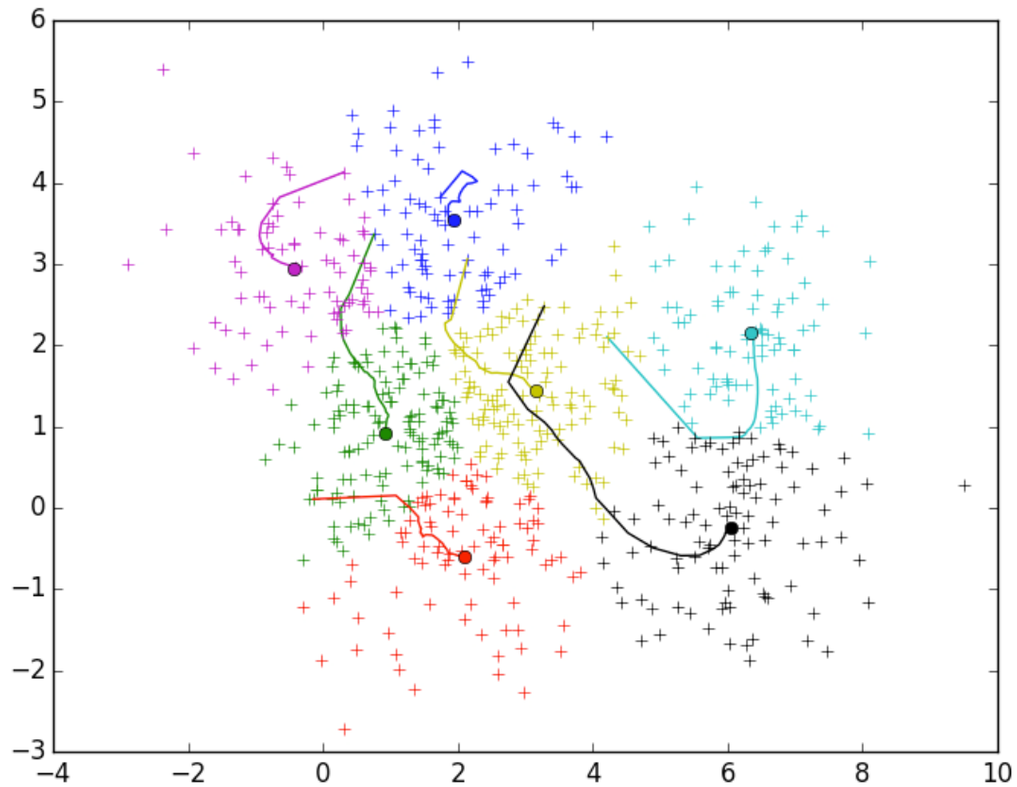
Clustering


Feature 1
Feature 2
Feature 3
. . .
Feature M
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
Sample 6
Sample N


6
5
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
8
10

Clustering

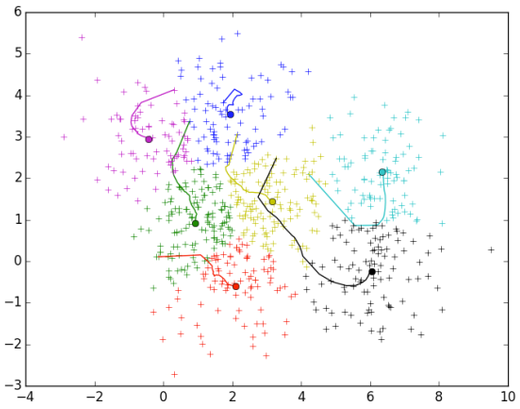

Feature 1
Feature 2
Feature 3
. . .
Feature M
Sample 1
Sample 2
Sample 3
Sample 4
Sample 5
Sample 6
.
Sample N


6
5
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6
8
10

Data Normalization

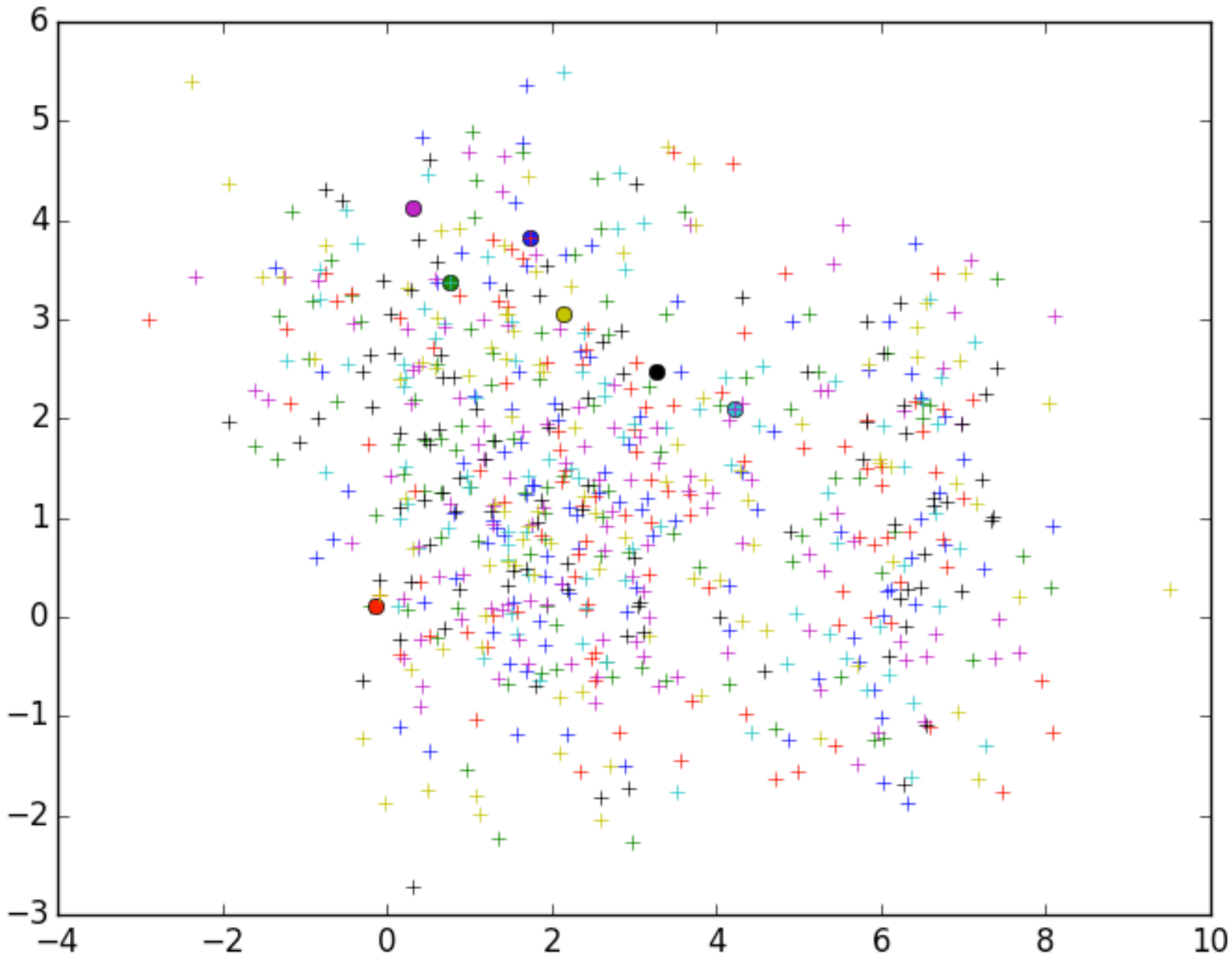
- The range of raw data values can vary widely.
- Using feature with very different ranges in the same analysis can cause numerical problems. Many algorithms are linear or use euclidean distances that are heavily influenced by the numerical values used (cm vs km, for example)
- To avoid difficulties, it's common to rescale the range of all features in such a way that each feature follows within the same range.
- Several possibilities:
  - Rescaling - $\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$
  - Standardization - $\hat{x} = \frac{x - \mu_x}{\sigma_x}$
  - Normalization - $\hat{x} = \frac{x}{||x||}$
  - In the rest of the discussion we will assume that the data has been normalized in some suitable way

