Lecture 9:

Machine Learning Overview

Markus Hohle

University California, Berkeley

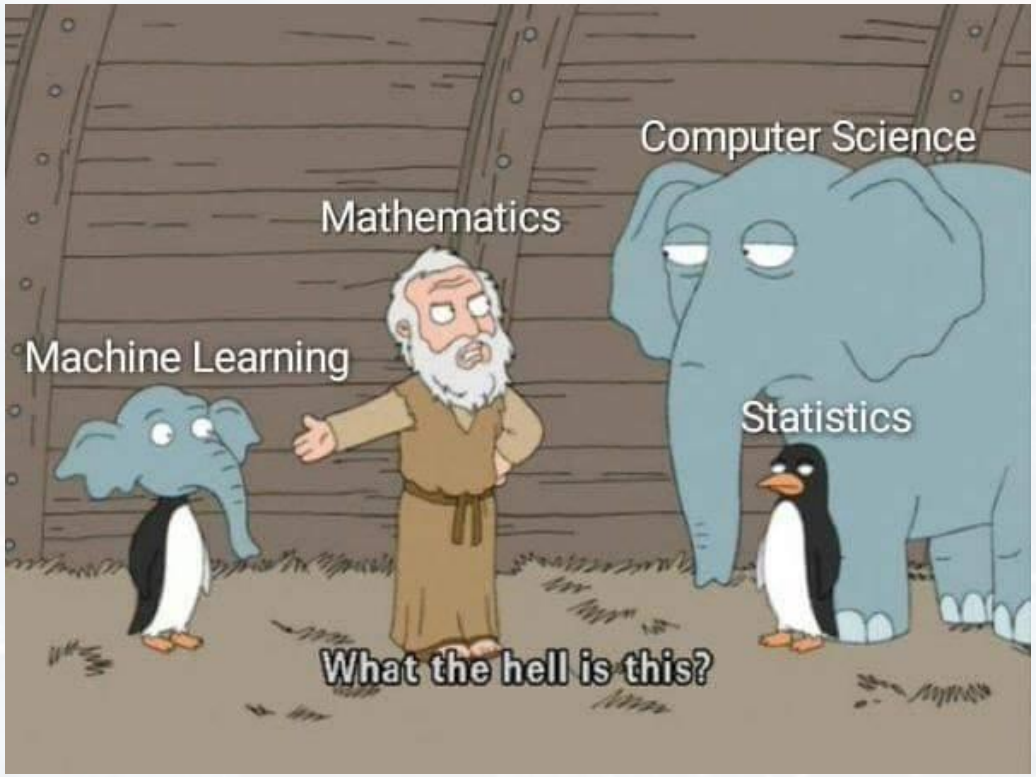
Bayesian Data Analysis and Machine Learning for Physical Sciences

Bayesian Data Analysis and Machine Learning for Physical Sciences

Course Map

Module 1	Maximum Entropy and Information, Bayes Theorem
Module 2	Naive Bayes, Bayesian Parameter Estimation, MAP
Module 3	MLE, Lin Regression
Module 4	Model selection I: Comparing Distributions
Module 5	Model Selection II: Bayesian Signal Detection
Module 6	Variational Bayes, Expectation Maximization
Module 7	Hidden Markov Models, Stochastic Processes
Module 8	Monte Carlo Methods
Module 9	**Machine Learning Overview, Supervised Methods & Unsupervised Methods**
Module 10	ANN: Perceptron, Backpropagation, SGD
Module 11	Convolution and Image Classification and Segmentation
Module 12	RNNs and LSTMs
Module 13	RNNs and LSTMs + CNNs
Module 14	Transformer and LLMs
Module 15	Graphs & GNNs

