Lecture 05:

Unsupervised Learning

Markus Hohle

University California, Berkeley

Machine Learning Algorithms

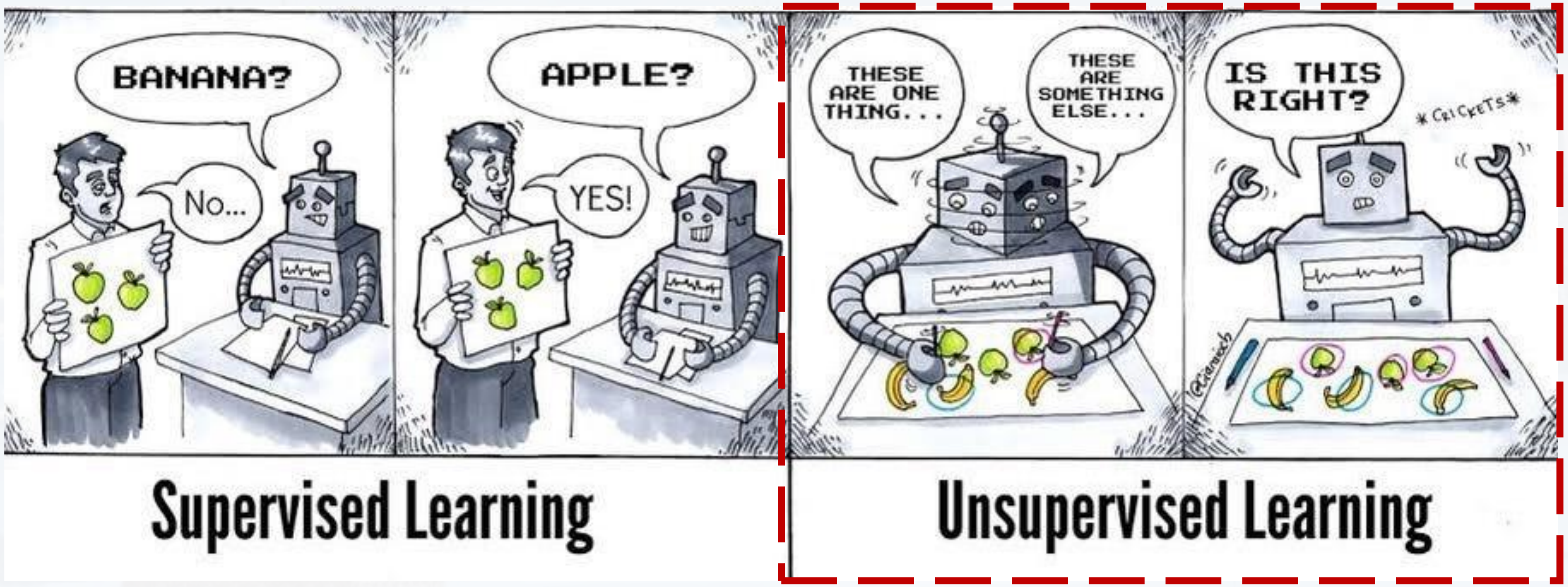
MSSE 277B, 3 Units

Spring 2025

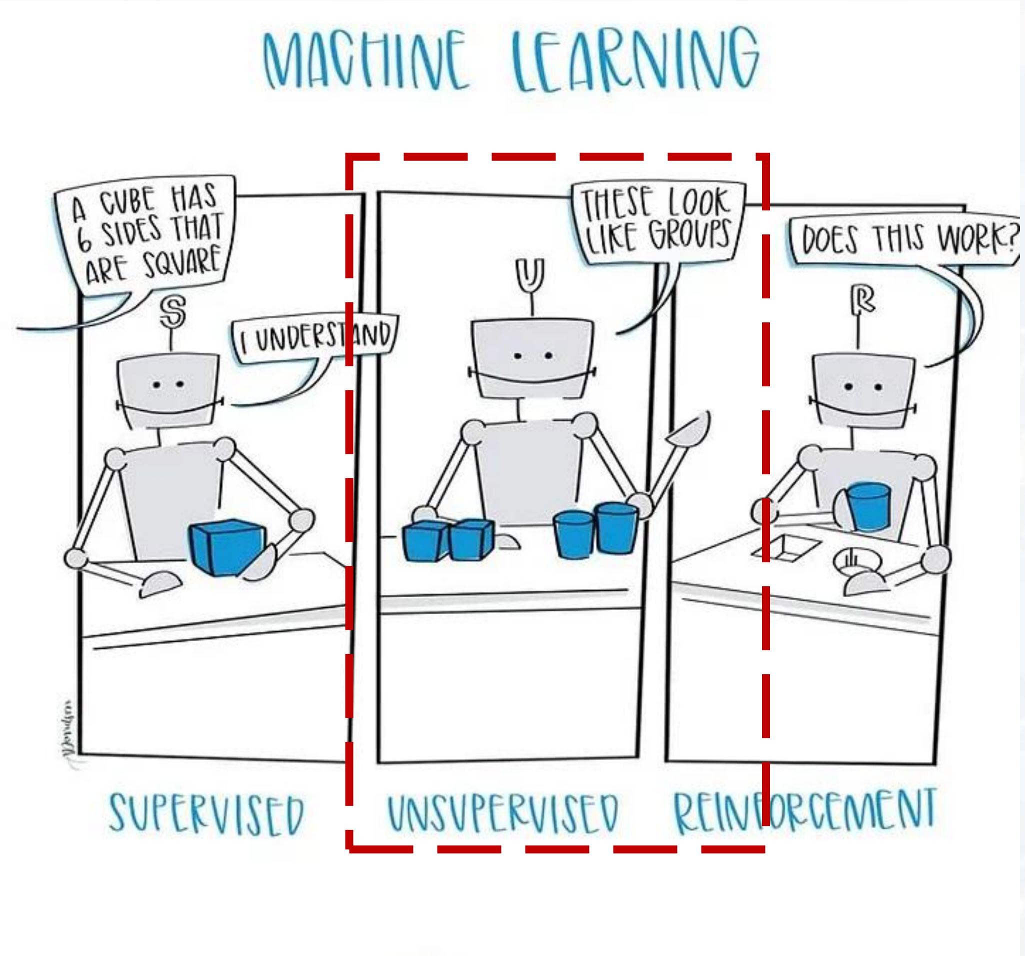
Machine Learning Algorithms:

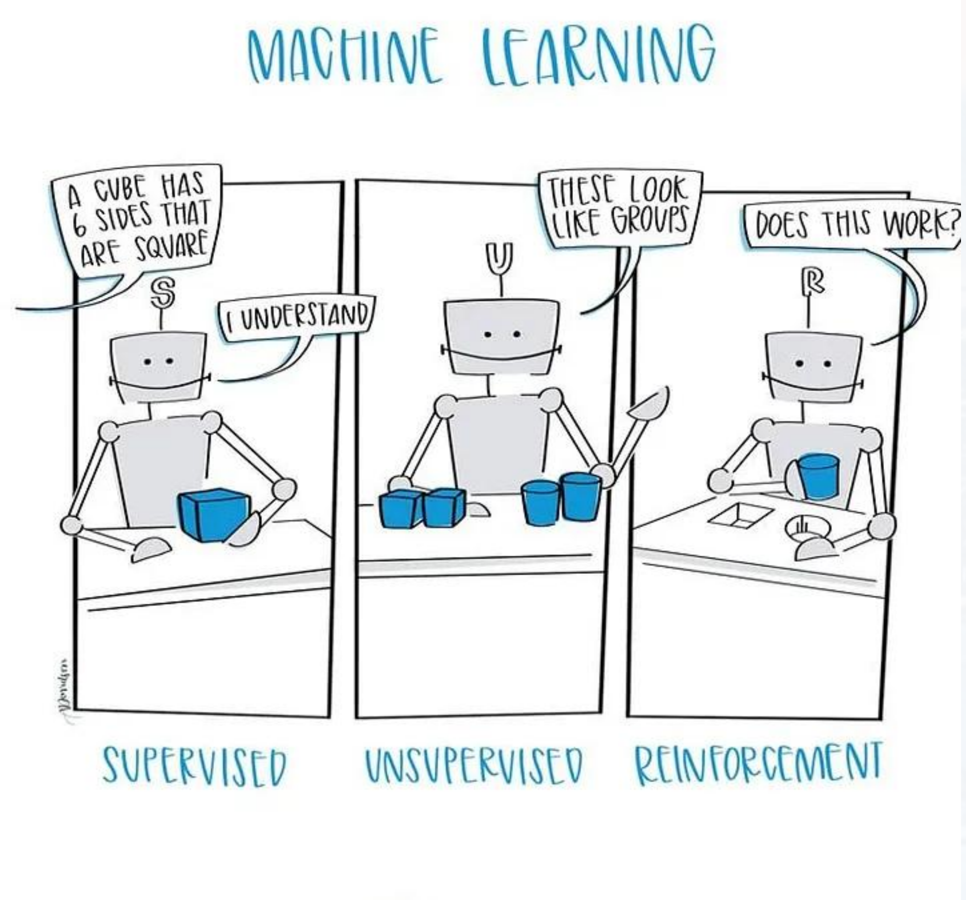
So far, there has been a **training** data set and a **test** data set…

… but maybe there are ways to learn *without* training data



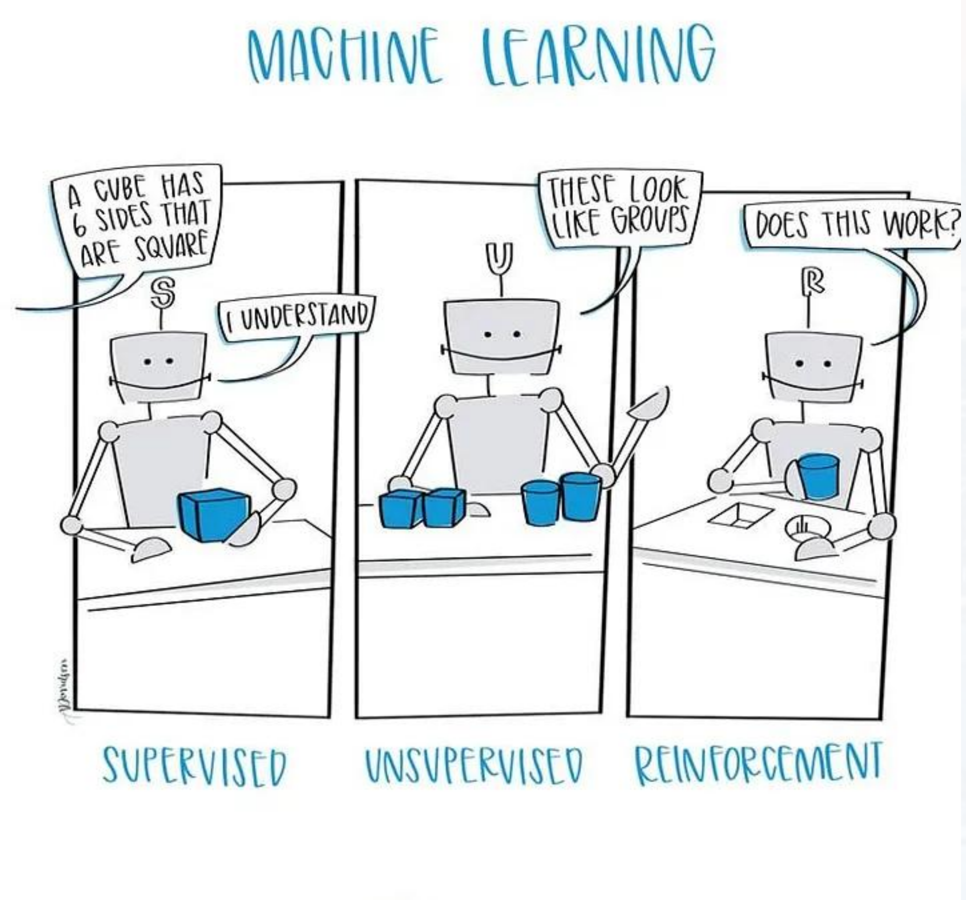
So far, there has been a **training** data set and a **test** data set…
… but maybe there are ways to learn *without* training data