K-Means



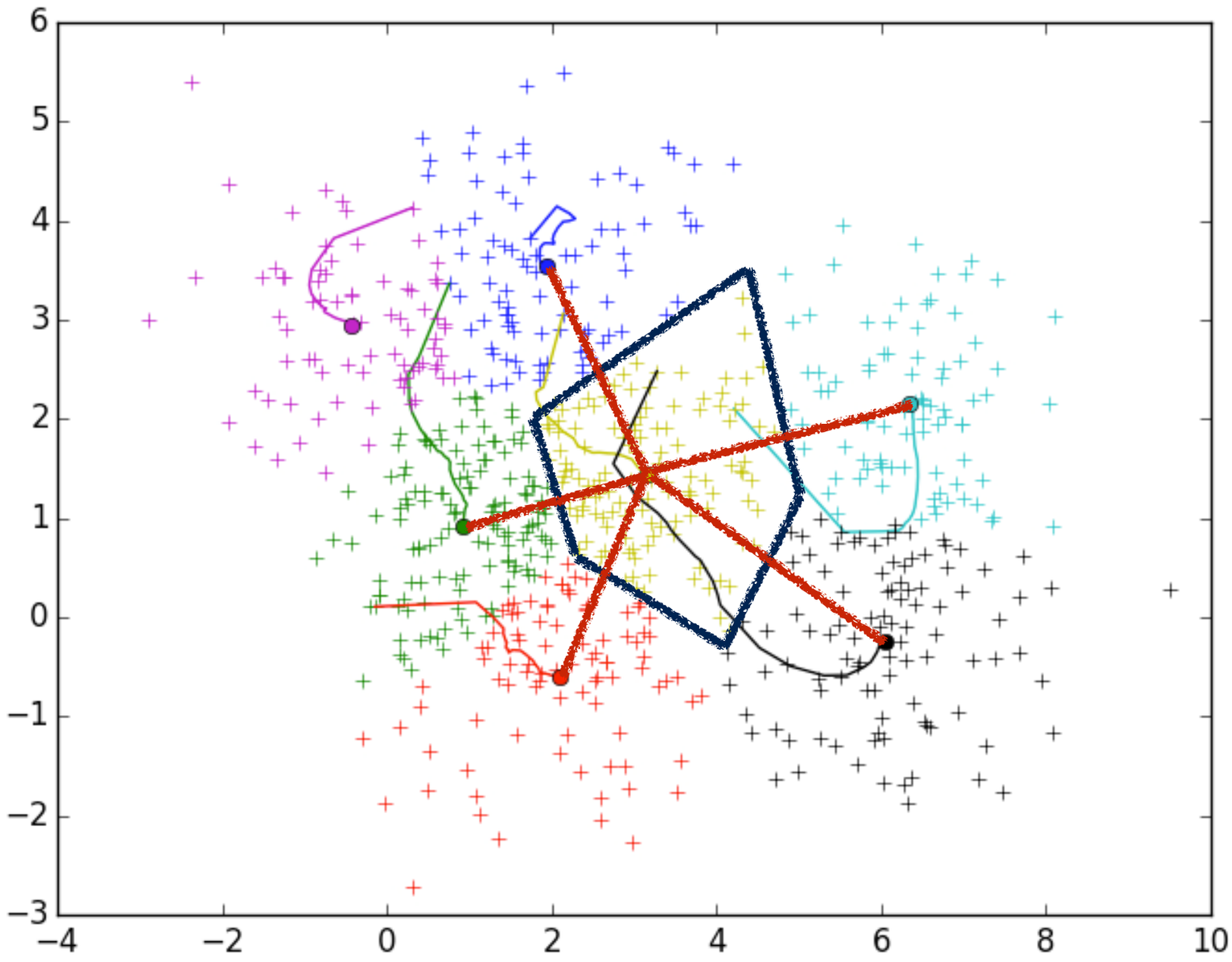
- Choose k randomly chosen points to be the centroid of each cluster
- Assign each point to belong the cluster whose centroid is closest
- Recompute the centroid positions (mean cluster position)
- Repeat until convergence

K-Means



K-Means: Structure

Voronoi Tesselation



K-Means: Convergence

- How to quantify the "quality" of the solution found at each iteration, n?

- Measure the "Inertia", the square intra-cluster distance:

$$I_n = \sum_{i=0}^{N} \|x_i - \mu_i\|^2$$

where μ_i are the coordinates of the centroid of the cluster to which x_i is assigned.