Machine Learning Overview, Supervised Methods & Unsupervised Methods



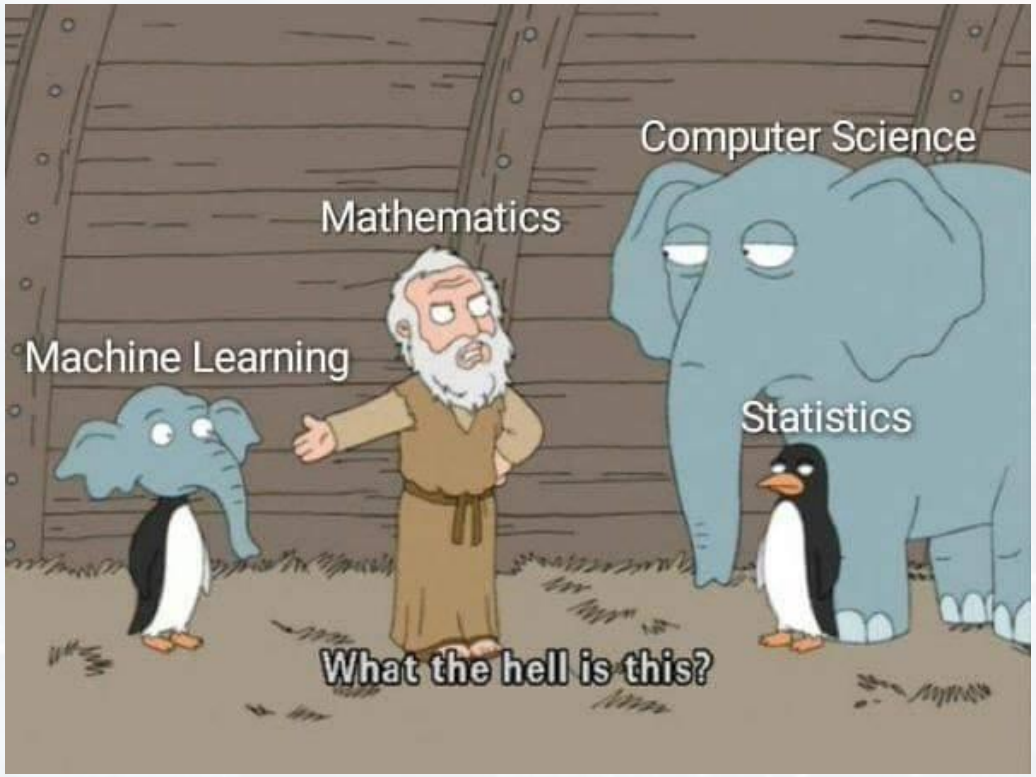
Outline

General Overview

Methods we haven't discussed yet

- Support Vector Machine
- Trees

Machine Learning Overview, Supervised Methods & Unsupervised Methods



Outline

General Overview

Methods we haven't discussed yet

- Support Vector Machine
- Trees

ML Overview

	supervised	unsupervised
regression	linear regression ✓ ANNs (soon)	
classification	logistic regression ✓ K-nearest (homework) ✓ **SVM (today)** ANNs (soon)	GMM ✓ K-means ✓ HMM ✓ **Trees (today)**

Workflow

supervised

1) creating the model:

```
my_model = library.method(argument1 = 'arg1', … )
```

2) training the model

```
out = my_model.fit(xtrain, ytrain)
```

3) evaluation

```
ypred = out.predict(xeval)
accur = (ypred == yeval).sum()/len(yeval)
```

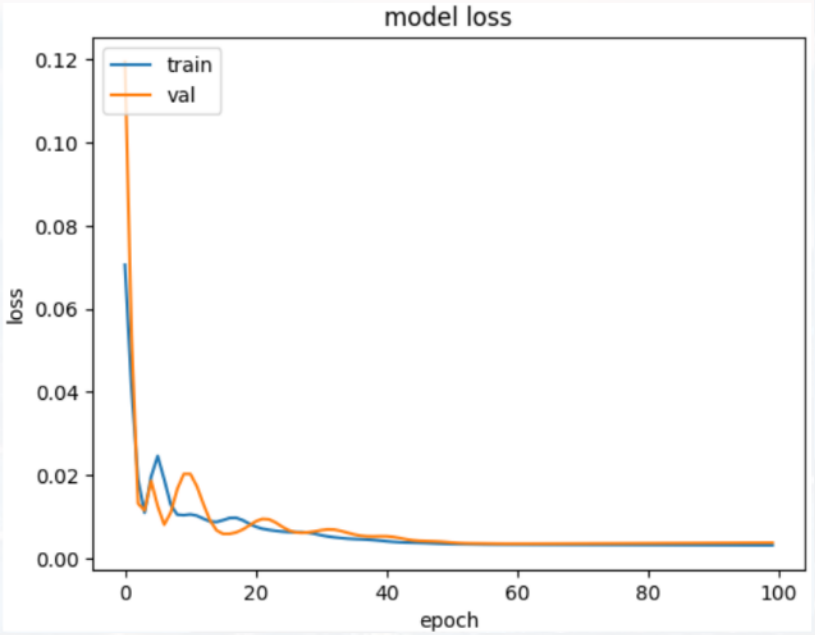
4) prediction (actual application)

```
ypred = out.predict(xnew)
```