Outline

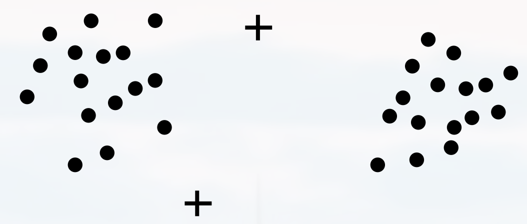
- K - means
- GMM
- trees



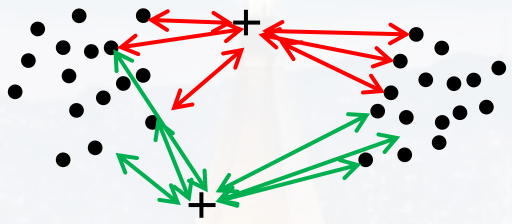
Outline

- **K - means**
- GMM
- trees

<u>idea:</u>



a) assign *k* means randomly



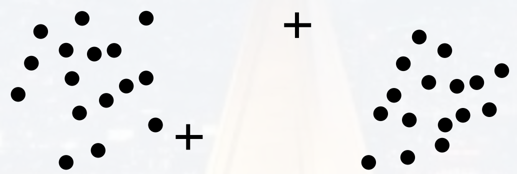
b) calculate *distance* from each point to each mean

c) assign each point to its closest mean

d) update the means accordingly

idea:

d) update the means accordingly

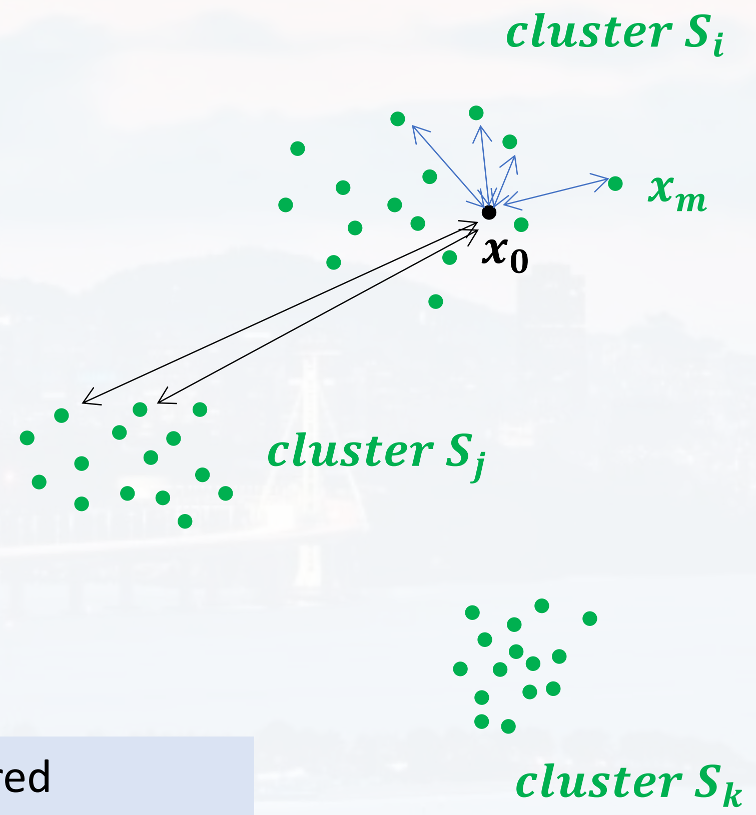


e) go back to b)

problem: ***k* = number of cluster,** is a hyperparameter. How do I know the correct value for ***k***?

→ silhouette Ψ

- distance $\boldsymbol{d_1}$ of a data point x_0 to ***its assigned cluster*** $\boldsymbol{S_i}$
  vs distance $\boldsymbol{d_2}$ to ***closest cluster (here*** $\boldsymbol{S_j}$***)***



$$\boxed{\Psi(x_0) = \begin{cases} 0 & if\ d_1 = 0 \\ \dfrac{d_2 - \boldsymbol{d_1}}{max[\boldsymbol{d_1}; d_2]} & \end{cases}}$$

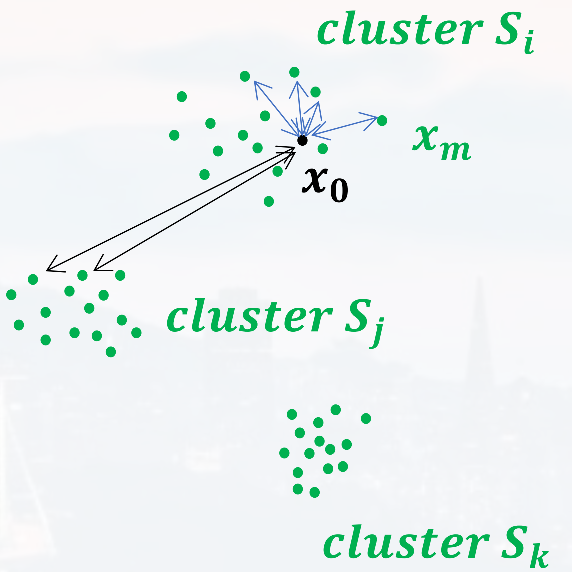
- average over all points → ψ_{tot}

if	$\psi_{tot} = 0.75\ \dots 1.00$	→ well clustered
	$\psi_{tot} = 0.50\ \dots 0.75$	→ medium clustered
	$\psi_{tot} = 0.25\ \dots 0.50$	→ poorly clustered
	$\psi_{tot} < 0.25$	→ data has no structure

problem: ***k*** is a hyperparameter. How do I know the correct value for ***k***?

→ silhouette Ψ

- distance d_1 of a data point x_0 to ***its assigned cluster*** $\boldsymbol{S_i}$
  vs distance d_2 to ***closest cluster (here*** $\boldsymbol{S_j}$***)***

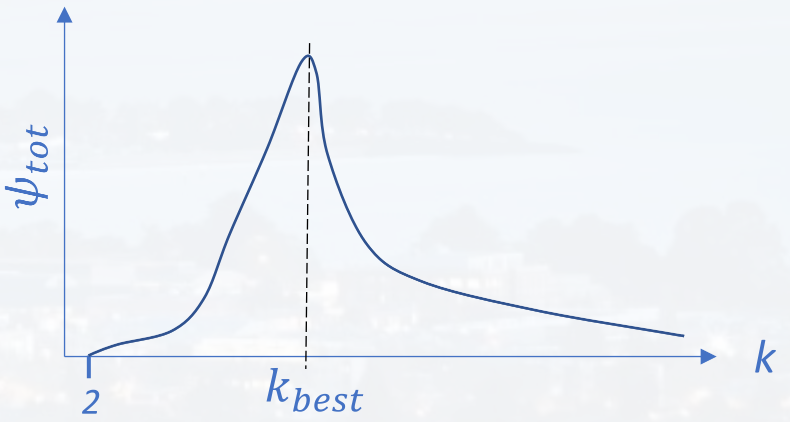


$$\boxed{\Psi(x_0) = \begin{cases} 0 & if\ d_1 = 0 \\ \dfrac{d_2 - d_1}{max[d_1; d_2]} & \end{cases}}$$

- average over all points → ψ_{tot}

$\psi_{tot} = 0.75\ \dots 1.00$ → well clustered
$\psi_{tot} = 0.50\ \dots 0.75$ → medium clustered
$\psi_{tot} = 0.25\ \dots 0.50$ → poorly clustered
$\psi_{tot} < 0.25$ → data has no structure

ideal world →



Machine Learning Algorithms:

K - Means

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

our standard libraries

```
from pyclustering.utils.metric import *
from nltk.cluster.kmeans import KMeansClusterer
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn import datasets
```

for having different distances available

performing k-means

calculating silhouette coefficient for different k

calling the famous “iris” data set

Berkeley UNIVERSITY OF CALIFORNIA

Machine Learning Algorithms: K - Means

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from pyclustering.utils.metric import *
from nltk.cluster.kmeans import KMeans(
from sklearn.metrics import silhouette_
from sklearn import datasets


iris = datasets.load_iris()

iris.DESCR
```

```
Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================
```

ideal world: three distinct cluster

```
iris = datasets.load_iris()

iris.DESCR
```

Iris Versicolor **Iris Setosa** **Iris Virginica**

loading & exploring the data:

```
iris = datasets.load_iris()

iris.DESCR
iris.feature_names
iris.target_names
```

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

four features → 4D

```
array(['setosa', 'versicolor', 'virginica']
```

check out the Jupyter Notebook **Walk_Through_Kmeans**

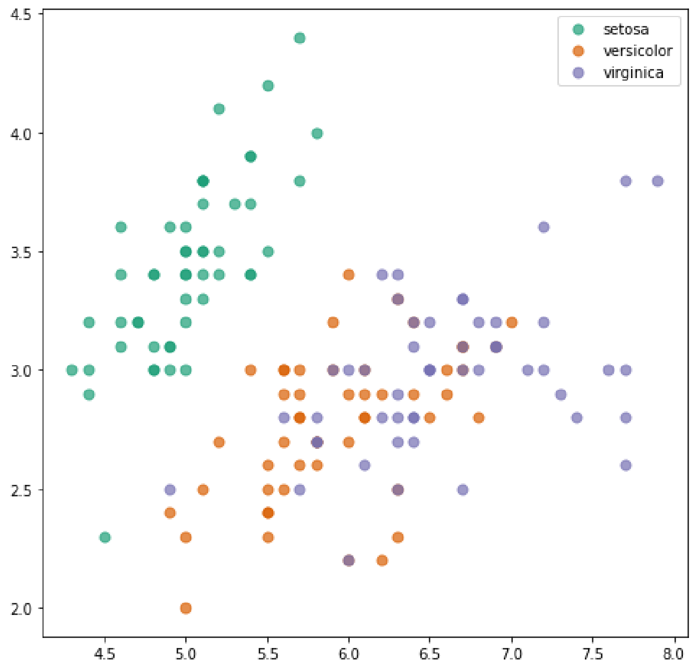
- plotting the data
- running k-means
- evaluating the result

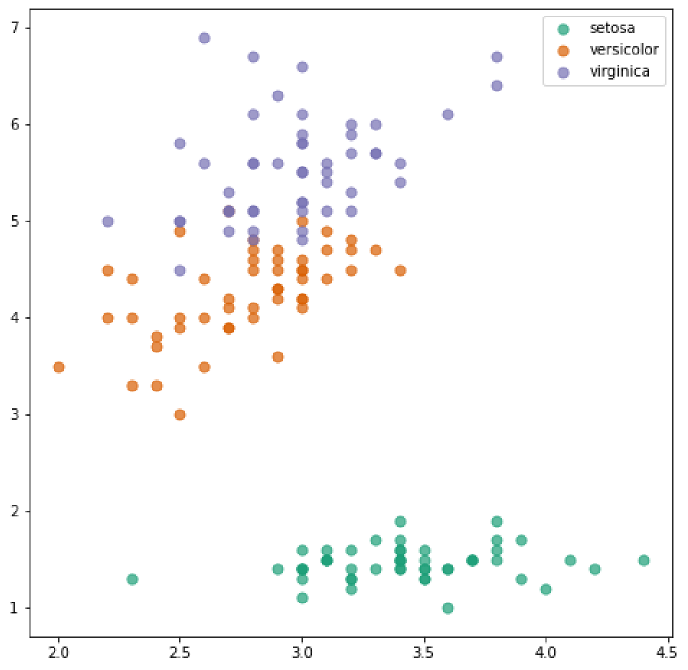
Machine Learning Algorithms:

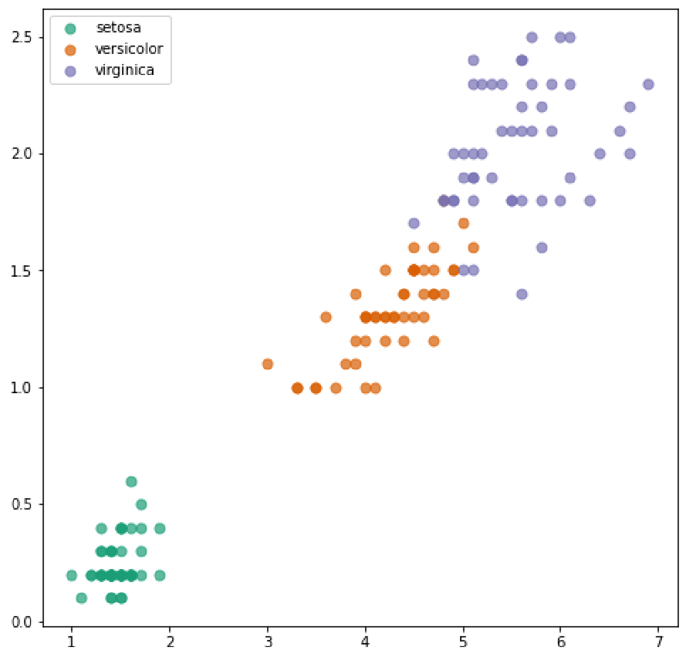

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

4D dataset → plotting two components

- **plotting the data**
- running k-means
- evaluating the result







Machine Learning Algorithms: K - Means

- plotting the data
- **running k-means**
- evaluating the result

```
nClust      = 3
rep         = 25
dist        = distance_metric(type_metric.EUCLIDEAN)


my_model   = KMeansClusterer(nClust, distance = dist,\
               repeats   = rep,\
               avoid_empty_clusters = True)

PredLabels = my_model.cluster(X2D,\
               assign_clusters = True)