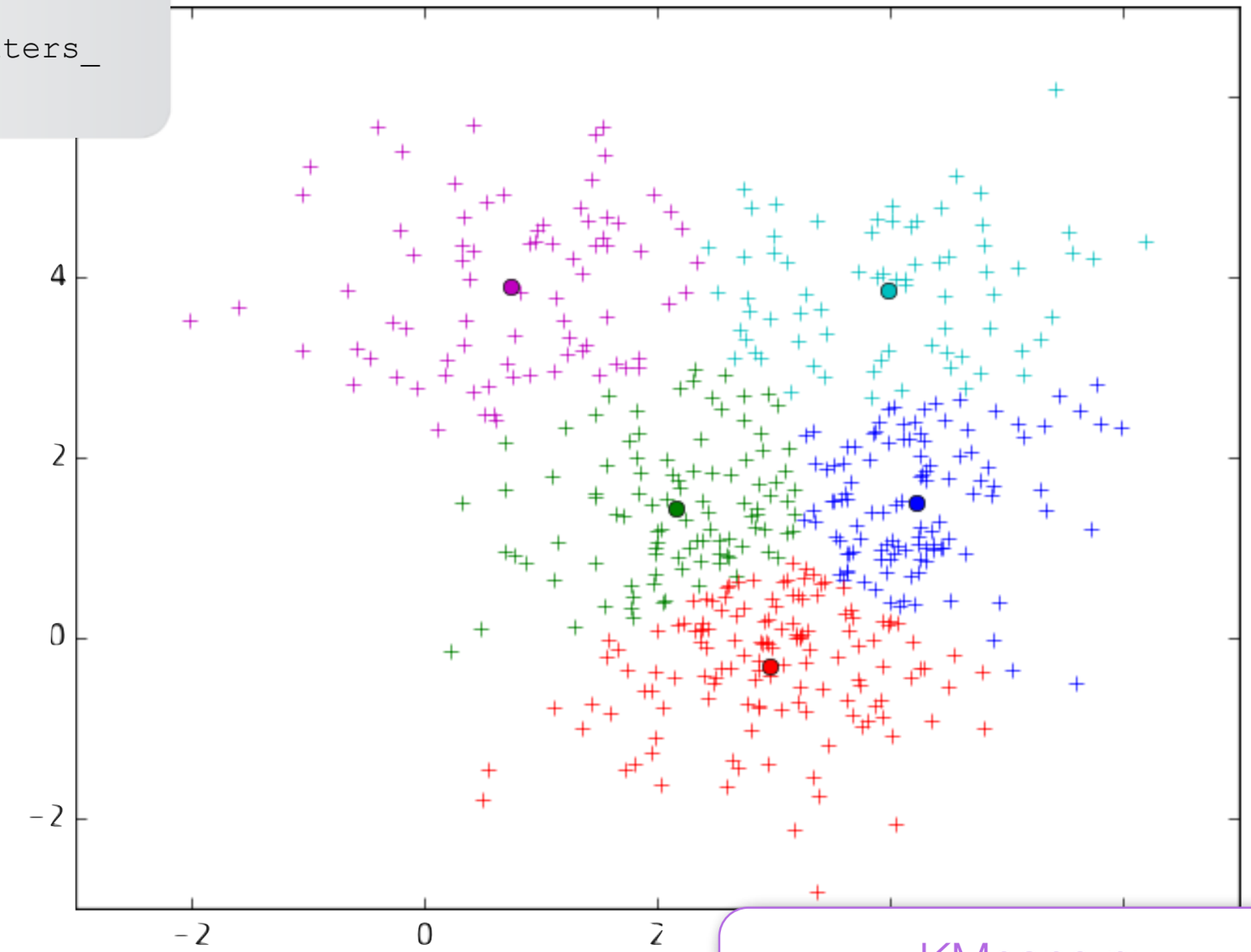
- Smaller values are better

- Can stop when the relative variation is smaller than some value

$$\frac{I_{n+1} - I_n}{I_n} < tol$$

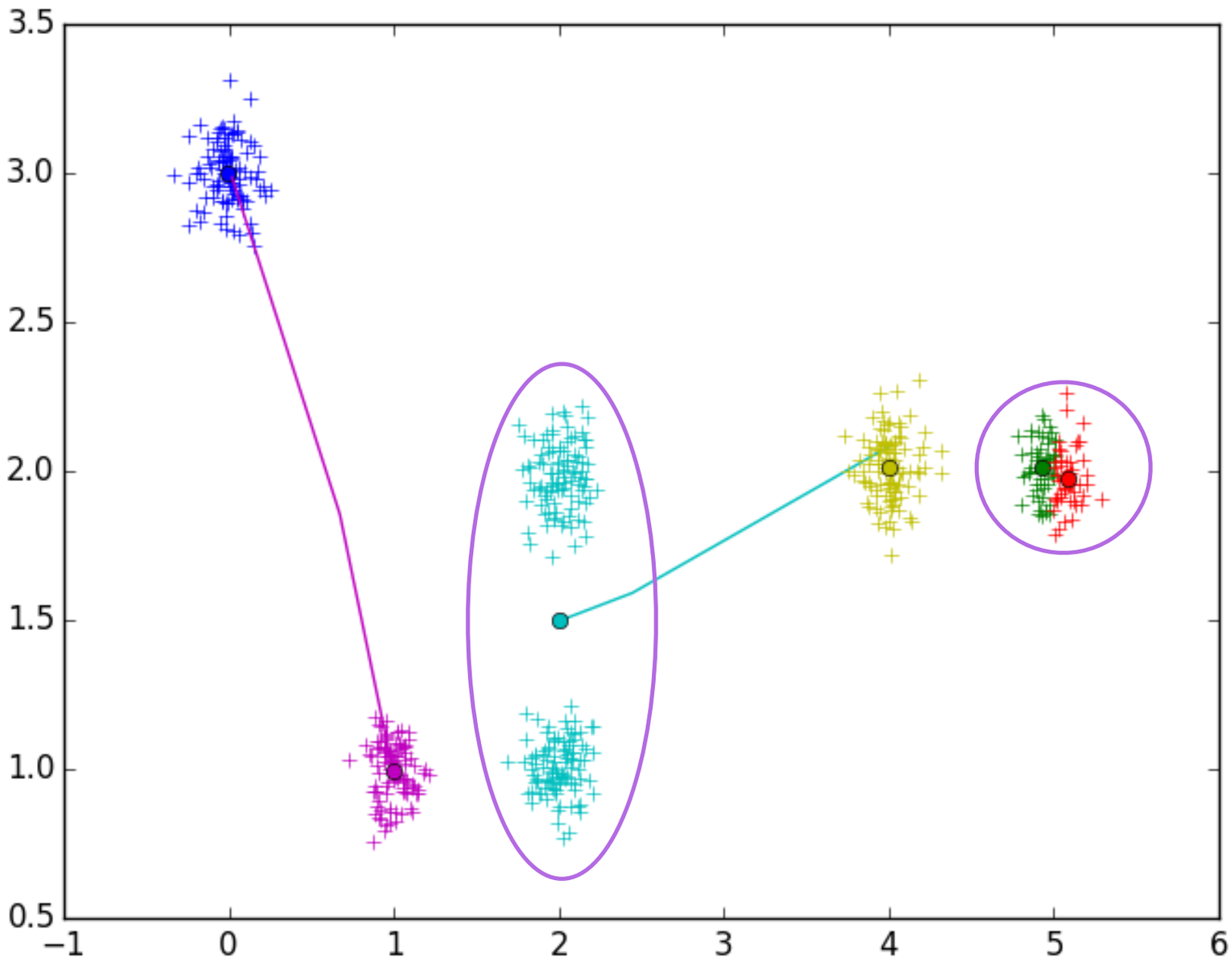
K-Means: sklearn

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=nclusters)
kmeans.fit(data)