How to split the data?

- 80% training **(60% actual training 20% evaluation)**
- 20% test

evaluation while training!



Workflow

supervised

1) creating the model:

```
my_model = library.method(argument1 = 'arg1', … )
```

2) training the model

```
out = my_model.fit(xtrain, ytrain)
```

3) evaluation

```
ypred = out.predict(xeval)
accur = (ypred == yeval).sum()/len(yeval)
```

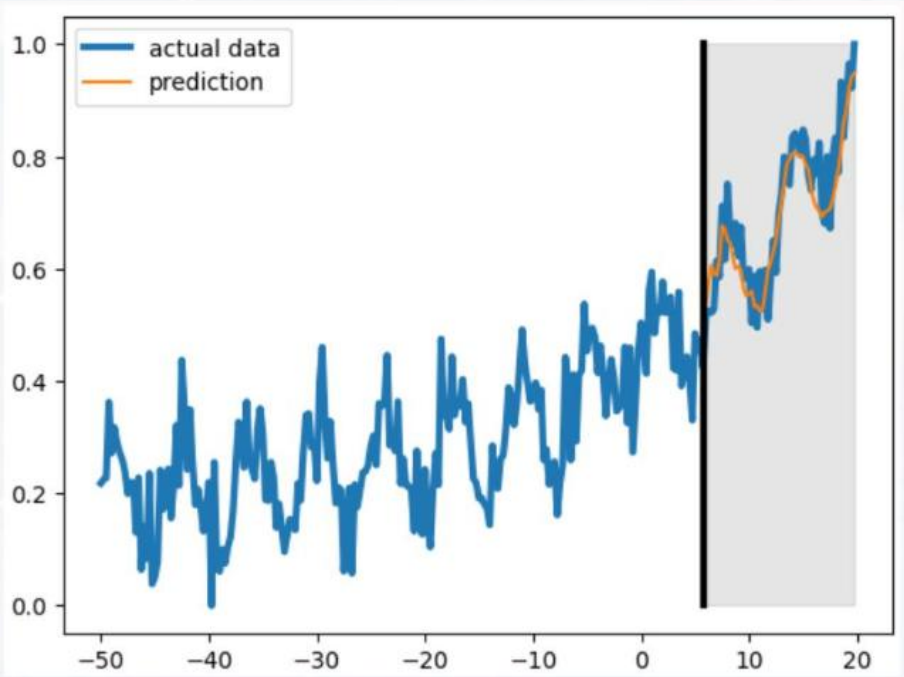
4) prediction (actual application)

```
ypred = out.predict(xnew)
```

How to split the data?

- 80% training **(60% actual training 20% evaluation)**
- 20% test

prediction after training/evaluation completed



unsupervised

Workflow

1) setting up the model

2) fitting the model: **X** are the **features** (sepal lengths/widths and petal lengths/widths),
Y are the **classes** (setosa, versicolor, virginica)