Center     = my_model.means()
```

we need to "guess" the number of cluster

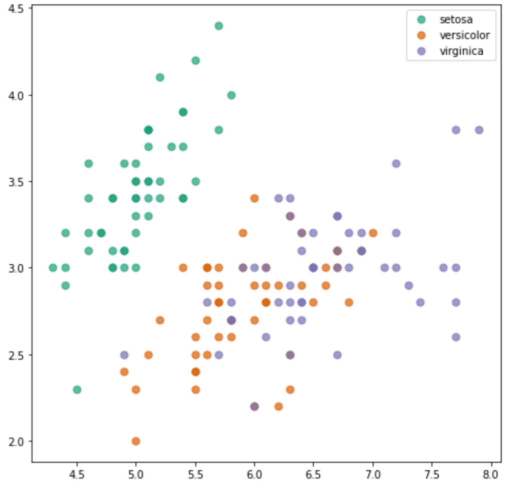
the **initial** means are assigned randomly.
→ repeat the procedure 25 times
→ avoiding local minimum,

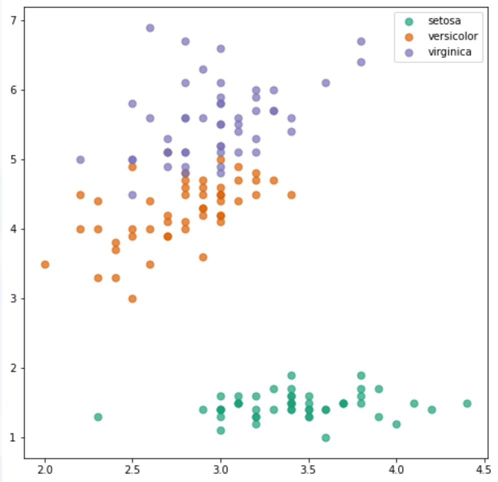
The features are meassured in cm, i. e. the correct distance to pick here is Euclidean

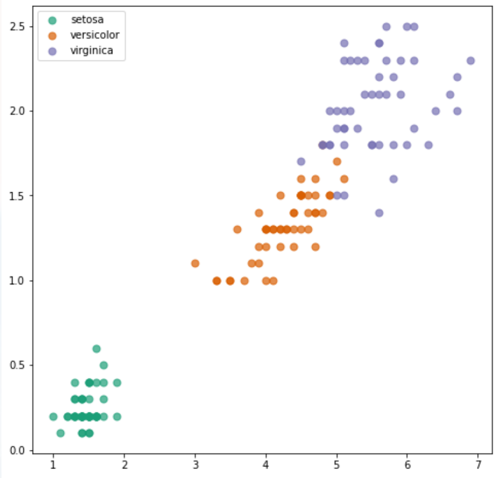
Machine Learning Algorithms: K - Means

- plotting the data
- **running k-means**
- evaluating the result

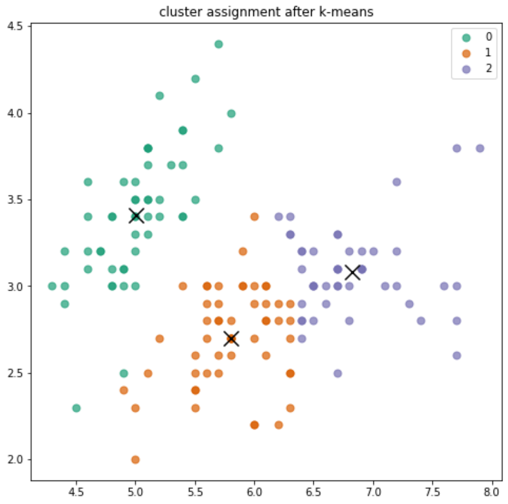
true classes

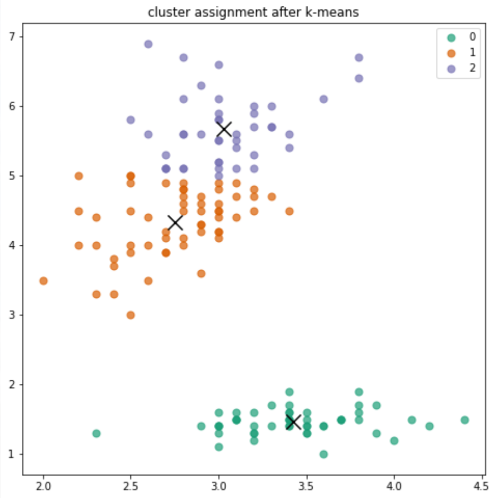


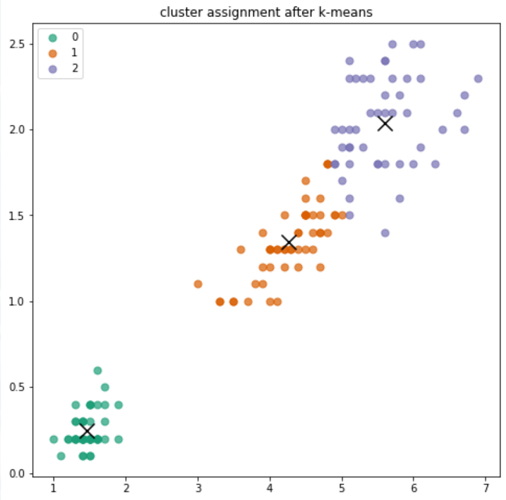




assigned classes



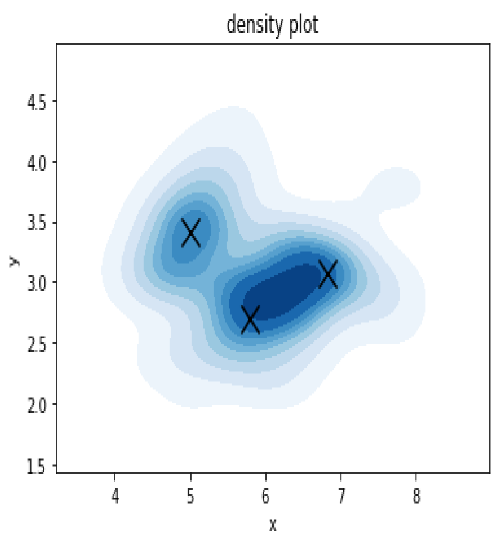


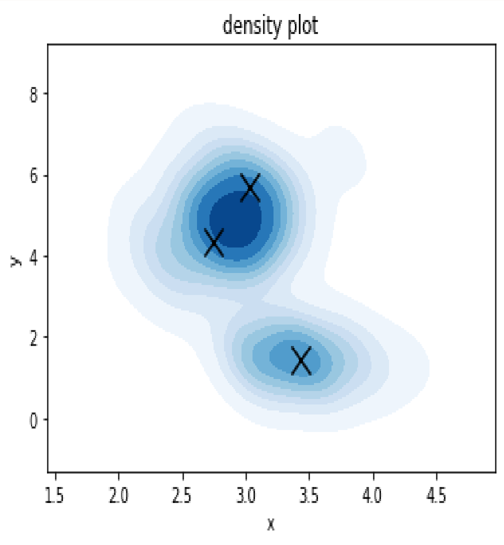


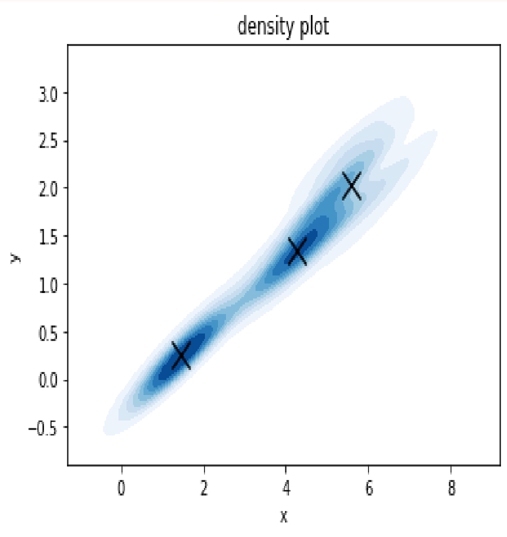
Machine Learning Algorithms: K - Means

- plotting the data
- **running k-means**
- evaluating the result

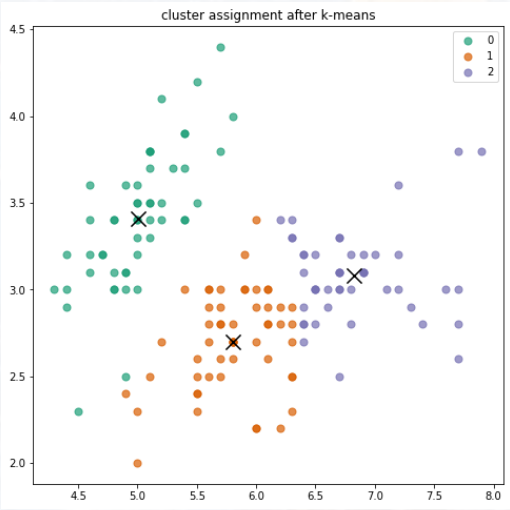
density

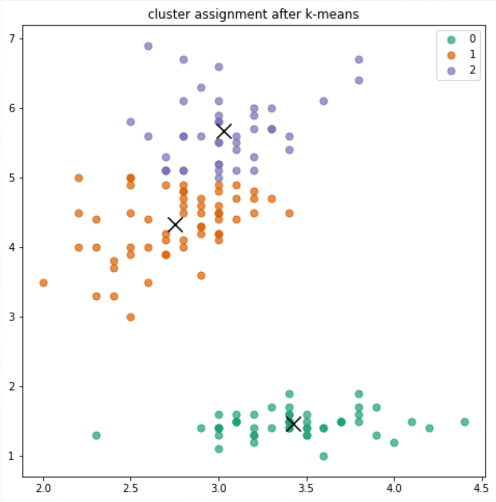


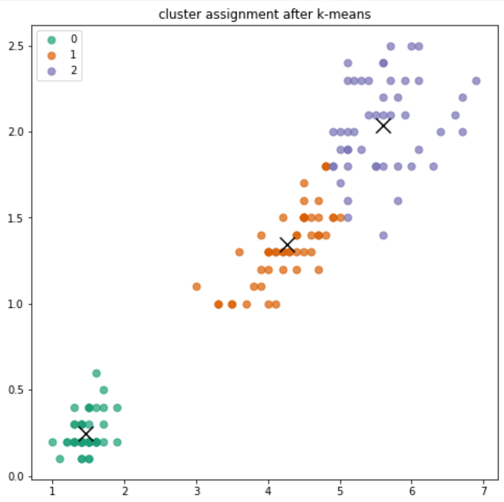




assigned classes



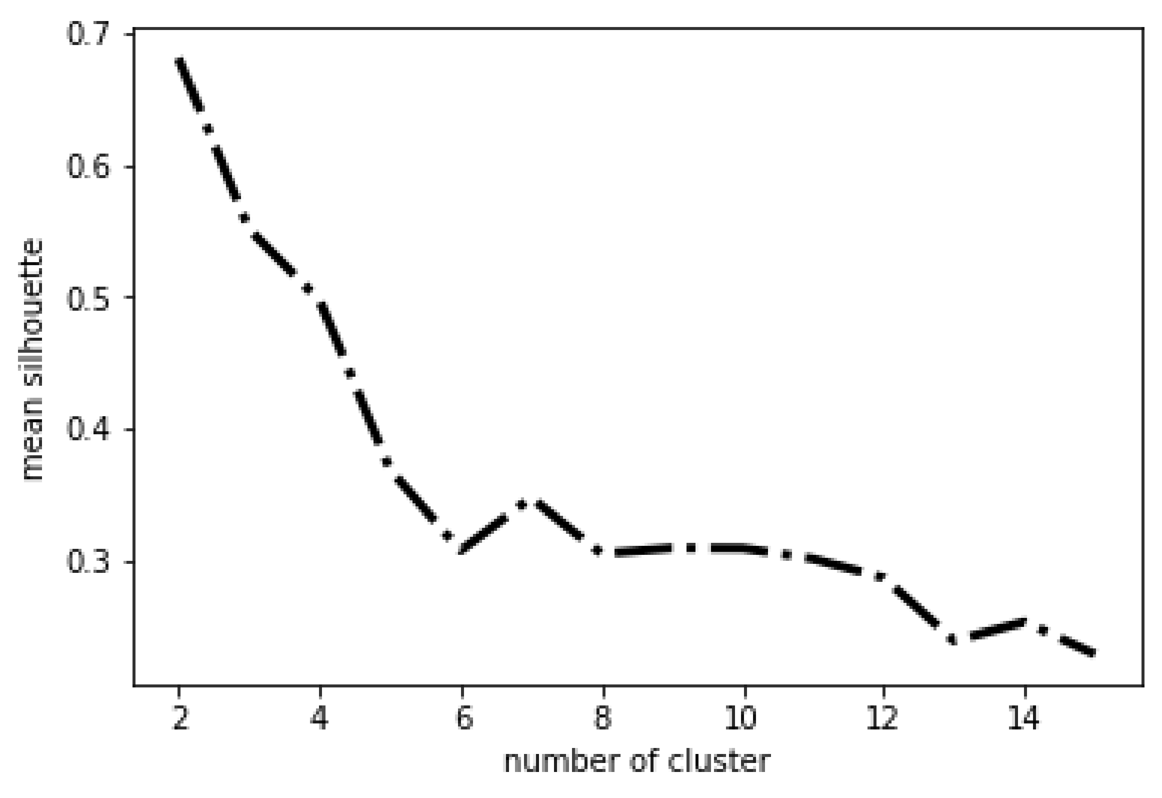


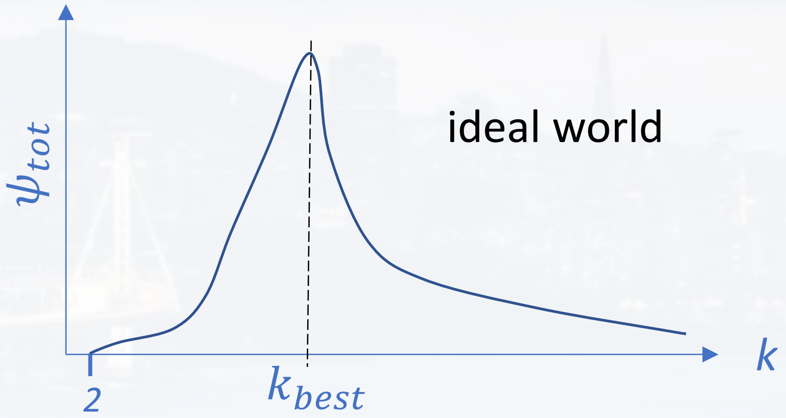


we run k-means now for the **full 4D** dataset
+ evaluate clustering with silhouette

- plotting the data
- running k-means
- **evaluating the result**