centroids = kmeans.cluster_centers_
labels = kmeans.labels_
```

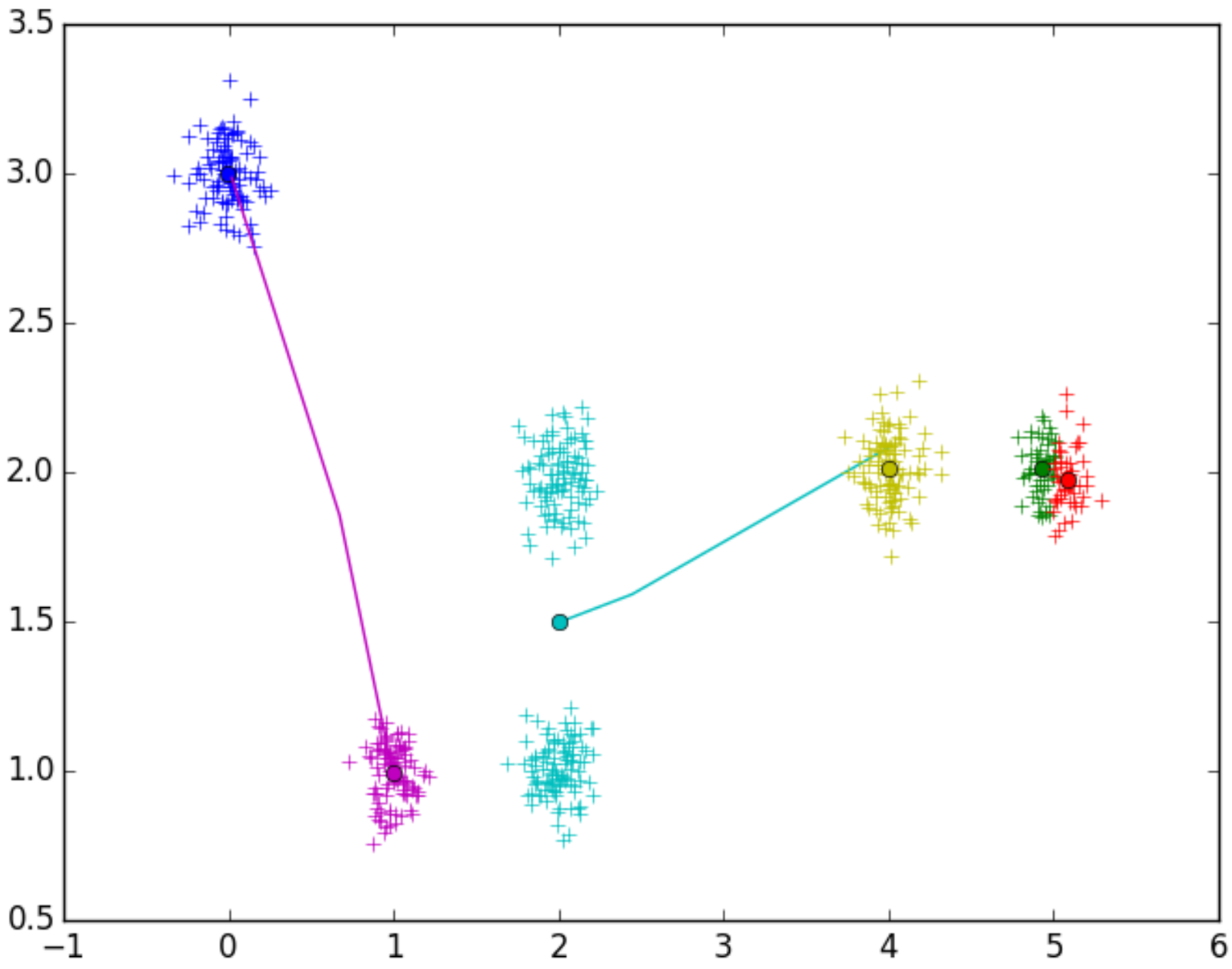


KMeans.py

K-Means: Limitations



K-Means: Limitations



- No guarantees about Finding "Best" solution
- Each run can find different solution
- No clear way to determine "k"

Silhouettes

- For each point x_i define $a_c\left(x_i\right)$ as:

$$a_c\left(x_i\right) = \frac{1}{N_c}\sum_{j \in c}\left\|x_i - x_j\right\|$$

the average distance between point x_i and every other point within cluster c