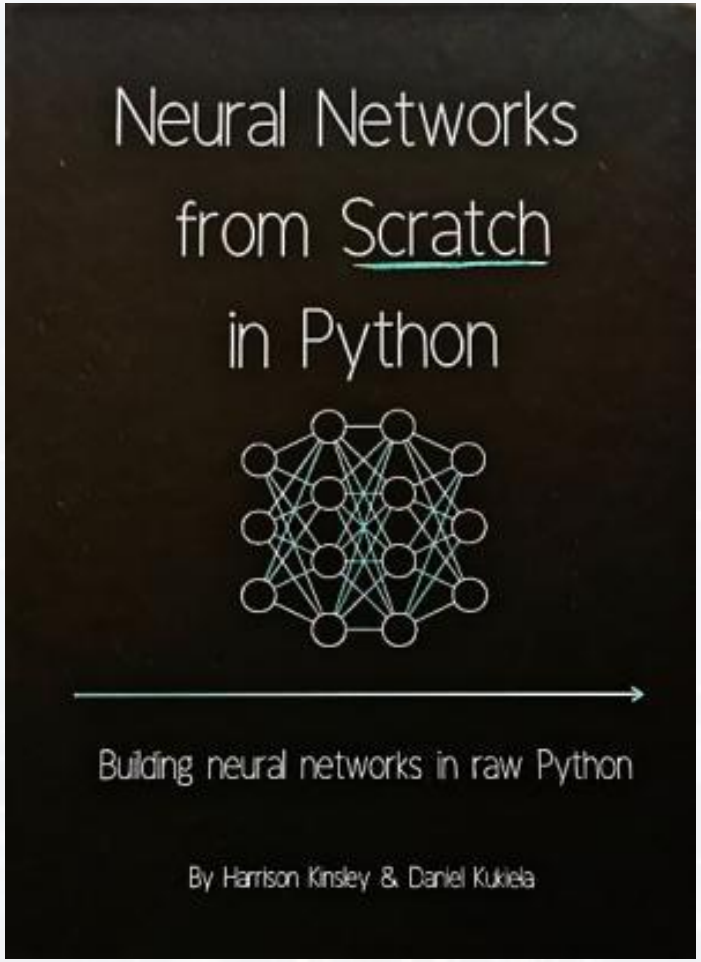
3) evaluating the model

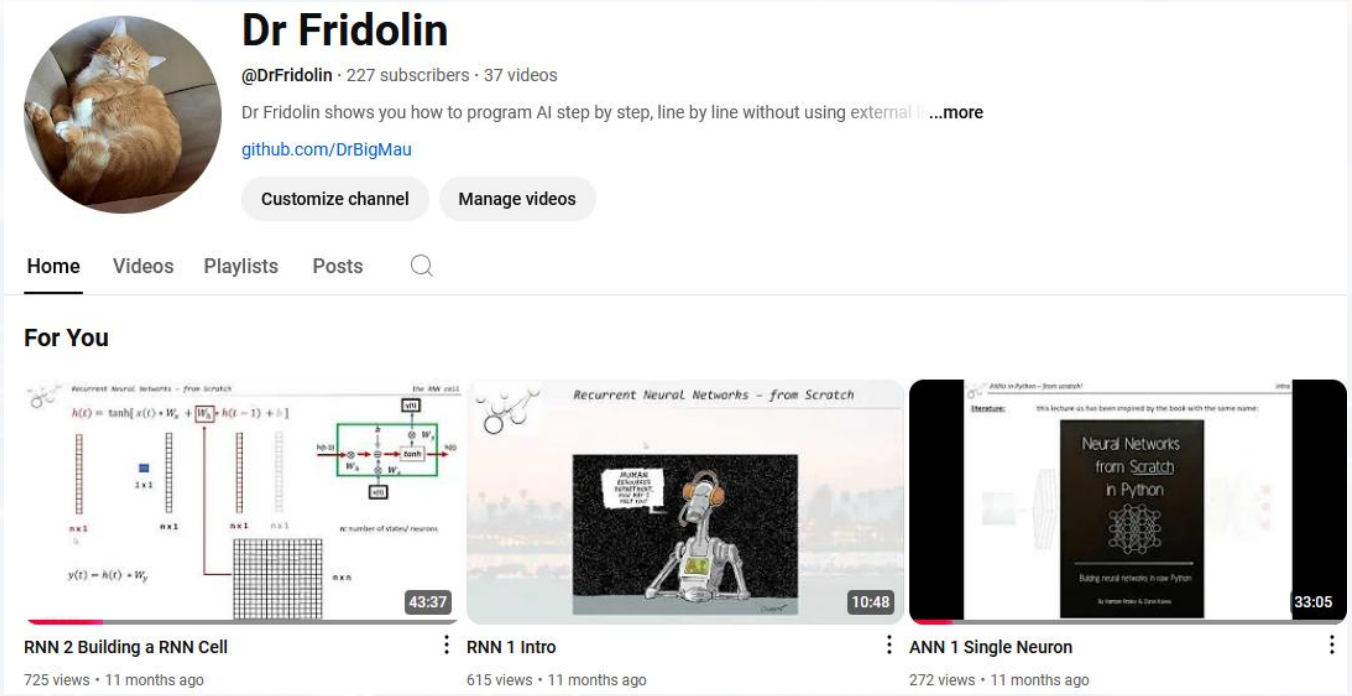
4) applying the model to a new data set

unsupervised learning:
- we don't know the classes
- sometimes we even don't know the **number *k* of classes**

resources for the next lectures to come:

Tutorials and Books





building ANNs, RNNs, LSTMs, CNNs **from scratch**
(only numpy and `matplotlib`)