```
silhouette_score(X, Labels)
```



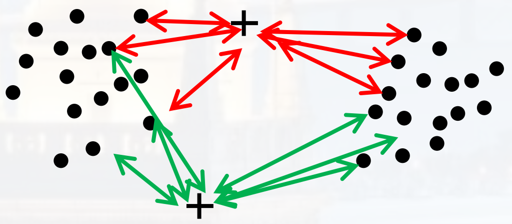


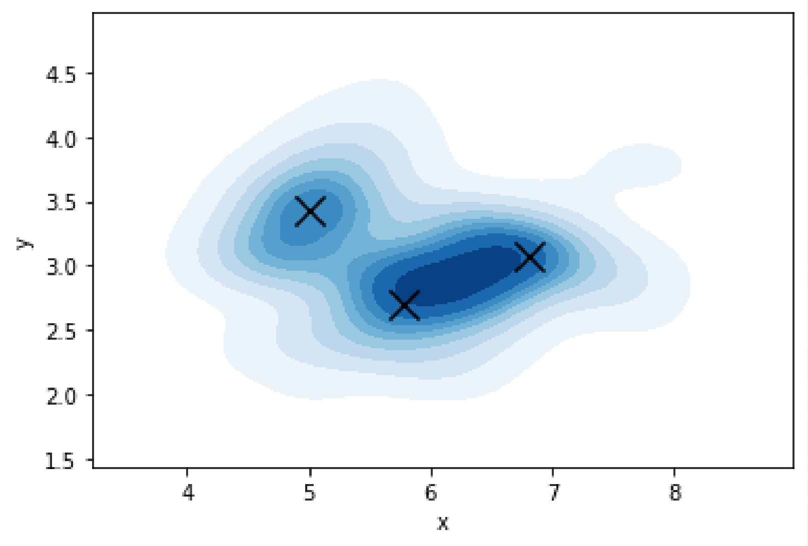
accuracy for 4D k = 3: 90%

summary:

- simple and fast
- unsupervised
- *k* has to be given → silhouette for determining best *k*
- problems if cluster have unusual shapes

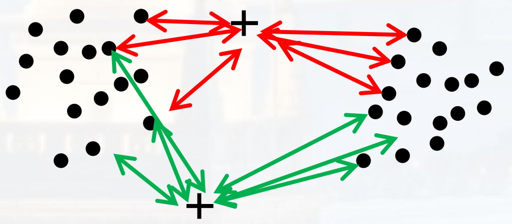
(elongated, hollow inside, scattered)

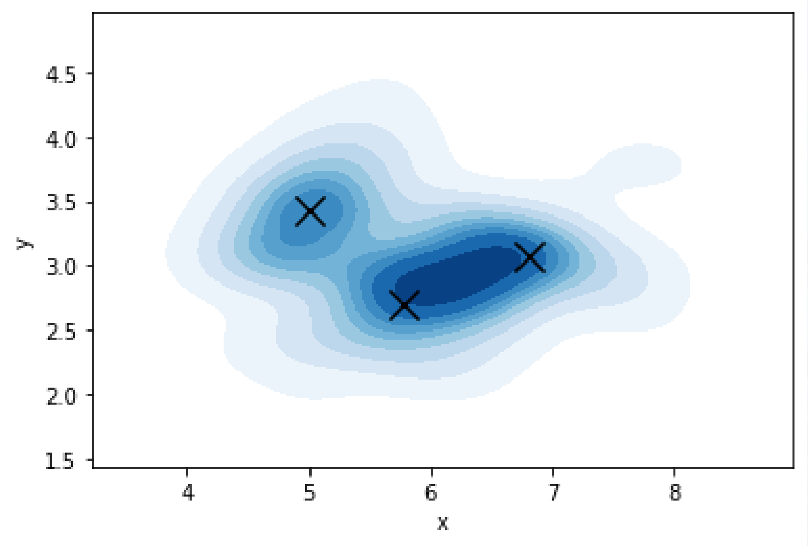


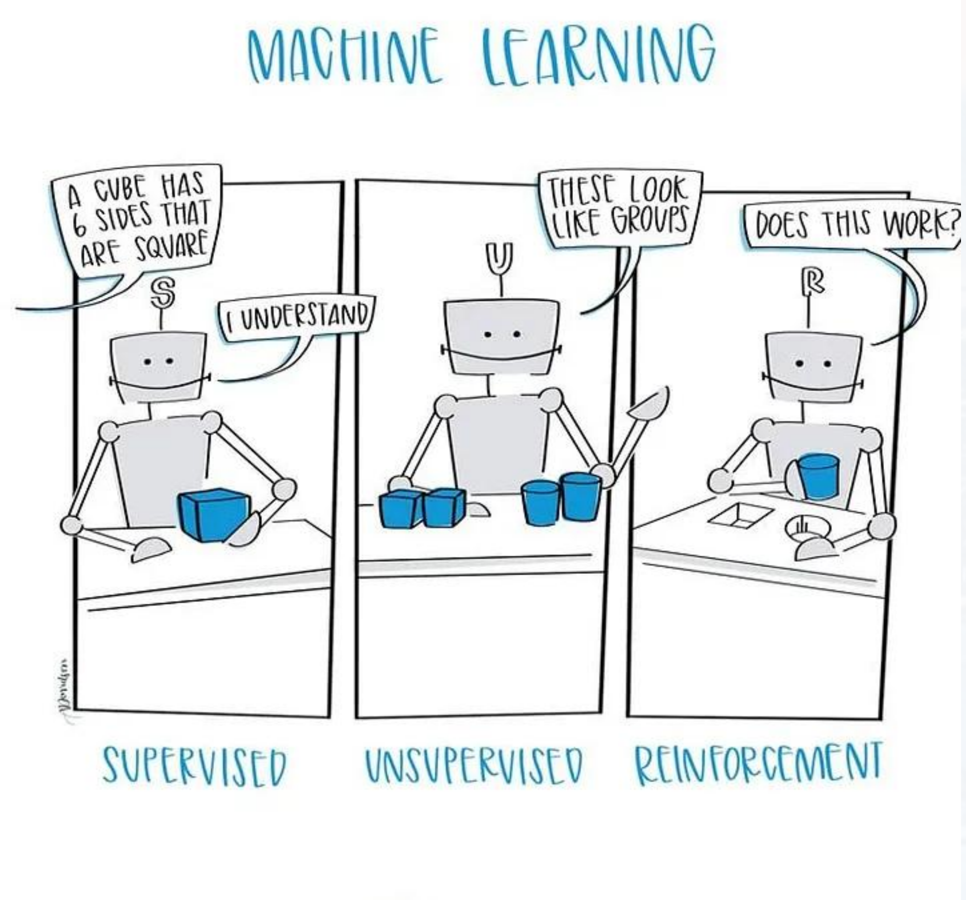


<u>topics for the discussion/office hour:</u>

- What is a *distance*?
- Which are different distances?
- When to use which distance?







Outline

- K - means

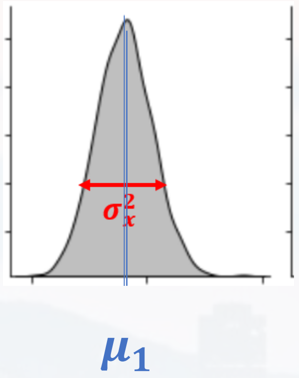
- GMM

- trees

Gaussian **M**ixture **M**odels

<u>one feature</u>

$$N_1(x_1) = \frac{1}{\sqrt{2\pi\, \sigma_{x1}^2}} \exp - \frac{1}{2}\left(\frac{x_1 - \mu_1}{\sigma_{x1}}\right)^2$$



<u>two features</u>

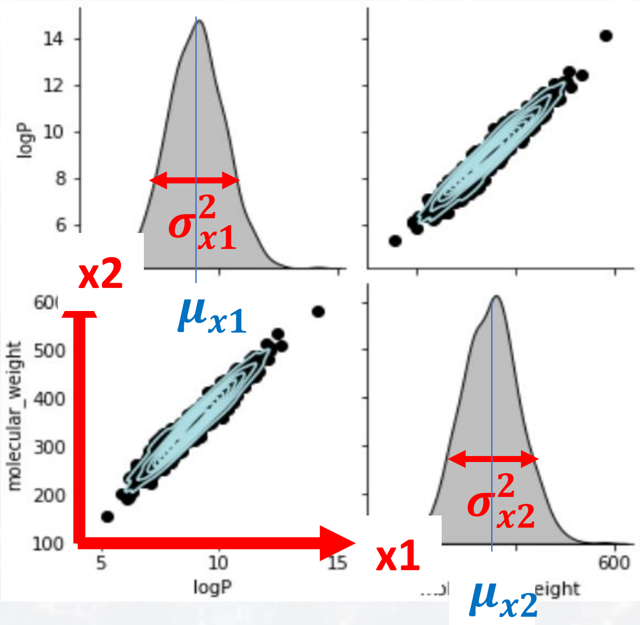
$$\Sigma = \begin{pmatrix} \sigma_{x1}^2 & cov(x_1, x_2) \\ cov(x_2, x_1) & \sigma_{x2}^2 \end{pmatrix}$$

covariance matrix

$$\begin{pmatrix} x_1 - \mu_{x1} \\ x_2 - \mu_{x2} \end{pmatrix}^T \Sigma^{-1} \begin{pmatrix} x_1 - \mu_{x1} \\ x_2 - \mu_{x2} \end{pmatrix}$$

see PCA lecture

$$N_2(x_1, x_2) = \frac{1}{2\pi \det(\Sigma)^{1/2}} \exp - \frac{1}{2}\left[\begin{pmatrix} x_1 - \mu_{x1} \\ x_2 - \mu_{x2} \end{pmatrix}^T \Sigma^{-1} \begin{pmatrix} x_1 - \mu_{x1} \\ x_2 - \mu_{x2} \end{pmatrix}\right]$$



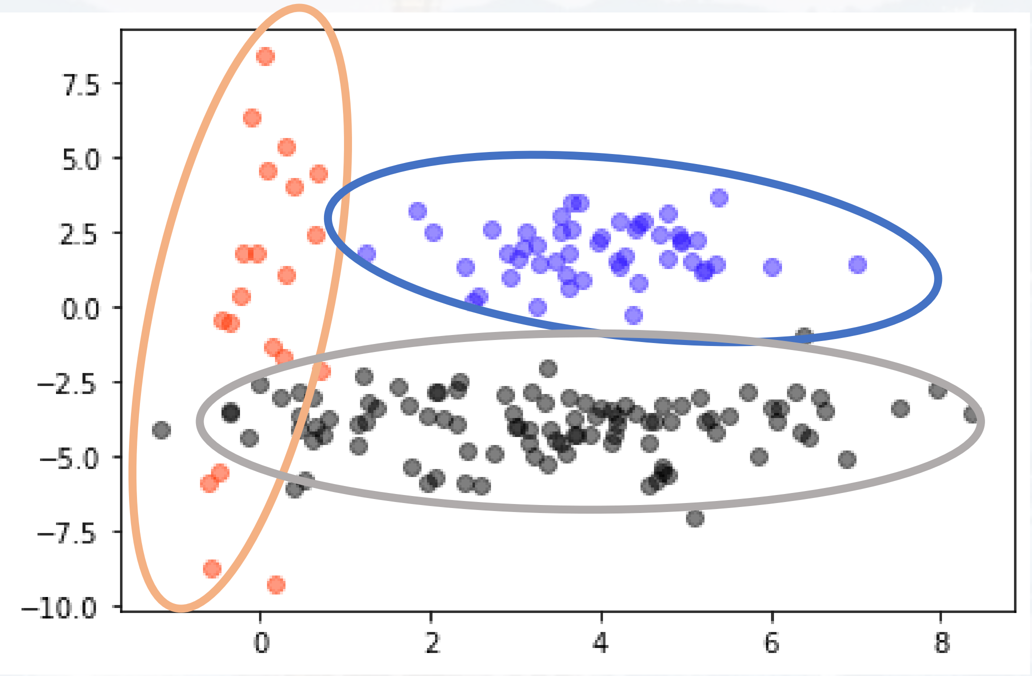
Machine Learning Algorithms:

Gaussian **M**ixture **M**odels

n features

$$N_k(, x_2 \dots x_n) = \frac{1}{(2\pi)^{n/2}\det(\Sigma)^{1/2}}\, exp\left[-\frac{1}{2}(x - \mu)^T \Sigma^{-1}(x - \mu)\right]$$

vectors x and μ



two features, *k=3* components

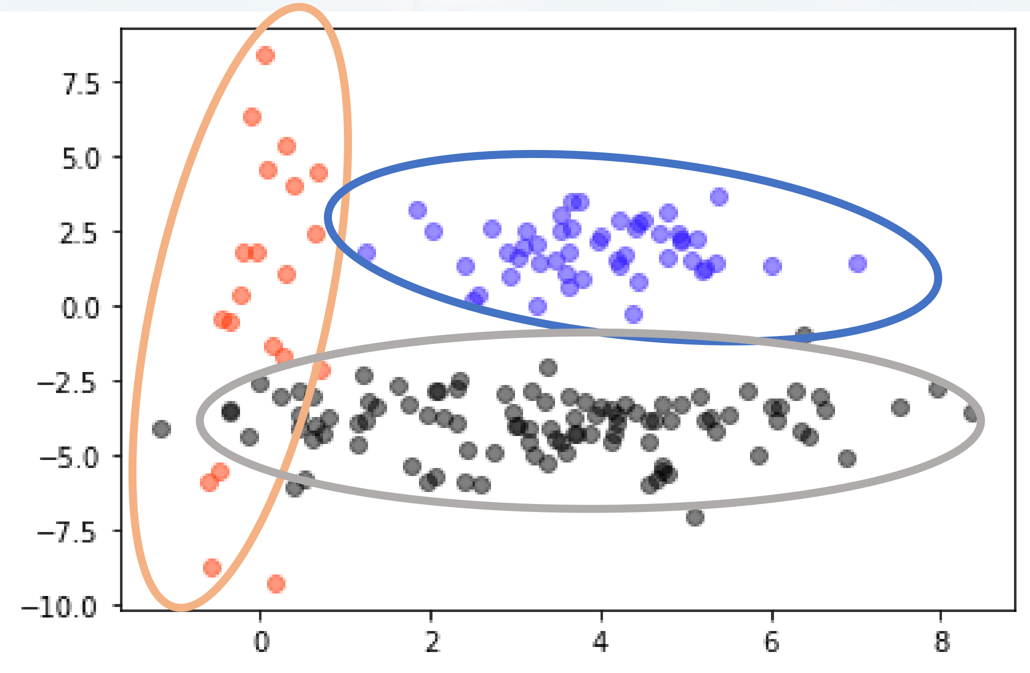
Machine Learning Algorithms:

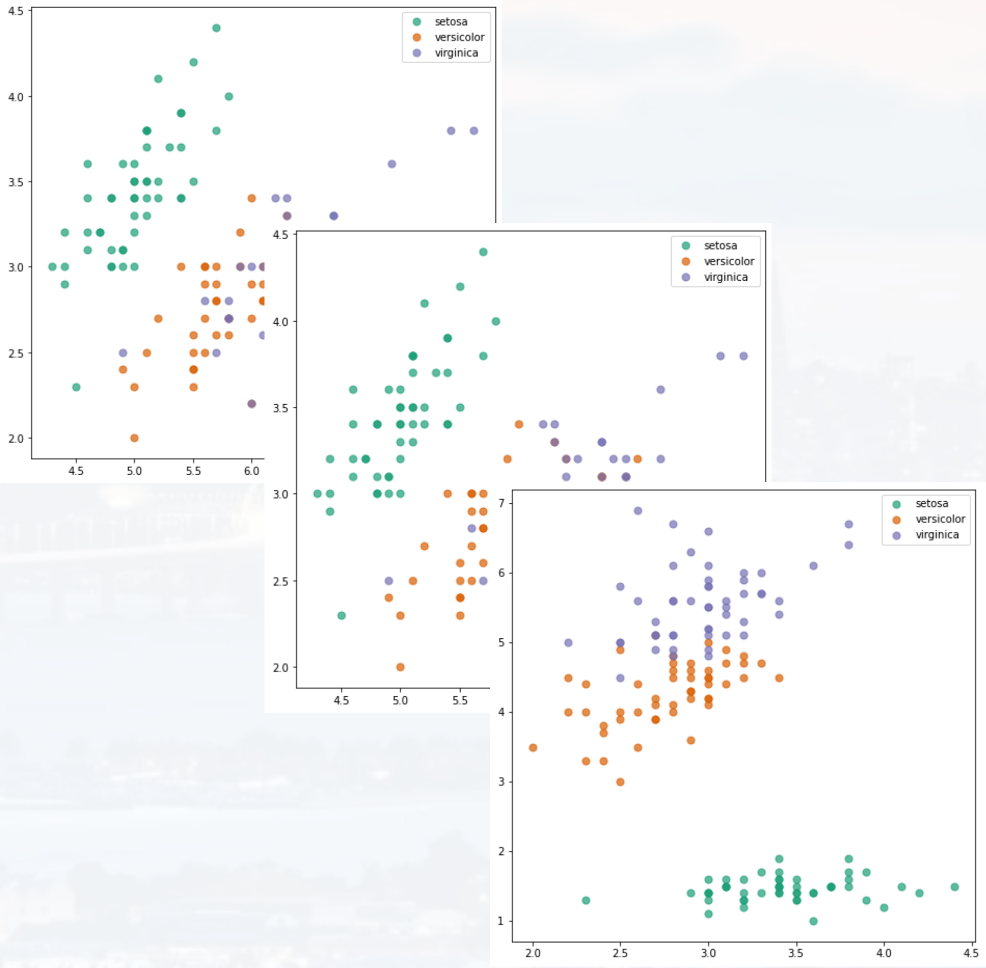
GMM

Gaussian Mixture Models

four features, *k=3* components

two features, *k=3* components





Gaussian **M**ixture **M**odels

idea: fitting the data to a GMM → analytical functions to **calculate** probabilities for labels

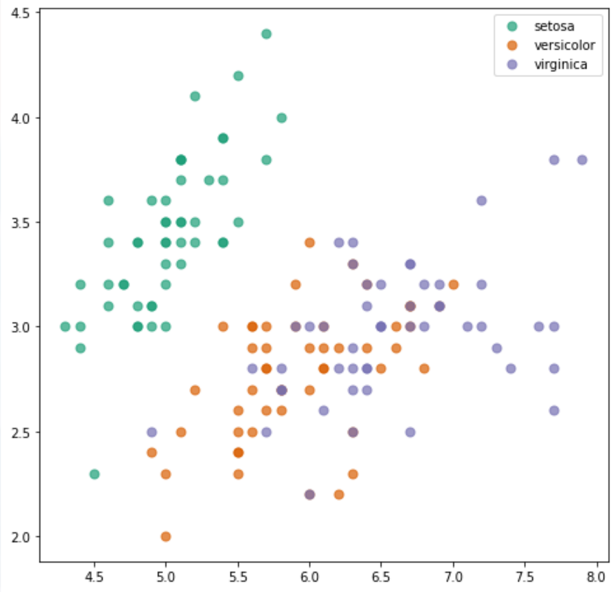
different algorithms:
- Bayesian
- **E**xpectation **M**aximization
- …

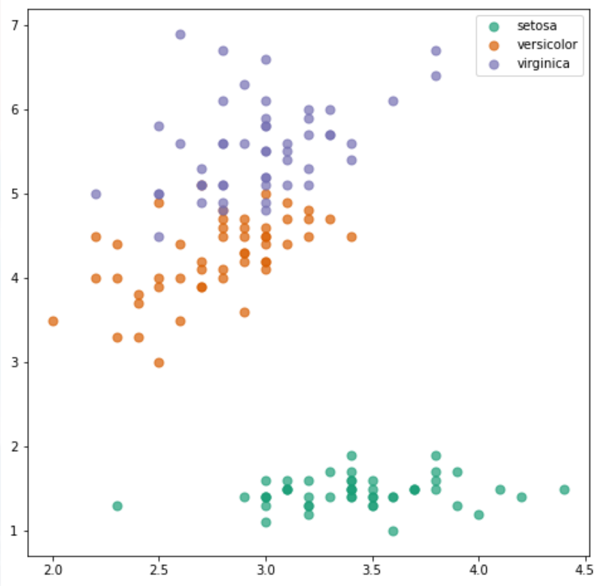
```
my_model   = GaussianMixture(n_components = k, random_state = 0).fit(X)