- Let $b\left(x_i\right)$ be:

$$b\left(x_i\right) = \min_{c \neq c_i} a_c\left(x_i\right)$$

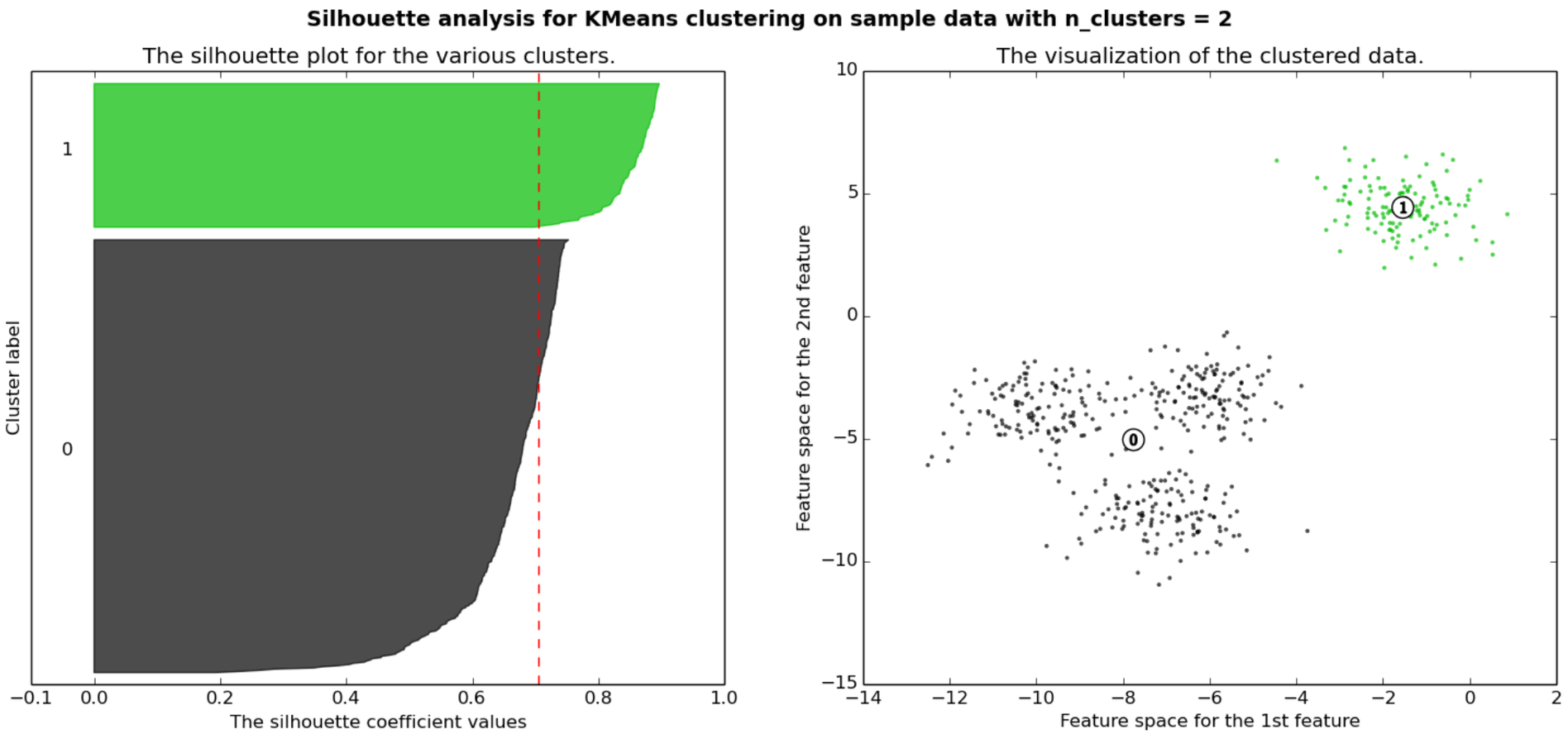
the minimum value of $a_c\left(x_i\right)$ excluding c_i

- The silhouette of x_i is then:

$$s\left(x_i\right) = \frac{b\left(x_i\right) - a_{c_i}\left(x_i\right)}{\max\left\{b\left(x_i\right), a_{c_i}\left(x_i\right)\right\}}$$

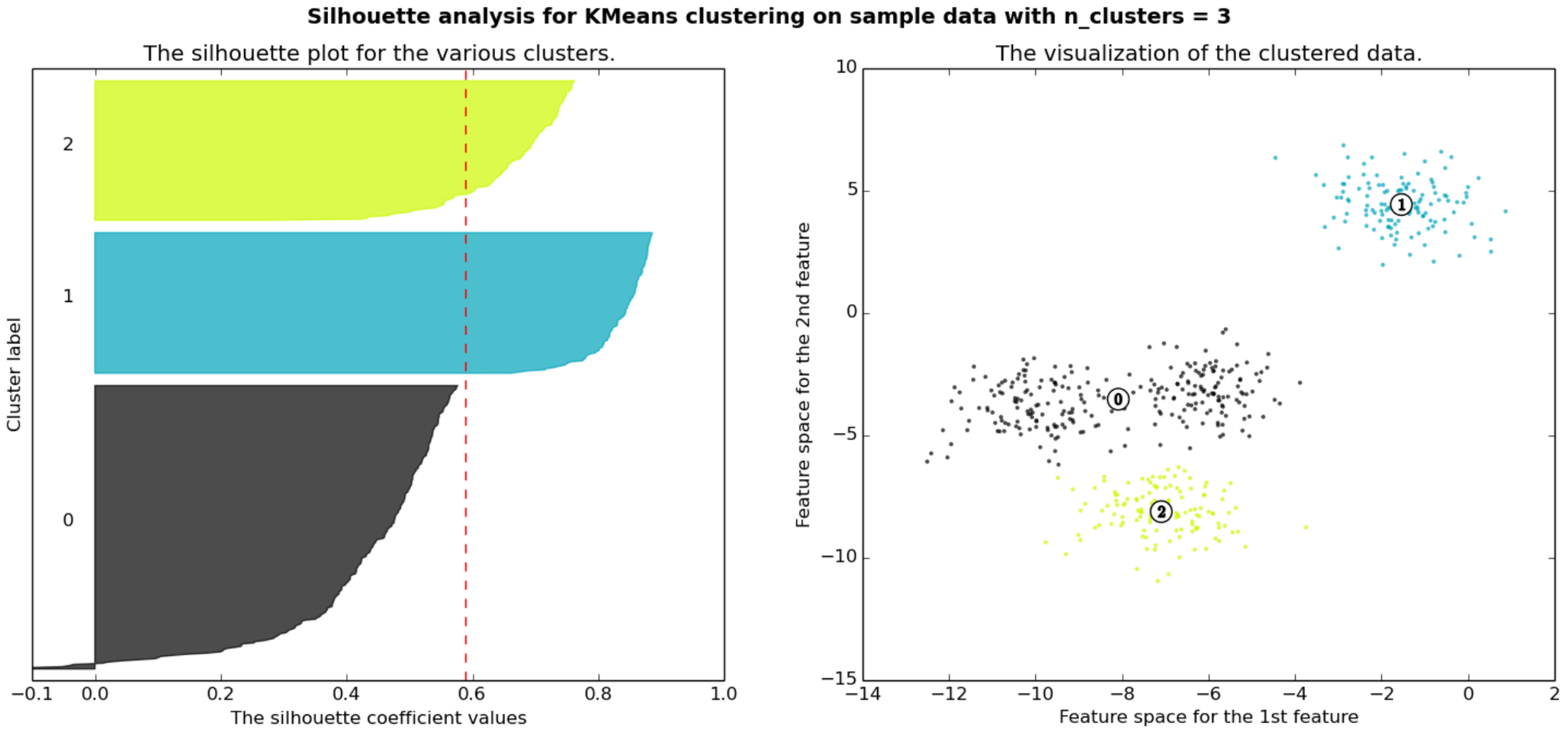
Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



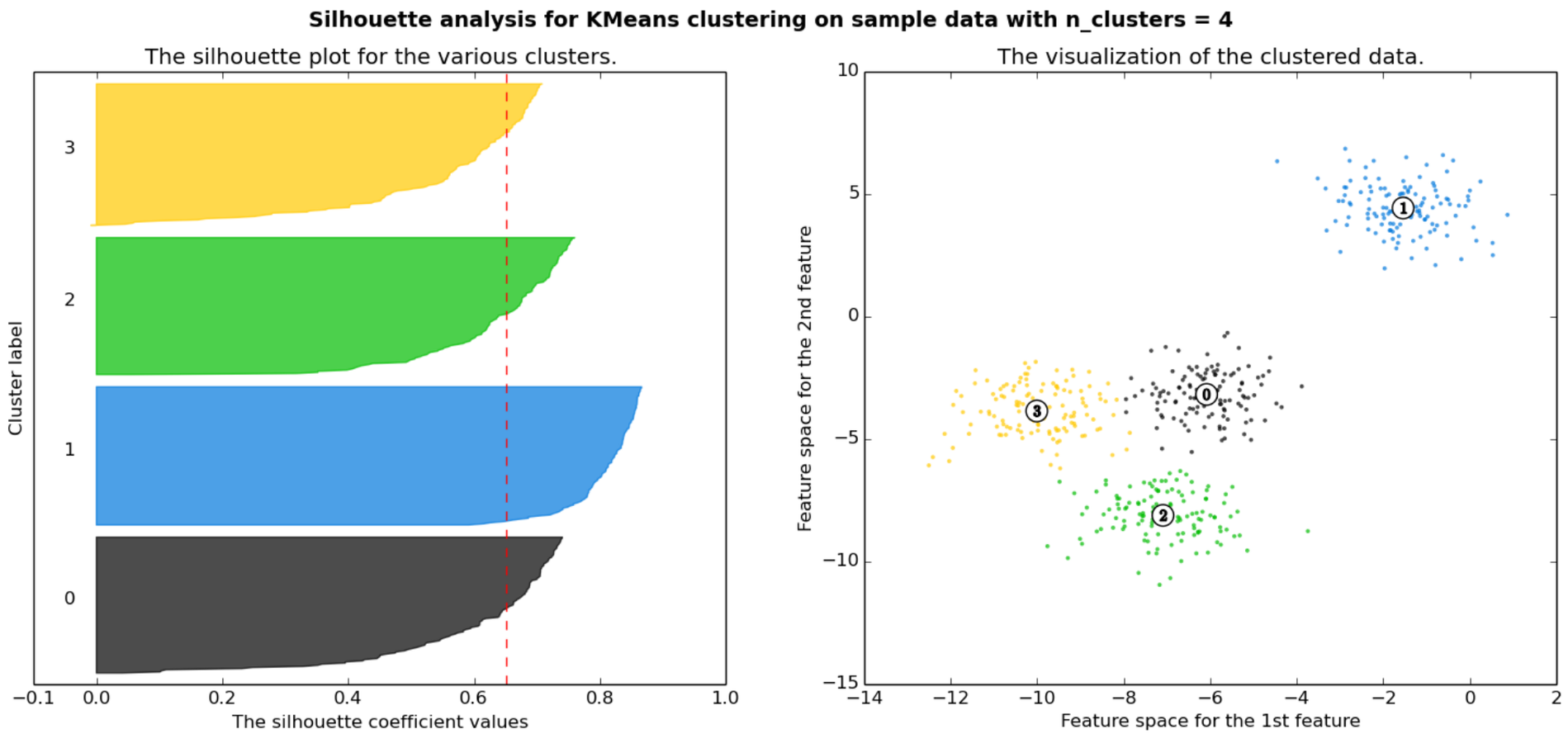
Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



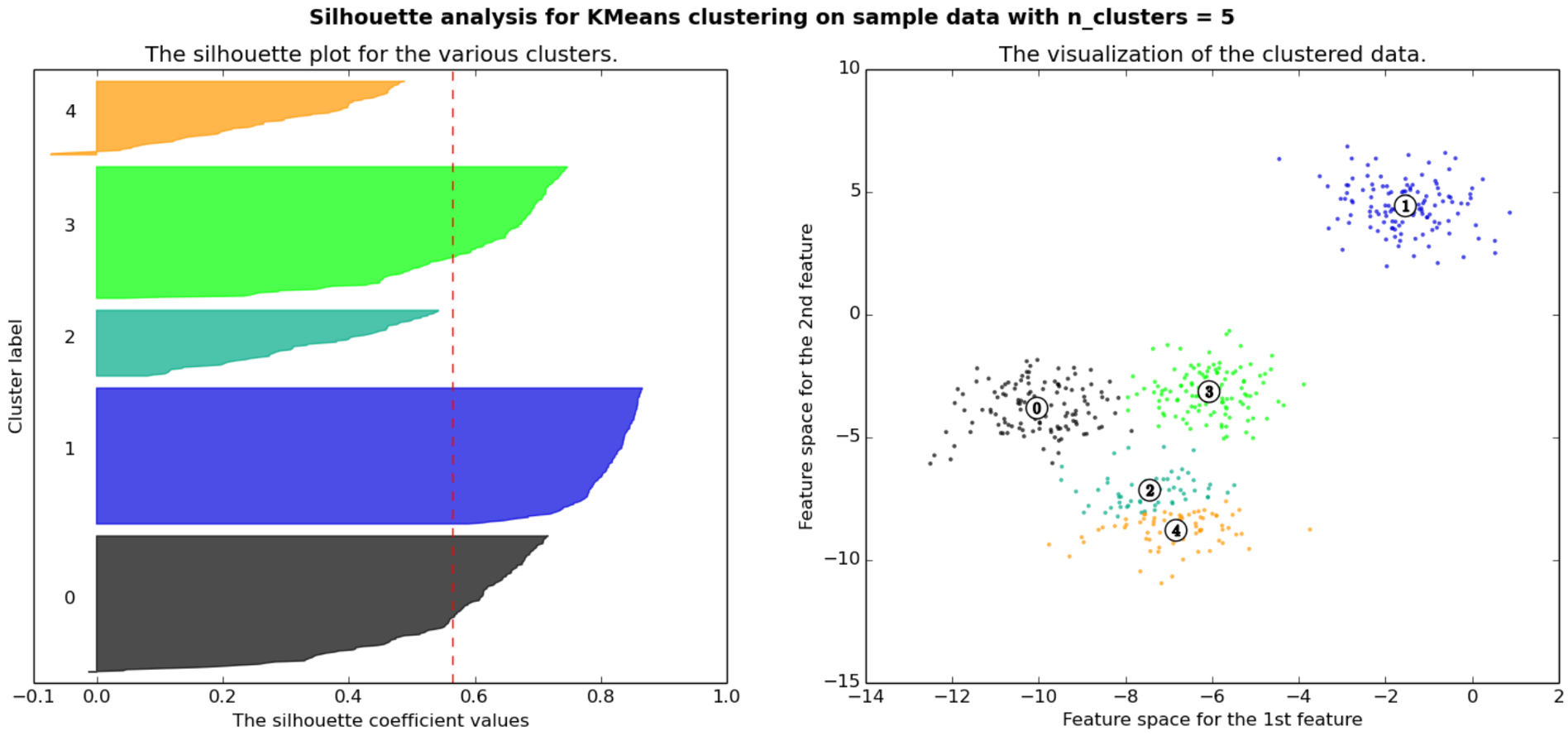
Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



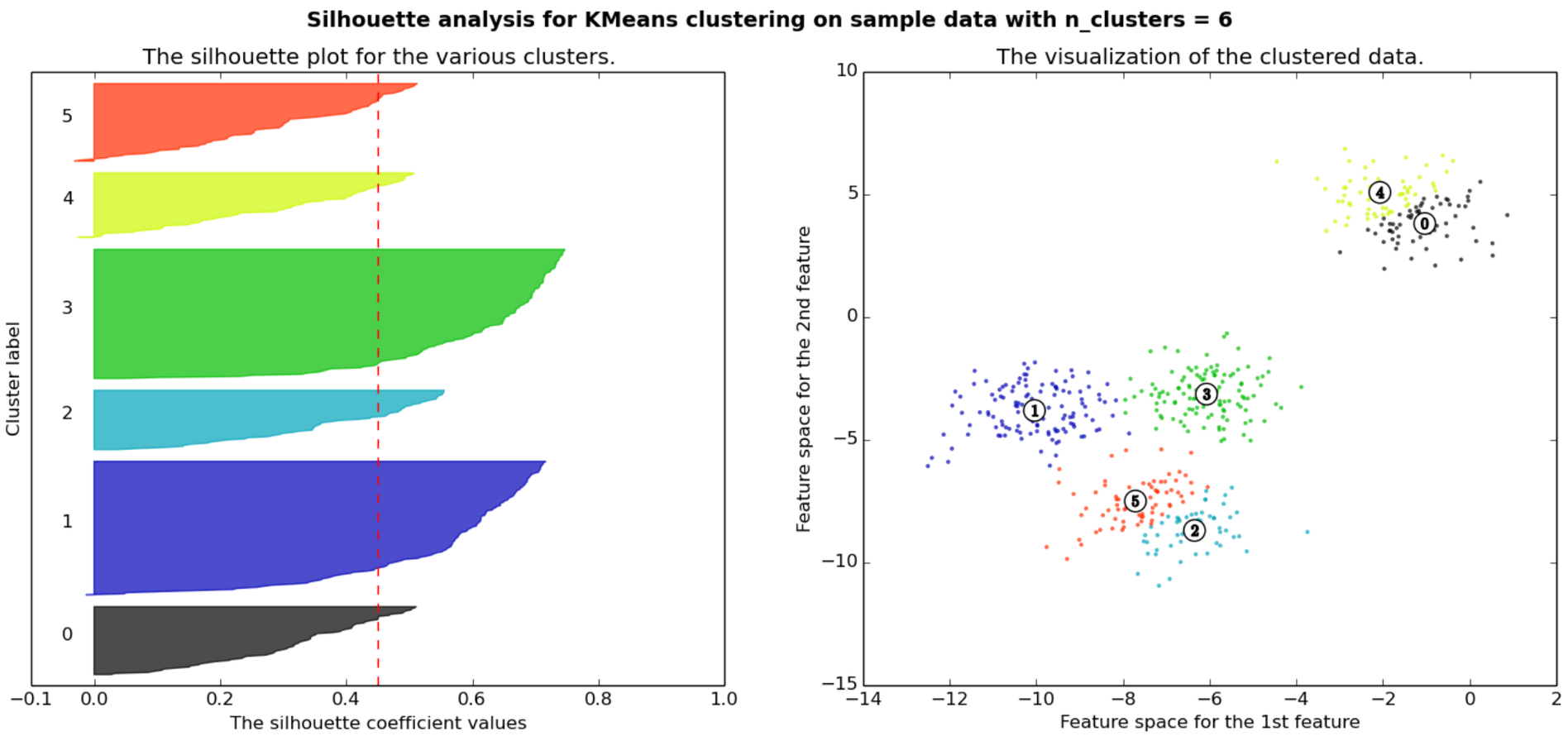
Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



Silhouettes

http://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



Expectation Maximization

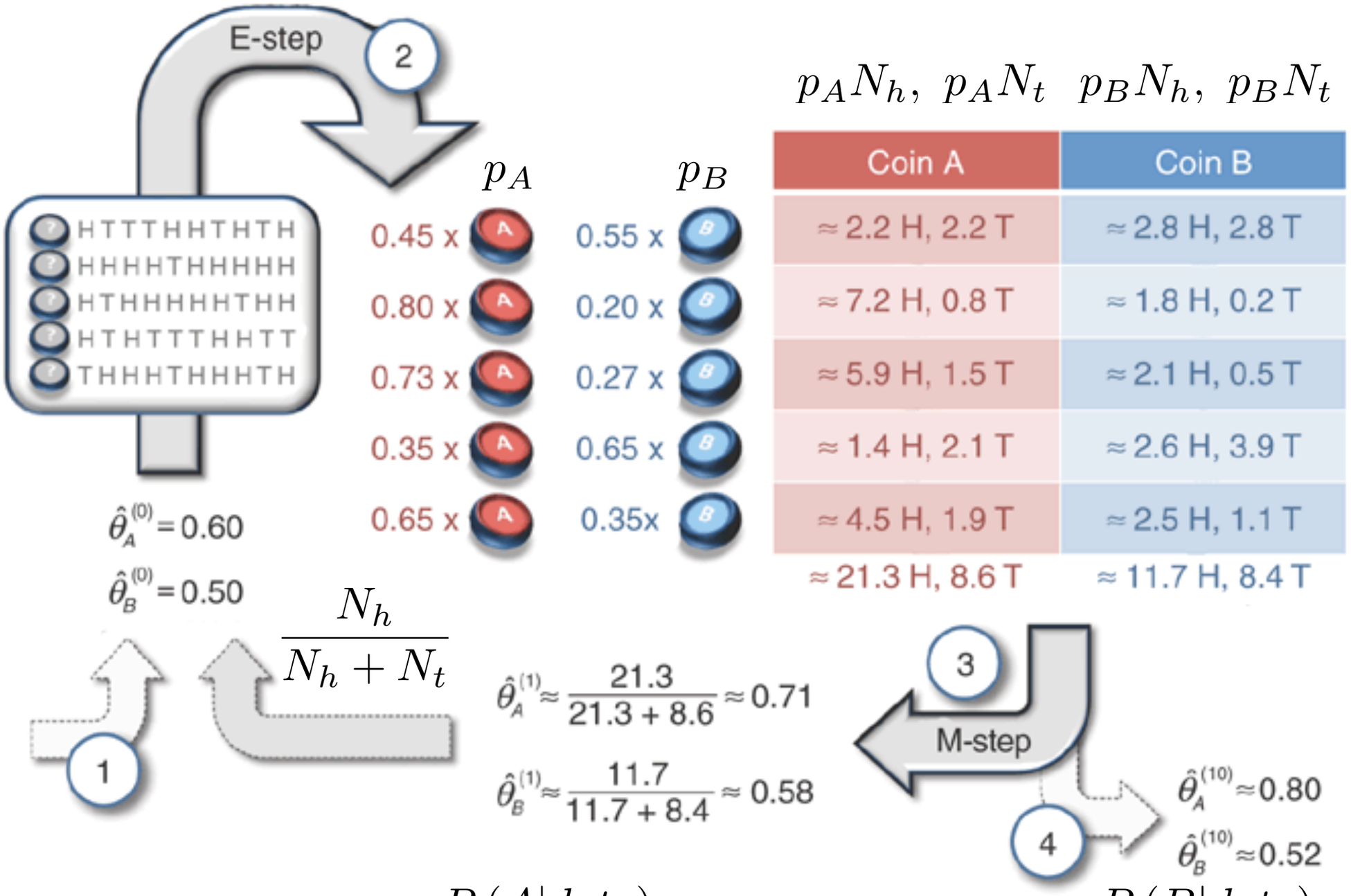
- Iterative algorithm to learn parameter estimates in models with unobserved latent variables

- Two steps for each iteration