Tutorials and Books

examples and **application** (CNN/RNN/LSTM and more):

Jason Brownlee
Machine Learning Mastery

all about **transformers, LLM/NLP and way more**

Andrej Karpathy

@AndrejKarpathy · 451K subscribers · 14 videos

FAQ ...more

karpathy.ai and 2 more links

Subscribe

Mısra Turp

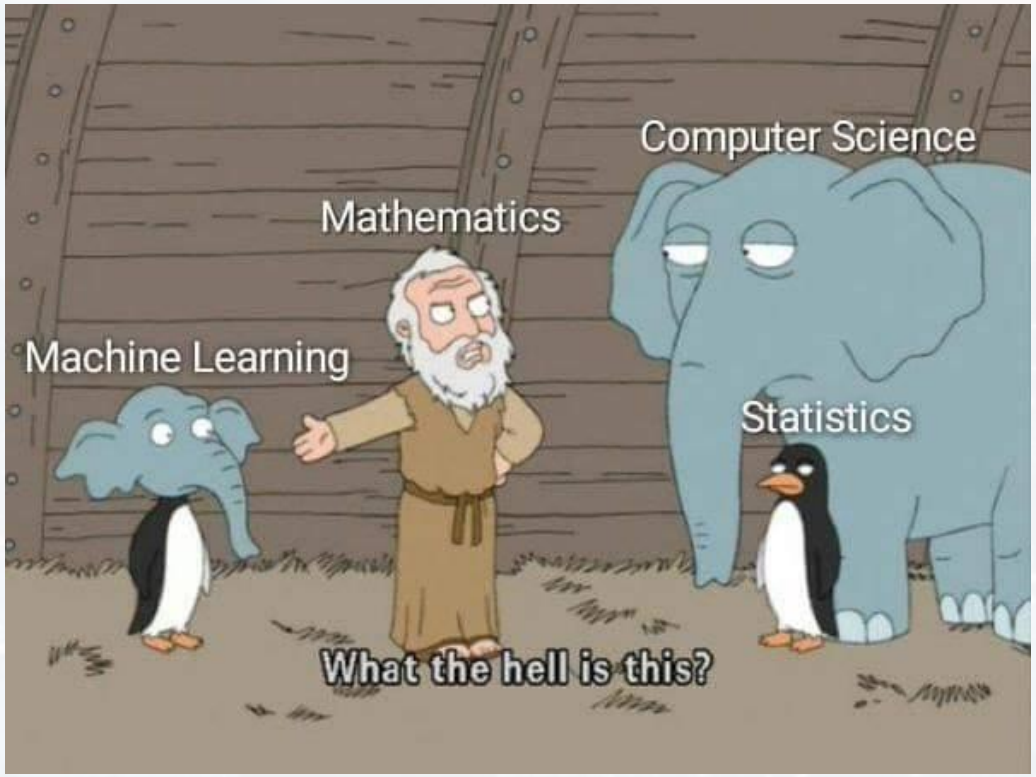
@misraturp · 38.2K subscribers · 163 videos