Center     = my_model.means_
PredLabels = my_model.predict(X)
```

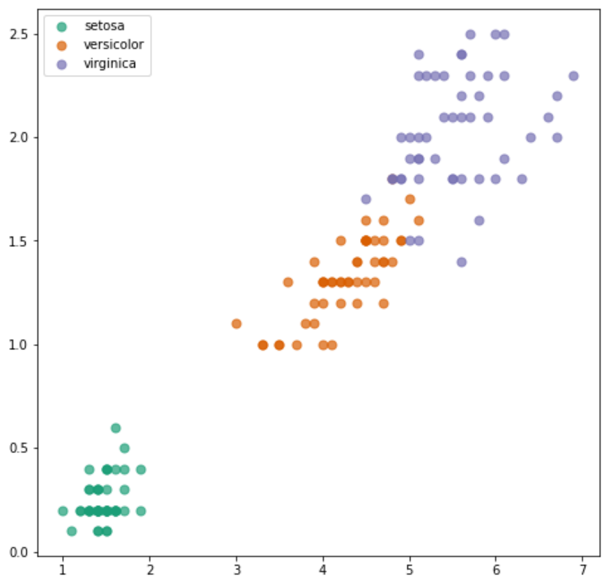
setting initial labels randomly

Machine Learning Algorithms: GMM

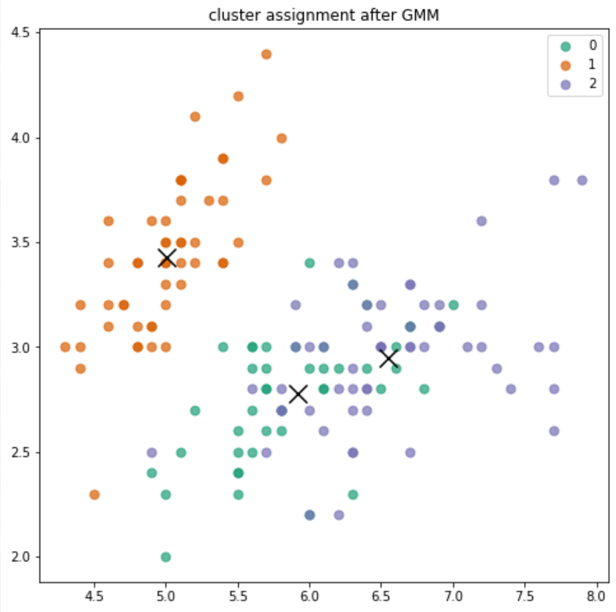
true classes

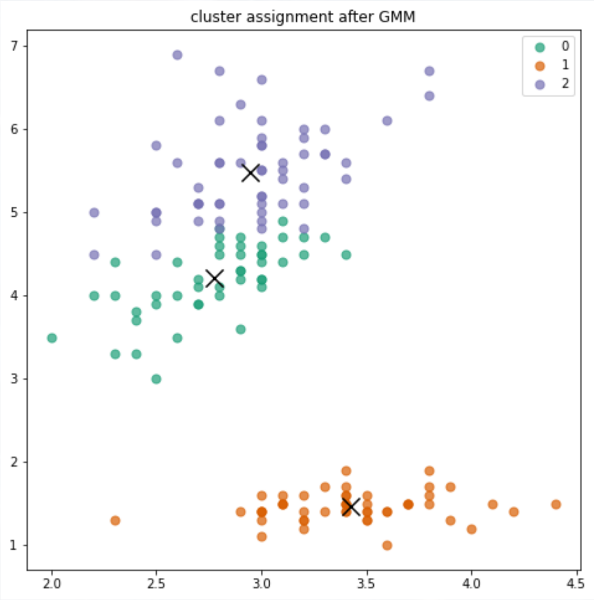


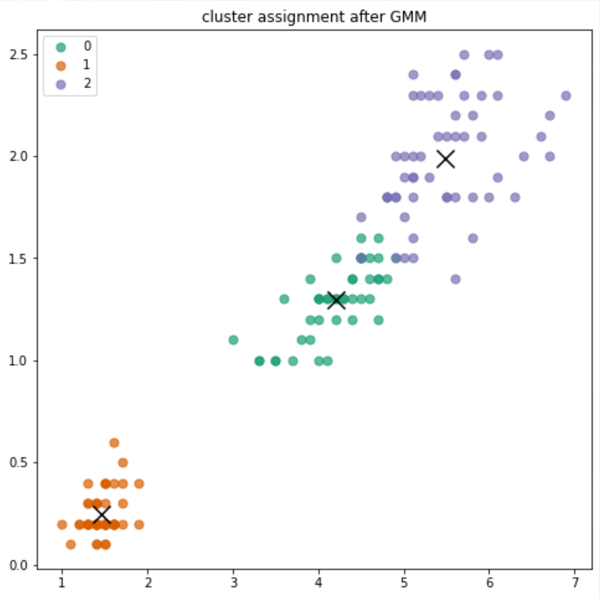




assigned classes

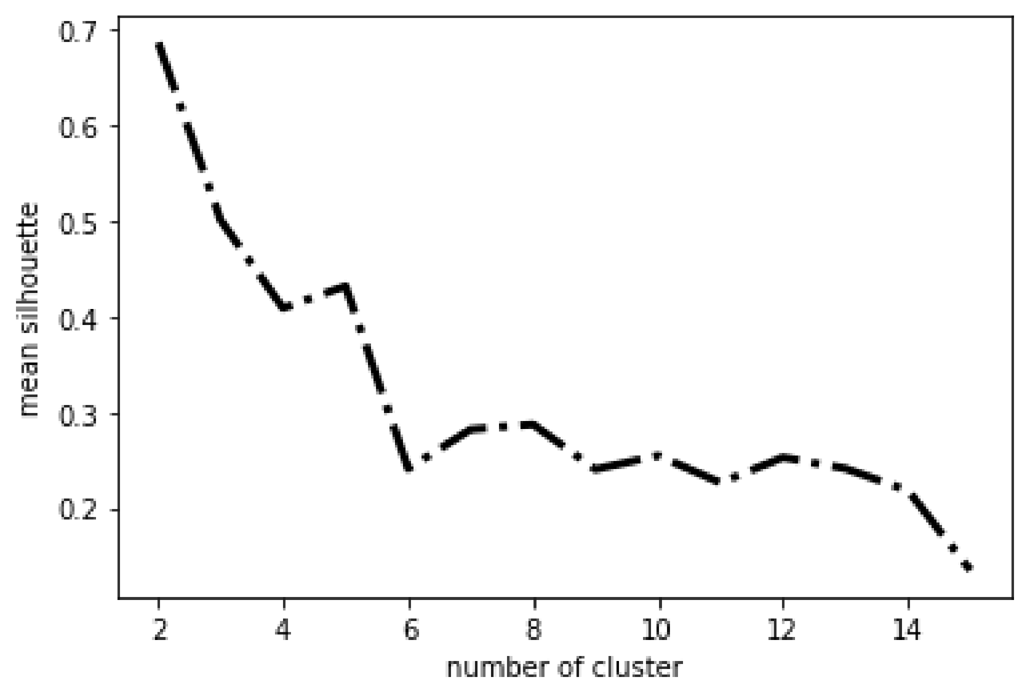






Machine Learning Algorithms:

we run k-means now for the **full 4D** dataset
+ evaluate clustering with silhouette

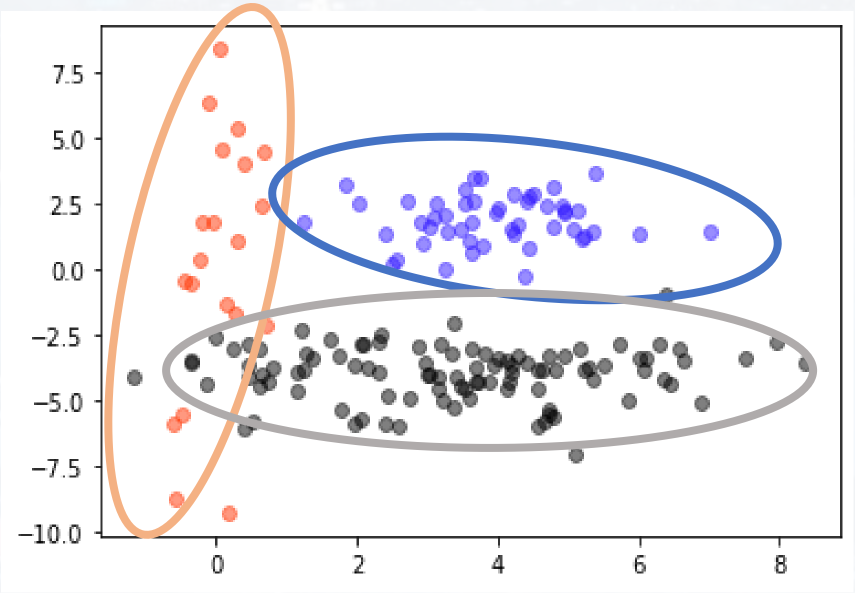


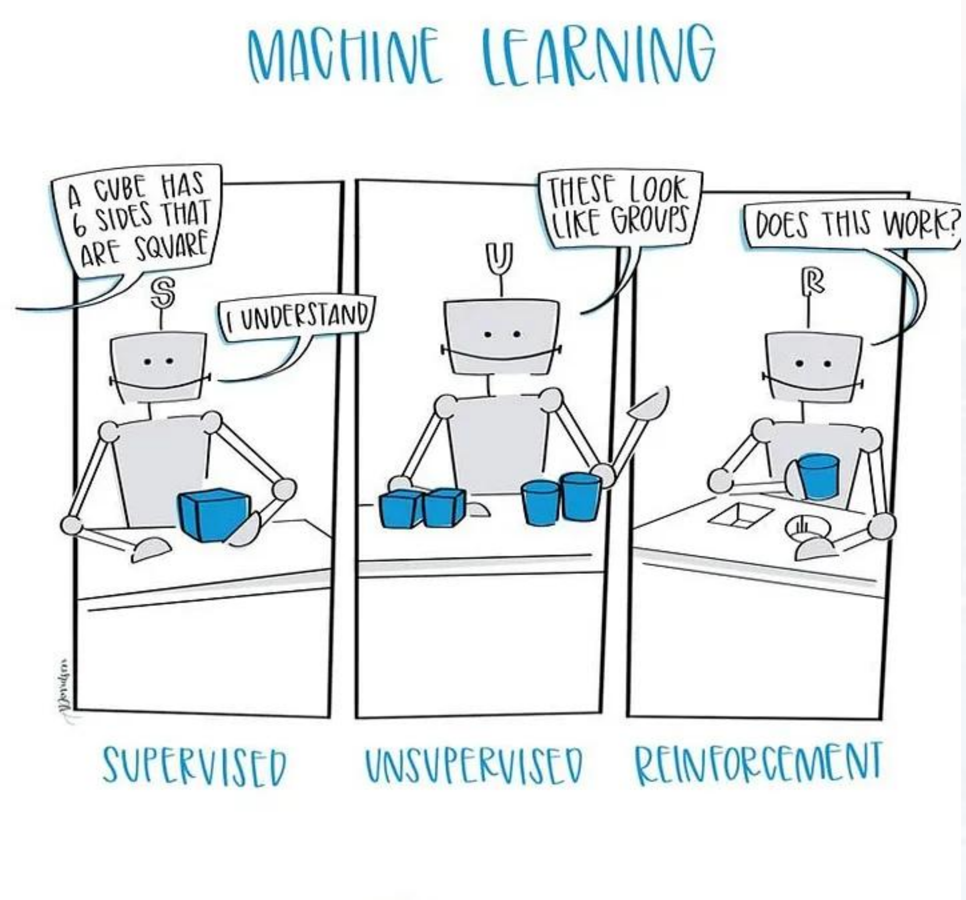
accuracy for 4D k = 3: 93%

Machine Learning Algorithms: GMM

topics for the discussion/office hour:

- EM algorithm
- mean, variance and covariance in more detail





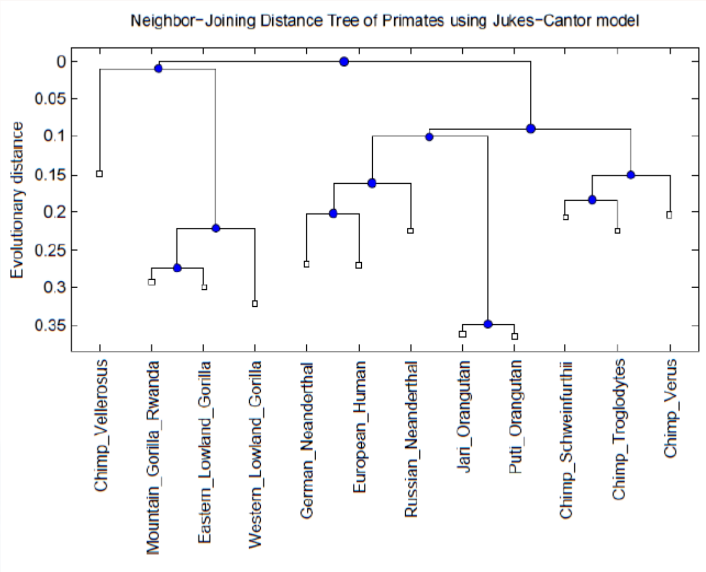
Outline

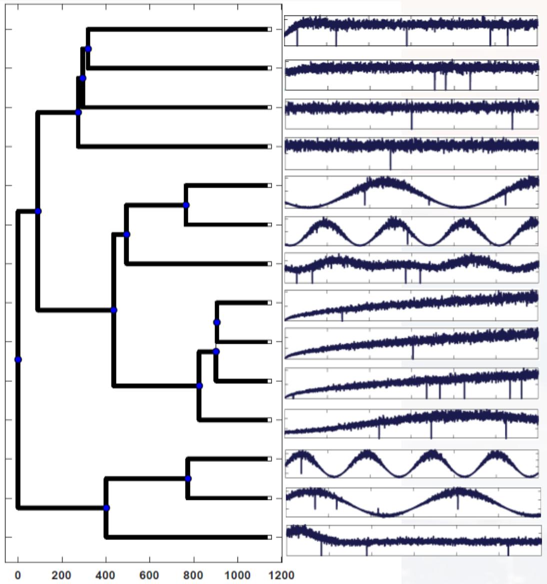
- K - means

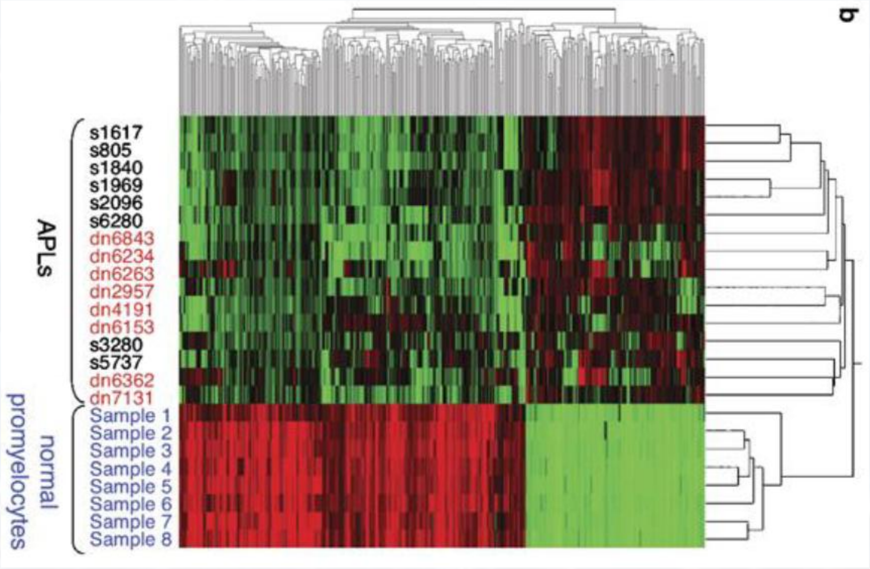
- GMM

- **trees**

Machine Learning Algorithms: Trees





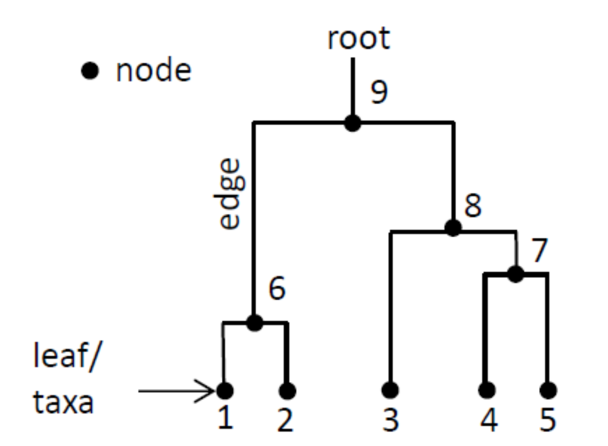


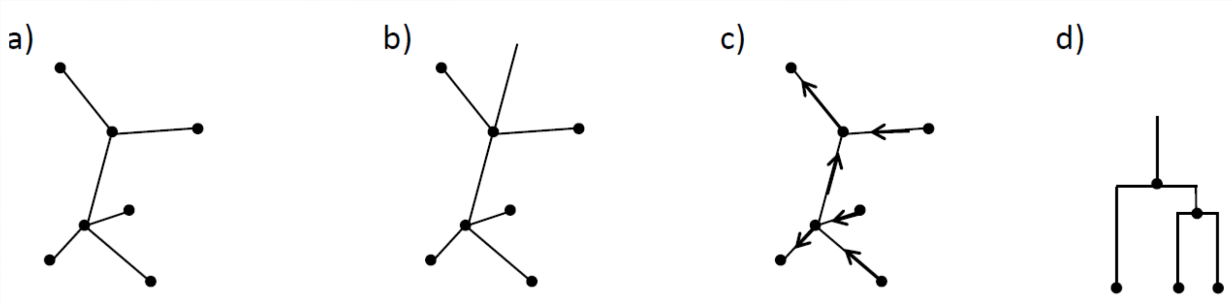
- What is a tree?
- Different kinds of trees...?
- How to build a tree?
- Why do we need trees?
- Examples...

Machine Learning Algorithms:

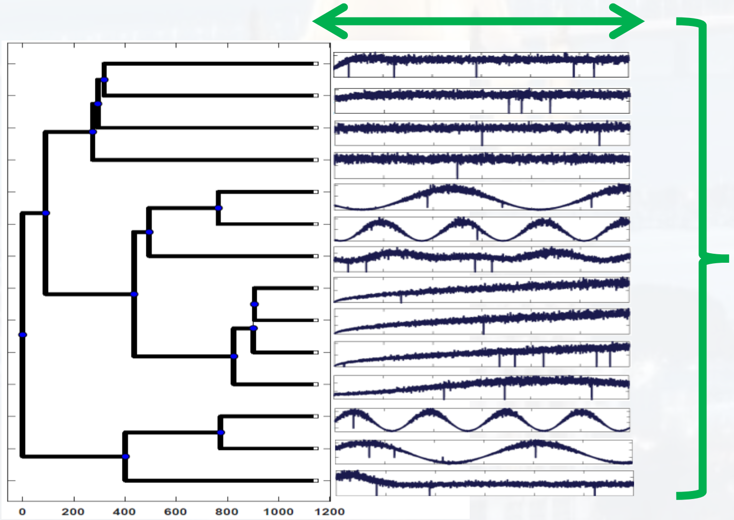
trees are a subclass of graphs **(but: not fully connected → "hierarchy", no loops)**:

a) unrooted, undirected multinary tree
b) rooted, undirected multinary tree
c) unrooted, directed multinary tree
d) rooted, undirected binary tree

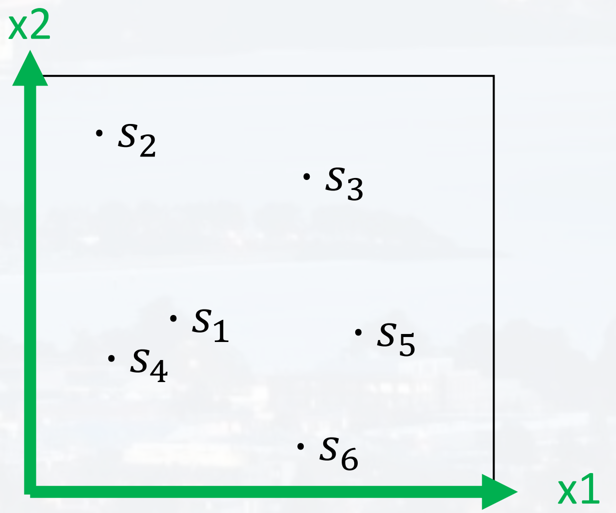




N timepoints



each sample s is a vector of N rows, hence, a data point in N-D



Machine Learning Algorithms:

constructing trees:

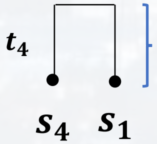
- calculating a distance between each pair of samples

	s_1	s_2	s_3	s_4	s_5	s_6
s_1	0	d(s_1, s_2)	d(s_1, s_3)	d(s_1, s_4)	d(s_1, s_5)	d(s_1, s_6)
s_2		0	d(s_2, s_3)	d(s_2, s_4)	d(s_2, s_5)	d(s_2, s_6)
s_3			0	d(s_3, s_4)	d(s_3, s_5)	d(s_3, s_6)
s_4				0	d(s_4, s_5)	d(s_4, s_6)
s_5					0	d(s_5, s_6)
s_6						0

Question: What could be a proper distance definition here?

...once a distance has been defined...

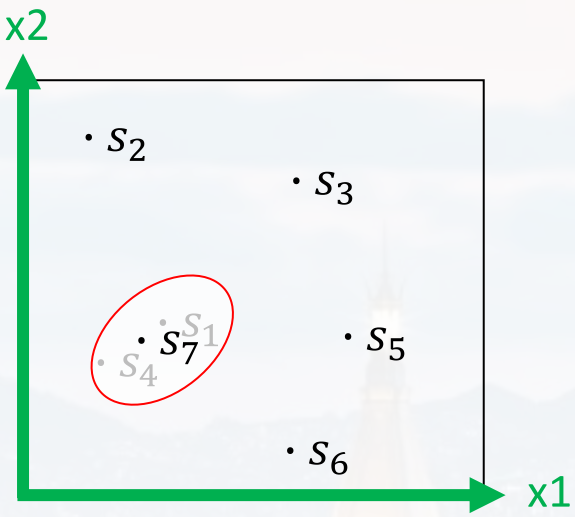
→ find the closest pair



$t_1 = t_4 = \frac{1}{2} d(s_1, s_4)$

Machine Learning Algorithms: Trees

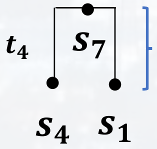
constructing trees:



- calculating a distance between each pair of samples

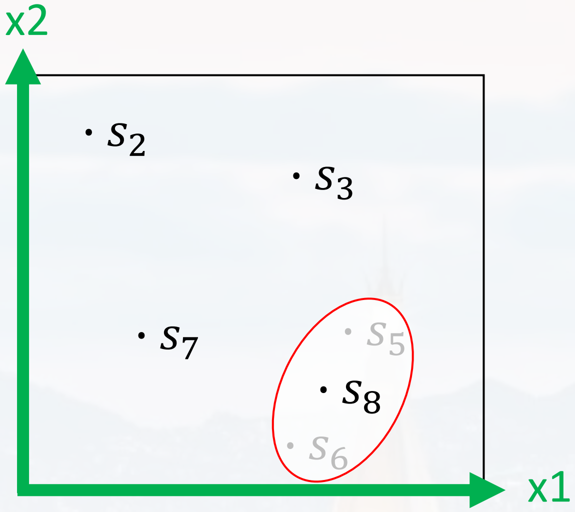
	s_1	s_2	s_3	s_4	s_5	s_6
s_1	0	d(s_1, s_2)	d(s_1, s_3)	d(s_1, s_4)	d(s_1, s_5)	d(s_1, s_6)
s_2		0	d(s_2, s_3)	d(s_2, s_4)	d(s_2, s_5)	d(s_2, s_6)