  - **Expectation:** Calculate the likelihood of the data given current parameter estimate
  - **Maximization:** Find the parameter values that maximize the likelihood

- Stop when the relative variation of the parameter estimates is smaller than some value

Expectation Maximization

Nature BioTech 26, 897 (2008)



@bgoncalves

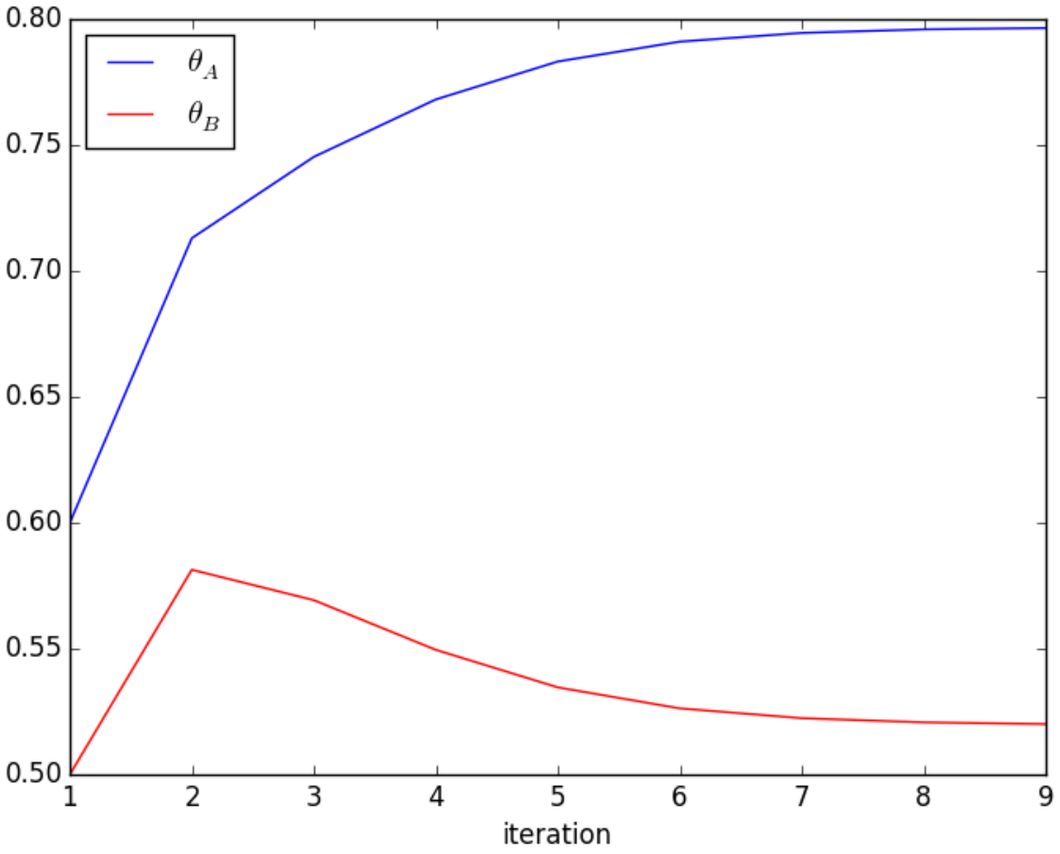
$$p_A = \frac{P(A|data)}{P(A|data) + P(B|data)} \qquad p_B = \frac{P(B|data)}{P(A|data) + P(B|data)}$$

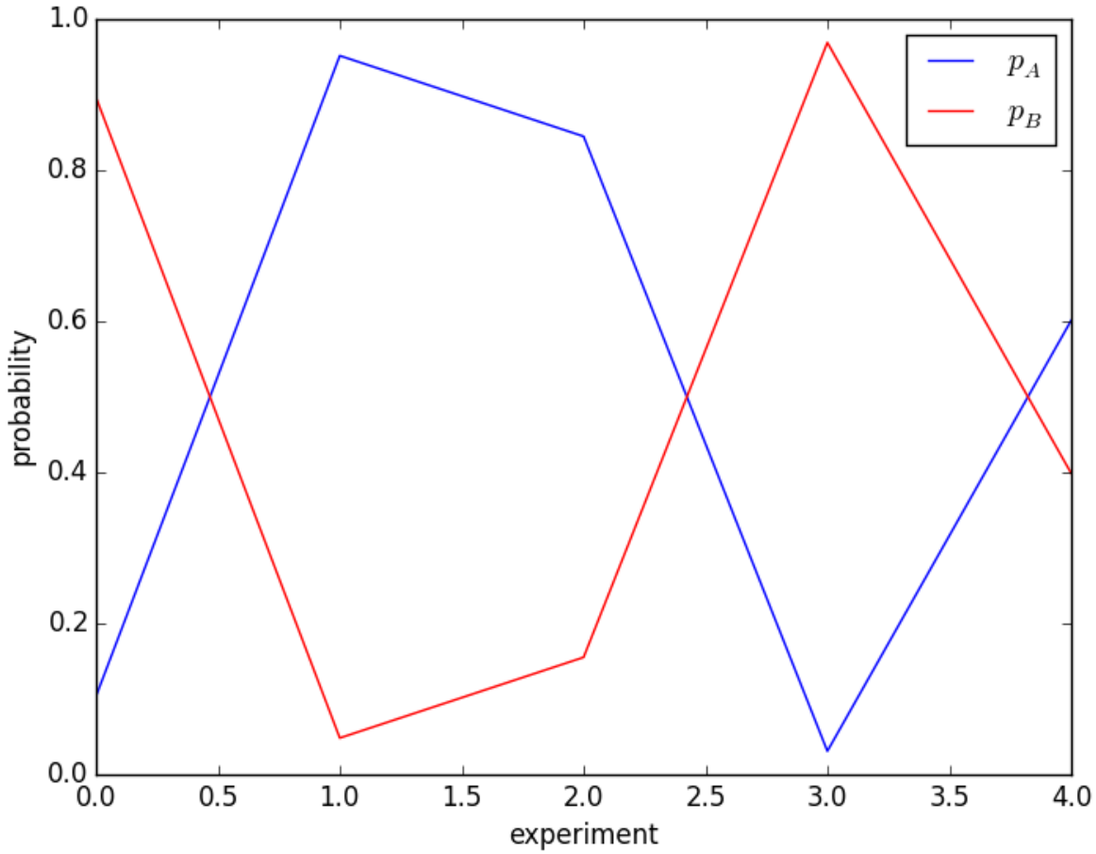
Expectation Maximization