Here is where we learn! This is a space to take it slow

misraturp.com/roadmap and 3 more links

Subscribe

Machine Learning Overview, Supervised Methods & Unsupervised Methods



Outline

General Overview

Methods we haven't discussed yet

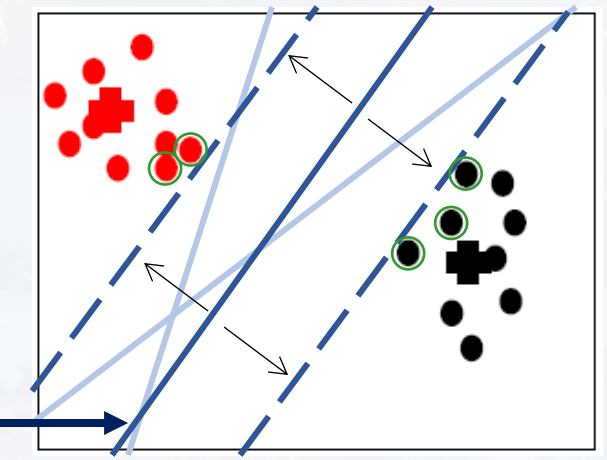
- Support Vector Machine
- Trees

SVM = **S**upport **V**ector **M**achine

idea:

1) finds best **linear** classifier for separating **two** classes by **maximizing margin** using **support vectors**
2) assign new data points to these categories
3) **supervised** learning
4) uses the "kernel trick"

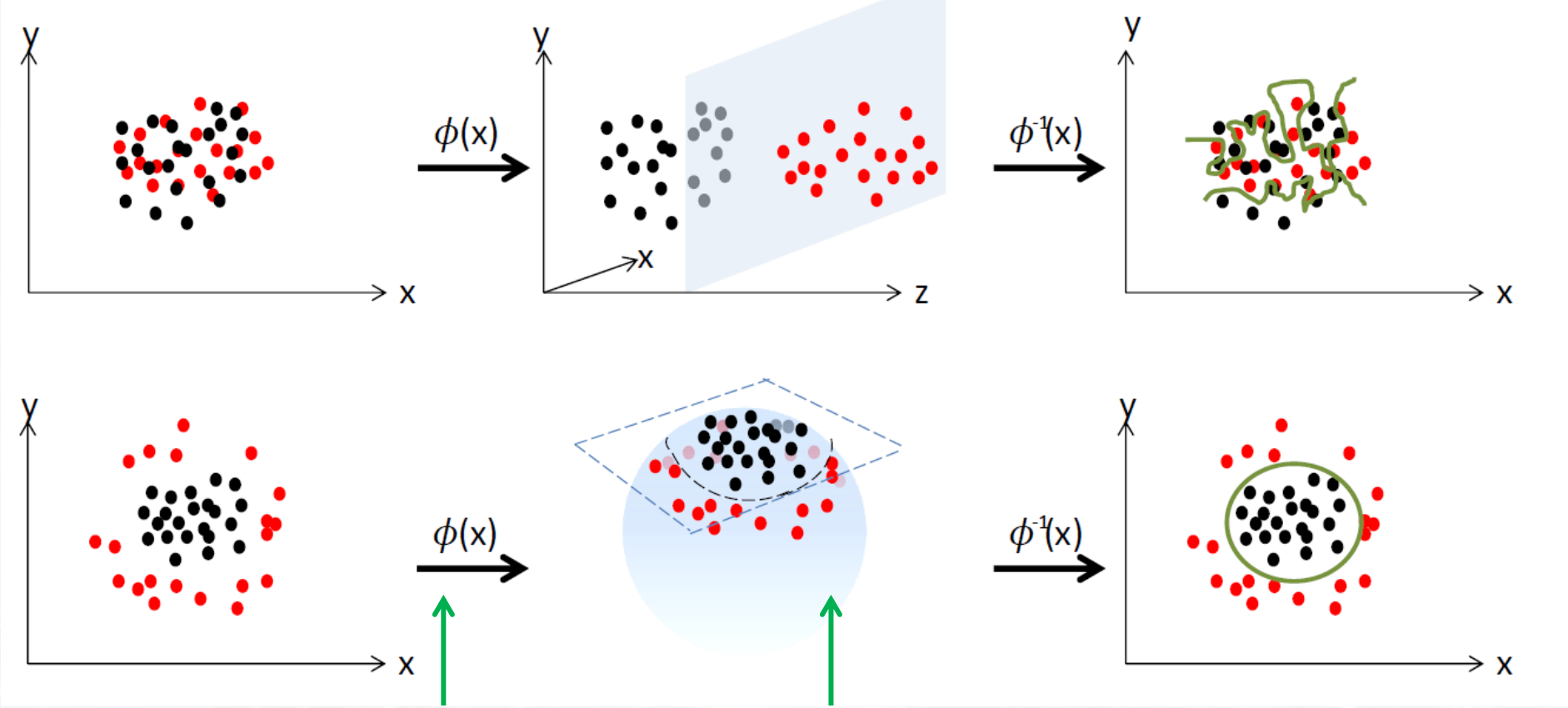
- support vectors (data points at the edge)