s_3			0	d(s_3, s_4)	d(s_3, s_5)	d(s_3, s_6)
s_4				0	d(s_4, s_5)	d(s_4, s_6)
s_5					0	d(s_5, s_6)
s_6						0

→ treat it as a new cluster $s_{1,4}$
→ use average of distance from cluster elements



$t_1 = t_4 = \frac{1}{2} d(s_1, s_4)$

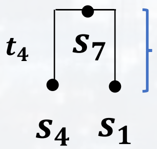
constructing trees:



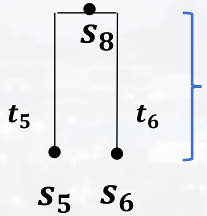
→ find the closest pair
(now including the cluster)

- calculating a distance between each pair of samples

	s_1	s_2	s_3	s_4	s_5	s_6
s_1	**0**	**d(s_1, s_2)**	**d(s_1, s_3)**	**d(s_1, s_4)**	**d(s_1, s_5)**	**d(s_1, s_6)**
s_2		**0**	**d(s_2, s_3)**	**d(s_2, s_4)**	**d(s_2, s_5)**	**d(s_2, s_6)**
s_3			**0**	**d(s_3, s_4)**	**d(s_3, s_5)**	**d(s_3, s_6)**
s_4				**0**	**d(s_4, s_5)**	**d(s_4, s_6)**
s_5					**0**	**d(s_5, s_6)**
s_6						**0**

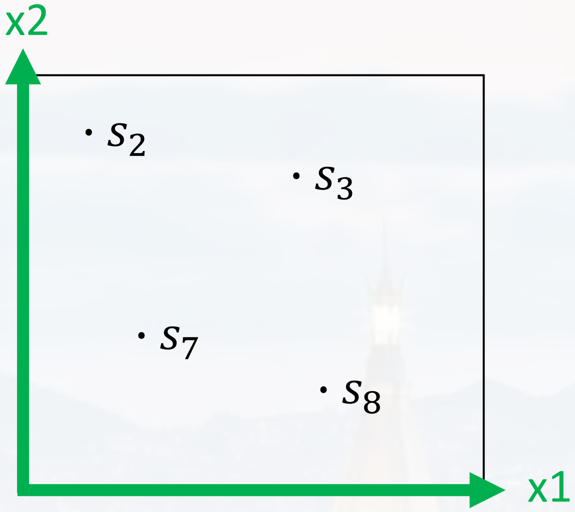


$t_1 = t_4 = \frac{1}{2} d(s_1, s_4)$



$t_6 = t_5 = \frac{1}{2} d(s_5, s_6)$

constructing trees:

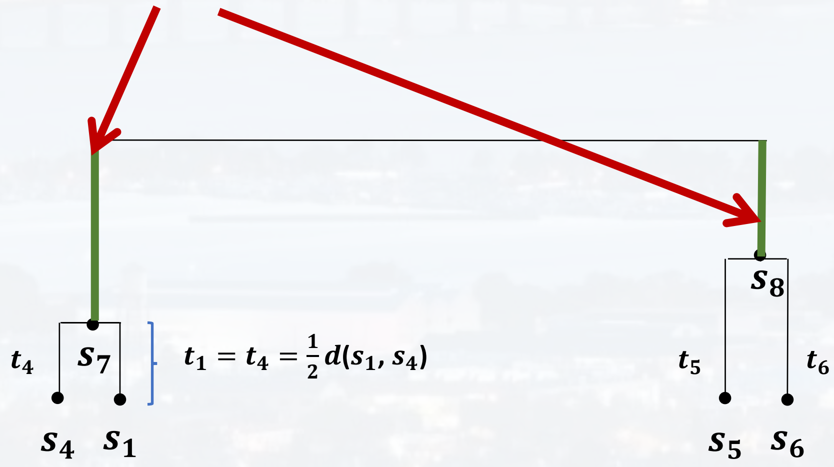


→ and so on....

- calculating a distance between each pair of samples

	s_1	s_2	s_3	s_4	s_5	s_6
s_1	0	d(s_1, s_2)	d(s_1, s_3)	d(s_1, s_4)	d(s_1, s_5)	d(s_1, s_6)
s_2		0	d(s_2, s_3)	d(s_2, s_4)	d(s_2, s_5)	d(s_2, s_6)
s_3			0	d(s_3, s_4)	d(s_3, s_5)	d(s_3, s_6)
s_4				0	d(s_4, s_5)	d(s_4, s_6)
s_5					0	d(s_5, s_6)
s_6						0

$$d(s_7, s_8) = \frac{d(s_5, s_7) + d(s_6, s_7)}{2} = \frac{d(s_1, s_5) + d(s_4, s_5) + d(s_1, s_6) + d(s_4, s_6)}{4}$$



$$t_1 = t_4 = \frac{1}{2} d(s_1, s_4)$$

$$t_8 = \frac{1}{2}\left(d(s_7, s_8) + \frac{1}{2} d(s_5, s_6) + \frac{1}{2} d(s_4, s_1)\right) - \frac{1}{2} d(s_5, s_6)$$

$$t_6 = t_5 = \frac{1}{2} d(s_5, s_6)$$

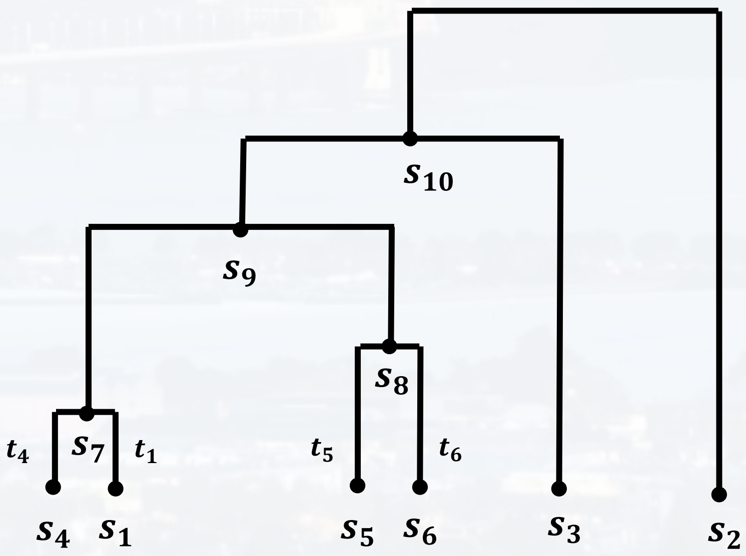
constructing trees:

- calculating a distance between each pair of samples

	s_1	s_2	s_3	s_4	s_5	s_6
s_1	0	d(s_1, s_2)	d(s_1, s_3)	d(s_1, s_4)	d(s_1, s_5)	d(s_1, s_6)
s_2		0	d(s_2, s_3)	d(s_2, s_4)	d(s_2, s_5)	d(s_2, s_6)
s_3			0	d(s_3, s_4)	d(s_3, s_5)	d(s_3, s_6)
s_4				0	d(s_4, s_5)	d(s_4, s_6)
s_5					0	d(s_5, s_6)
s_6						0

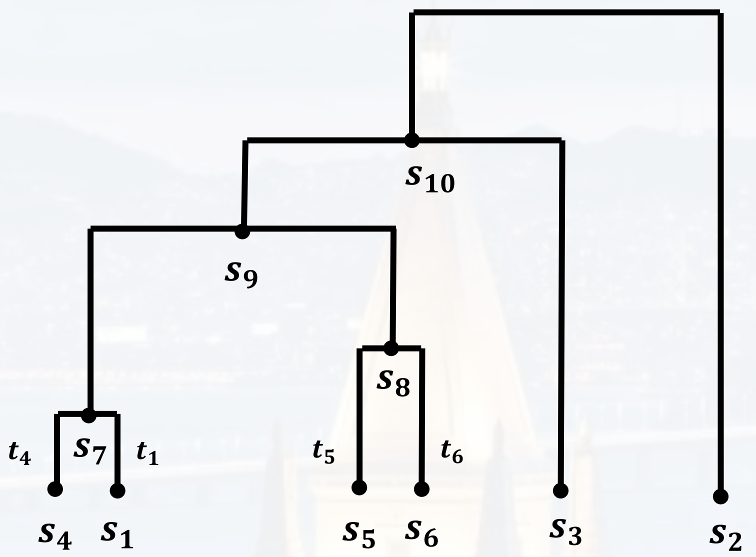
....finally

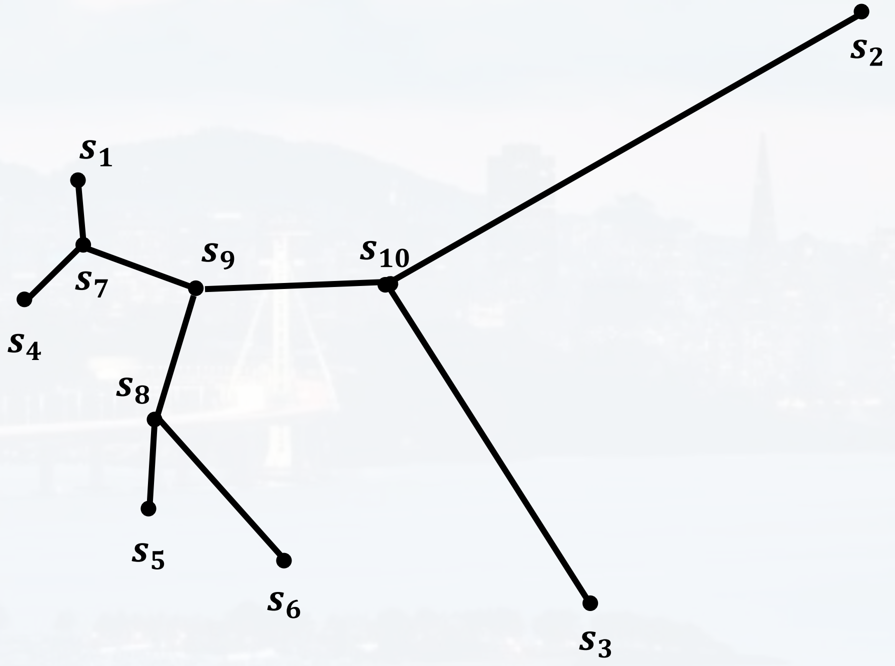
→ **U**nweighted **P**air **G**roup **M**ethod **U**sing **A**rithmetic **A**verages (UPGMA)



constructing trees:

→ **U**nweighted **P**air **G**roup **M**ethod
Using **A**rithmetic **A**verages (UPGMA)





note: sometimes these diagrams might be misleading when interpreting the distances between the nodes

Python libraries:

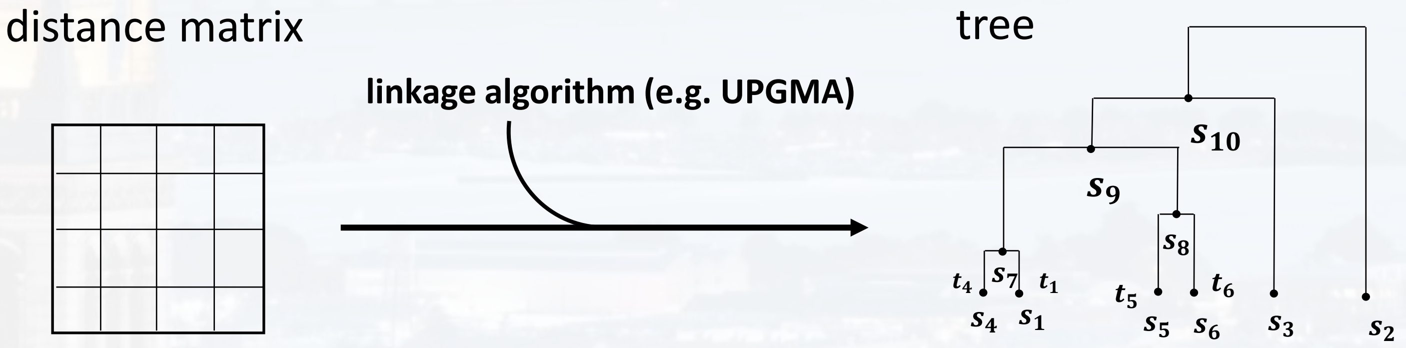
libraries from the *Bio* package