```
while (improvement > delta):
    expectation_A = np.zeros((5, 2), dtype=float)
    expectation_B = np.zeros((5, 2), dtype=float)

    for i in range(0, len(experiments)):
        e = experiments[i]  # i'th experiment
        ll_A = get_mn_likelihood(e, np.array([tA[-1], 1-tA[-1]]))
        ll_B = get_mn_likelihood(e, np.array([tB[-1], 1-tB[-1]]))

        weightA = ll_A/(ll_A + ll_B)
        weightB = ll_B/(ll_A + ll_B)

        expectation_A[i] = np.dot(weightA, e)
        expectation_B[i] = np.dot(weightB, e)

    tA.append(sum(expectation_A)[0] / sum(sum(expectation_A)))
    tB.append(sum(expectation_B)[0] / sum(sum(expectation_B)))

    improvement = max(abs(np.array([tA[-1], tB[-1]]) - np.array([tA[-2], tB[-2]])))
```

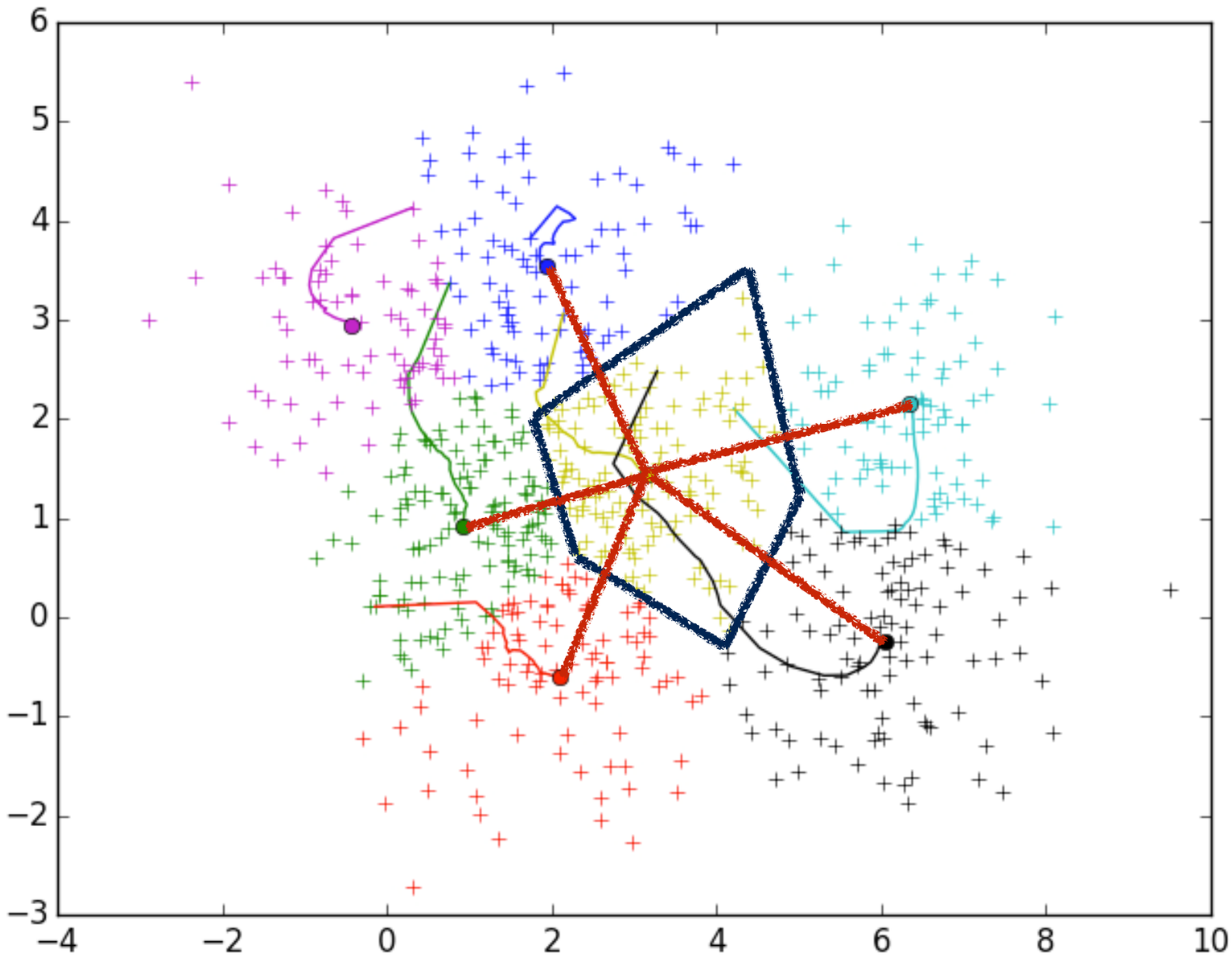

EM.py

Expectation Maximization





Gaussian Mixture Models



Gaussian Mixture Models

- One solution is to try to characterize each cluster as a Gaussian. In this case we want to find the set of parameters and mixtures that better reproduces the data.

- Given some data points x_i we can calculate the prior:

$$p(\theta) = \sum_i \phi_i \mathcal{N}(\mu_i, \sigma_i)$$

- which we can update using the data: $p(x|\theta)$ to obtain the posterior:

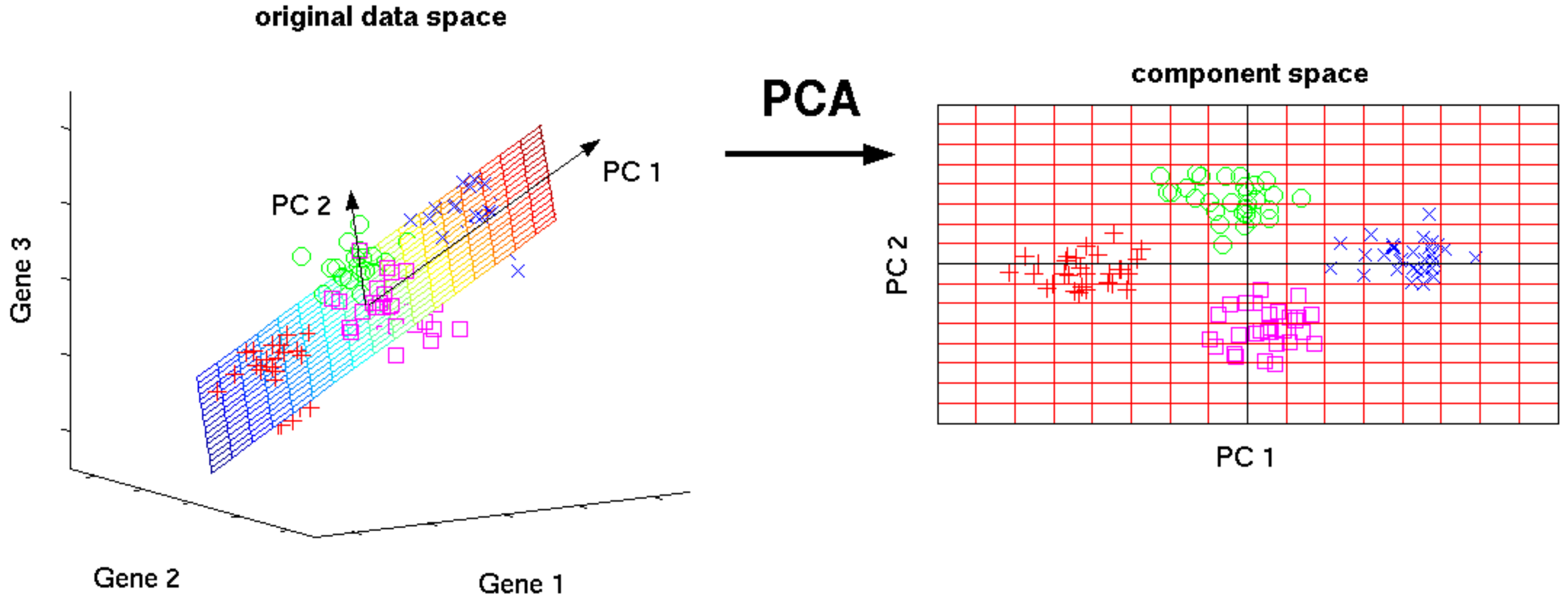
$$p(\theta|x) = \frac{p(x|\theta)\, p(\theta)}{p(x)}$$

- which we can use to choose a new set of parameters and mixtures.

- Iterate using Expectation Maximization.

Principle Component Analysis

- Finds the directions of maximum variance of the dataset
- Useful for dimensionality reduction
- Often used as preprocessing of the dataset



Principle Component Analysis

- The Principle Component projection, T, of a matrix A is defined as:

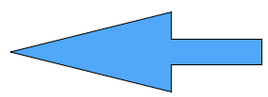
$$T = AW$$

- where W is the eigenvector matrix of:

$$A^T A$$

- and corresponds to the right singular vectors of A obtained by Singular Value Decomposition (SVD):

$$A = U\Sigma W^T$$



Generalization of Eigenvalue/ Eigenvector decomposition for non-square matrices.

- So we can write:

$$T = U\Sigma W^T W \equiv U\Sigma$$

- Showing that the Principle Component projection corresponds to the left singular vectors of A scaled by the respective singular values Σ

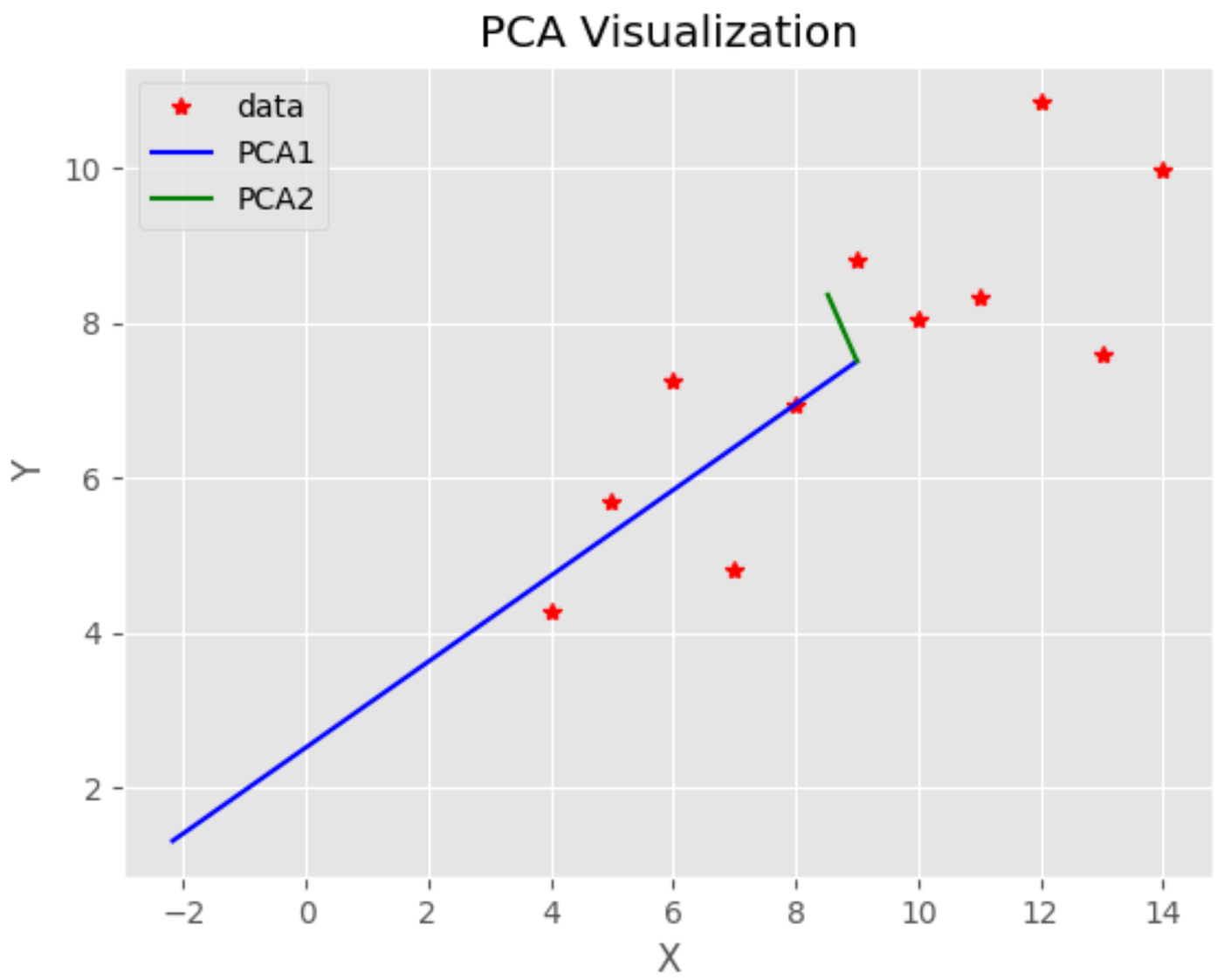
- Columns of T are ordered in order of decreasing variance.

Princ

```python
import sys
from sklearn.decomposition import PCA
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt(sys.argv[1])

x = data.T[0]
y = data.T[1]

pca = PCA()
pca.fit(data)

meanX = np.mean(x)
meanY = np.mean(y)

plt.style.use('ggplot')
plt.plot(x, y, 'r*')

plt.plot([meanX, meanX+pca.components_[0][0]*pca.explained_variance_[0]],
         [meanY, meanY+pca.components_[0][1]*pca.explained_variance_[0]], 'b-')
plt.plot([meanX, meanX+pca.components_[1][0]*pca.explained_variance_[1]],
         [meanY, meanY+pca.components_[1][1]*pca.explained_variance_[1]], 'g-')
plt.title('PCA Visualization')
plt.legend(['data', 'PCA1', 'PCA2'], loc=2)
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('PCA.png')
plt.close()

transform = pca.transform(data)
plt.plot(transform.T[0], transform.T[1], 'r*')
plt.title('PCA Transform Visualization')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.savefig('PCATransform.png')
plt.close()
```

PCA.py

Principle Component Analysis



Principle Component Analysis