1) What if linear separation is not possible?
2) Is there a multiclass SVM?

1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

idea: mapping data to higher dimensional feature space



a) data space in N-D

b) a function ϕ maps the data into a M-D (M>N) feature space

c) find hyperplane separator in feature space

d) map back into data space (separator not linear)

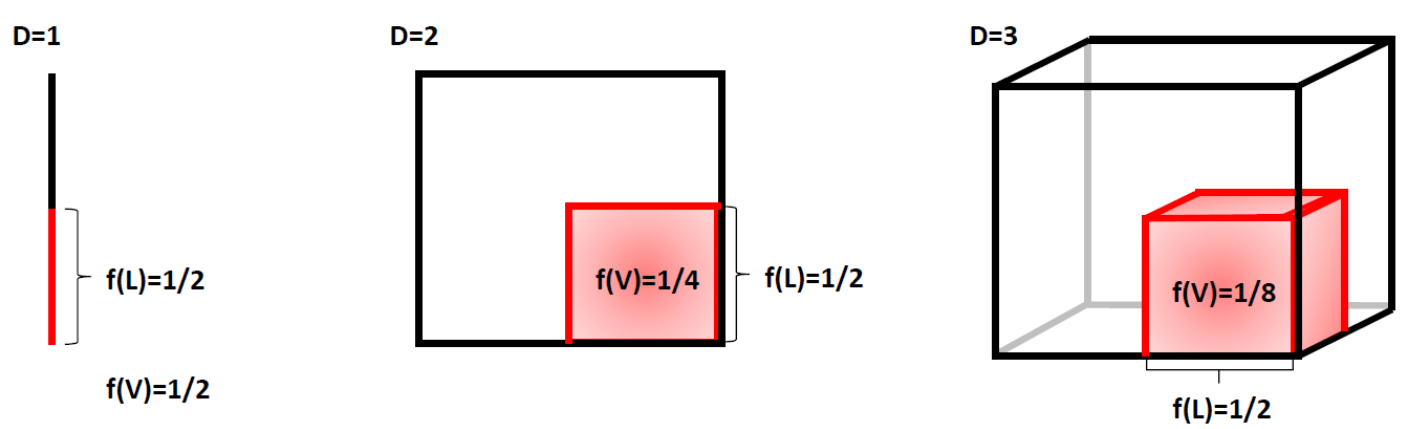
1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

problems: - computationally intensive
- ϕ usually unknown

…and dimensionality!!

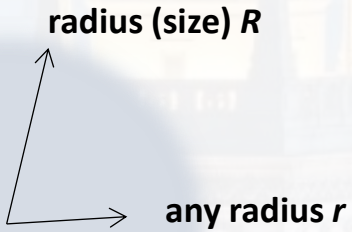
1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

What is the fraction of volume f(V) covered by a certain fraction of length f(L) for different dimensions D?