```
from Bio.Phylo.TreeConstruction import DistanceCalculator, DistanceTreeConstructor
from Bio import Phylo
from scipy import spatial
```

for plotting a phylogenetic tree

for rearranging a distance matrix

general idea:



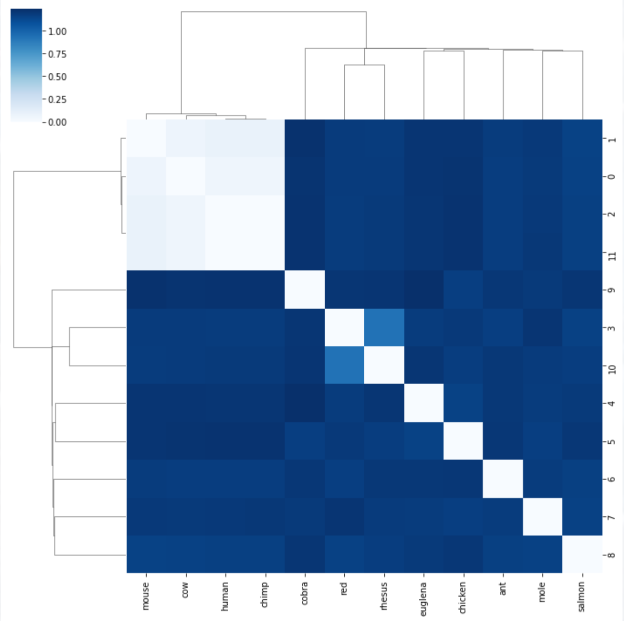
Machine Learning Algorithms: Trees

Python libraries:

also most heatmap tools have some abilities to construct trees: `sns.clustermap`

```
sns.clustermap(D_df, cmap = 'Blues', \
                          row_cluster = True, col_cluster = True, \
                          metric = 'euclidean', method = 'average', \
                          yticklabels = True)
plt.show()
```

similar to UPGMA



Machine Learning Algorithms: Trees

topics for the discussion/office hour:

- distances
- random forest
- graphs

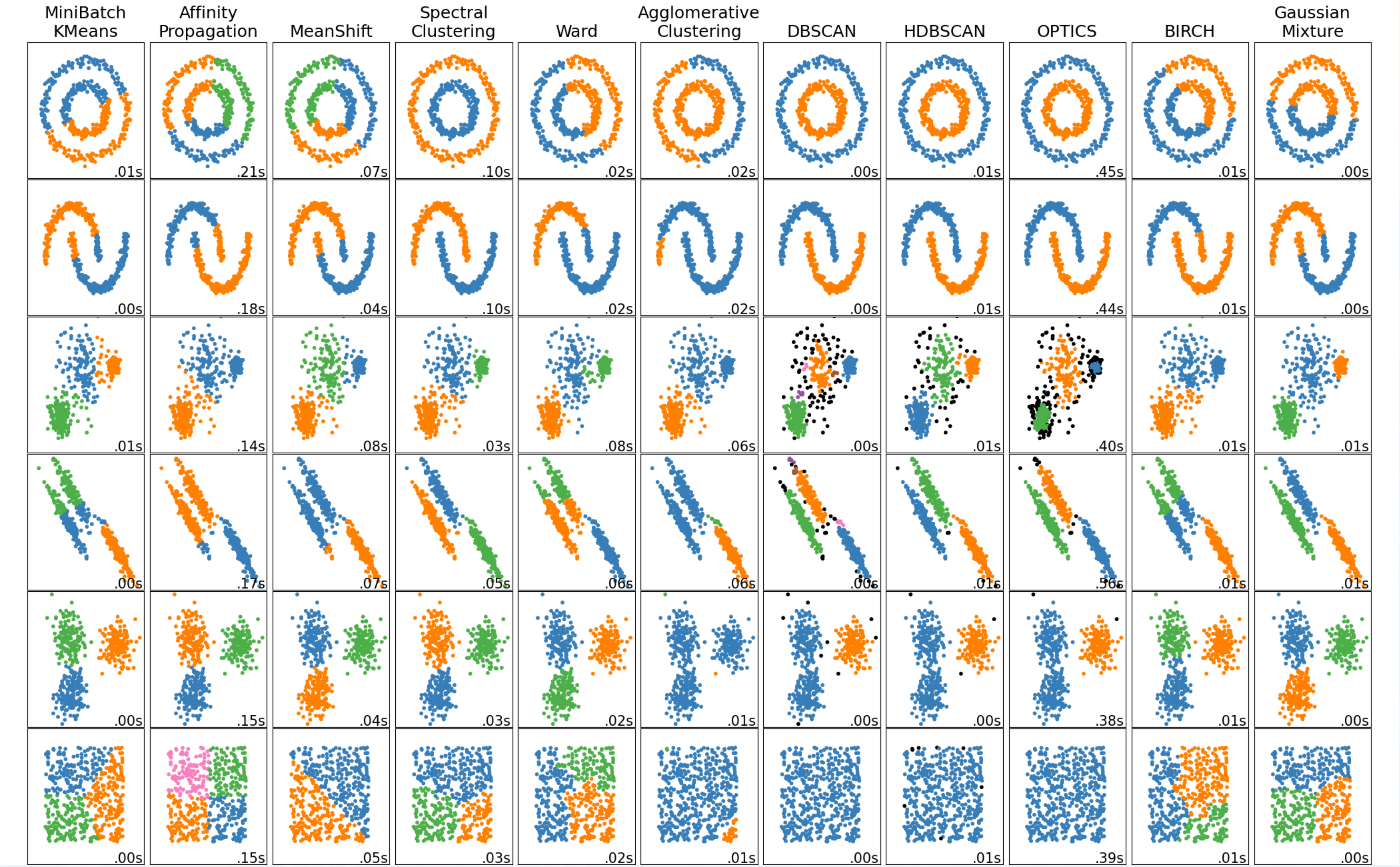
see also

`Walk_Through_Tree.ipynb`

for more details

Machine Learning Algorithms: Unsupervised Learning

there is a lot more...



Thank you very much for your attention!