answer: $\boldsymbol{f(V) = f(L)^D}$

N - D space



hypersphere:

$$V_D(r) = C(D)\, r^D$$

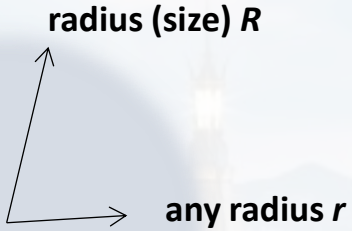
$C(D)$: constant that only depends on D

fraction of volume between r and $r - dr$

$$\frac{V_D(r) - V_D(r - dr)}{V_D(r)} = 1 - \frac{(r - dr)^D}{r^D}$$

1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

N - D space



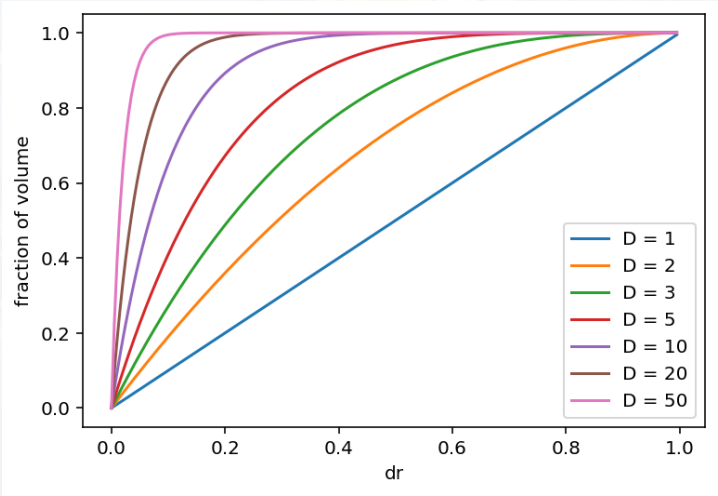
hypersphere:

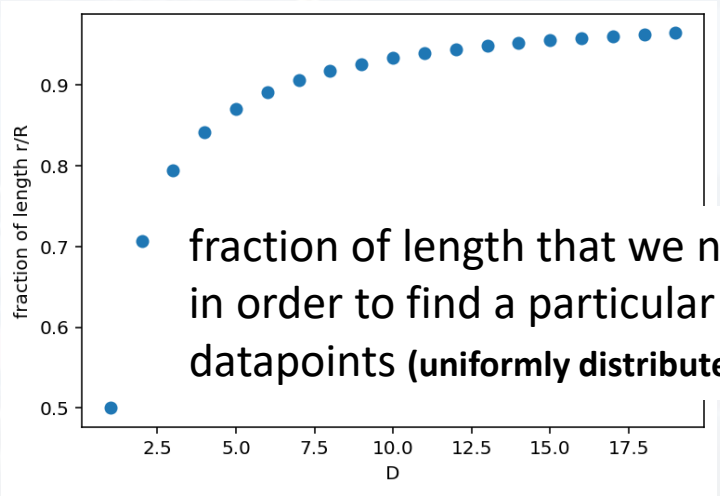
$$V_D(r) = C(D)\, r^D$$

$C(D)$: constant that only depends on D

fraction of volume between r and $r - dr$

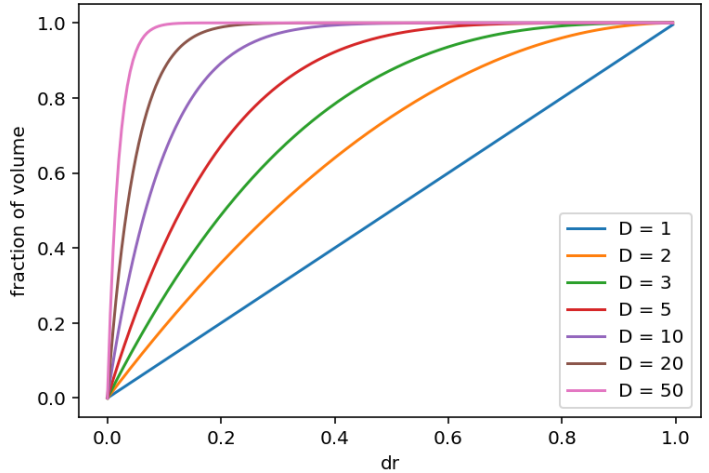
$$\frac{V_D(r) - V_D(r - dr)}{V_D(r)} = 1 - \frac{(r - dr)^D}{r^D}$$

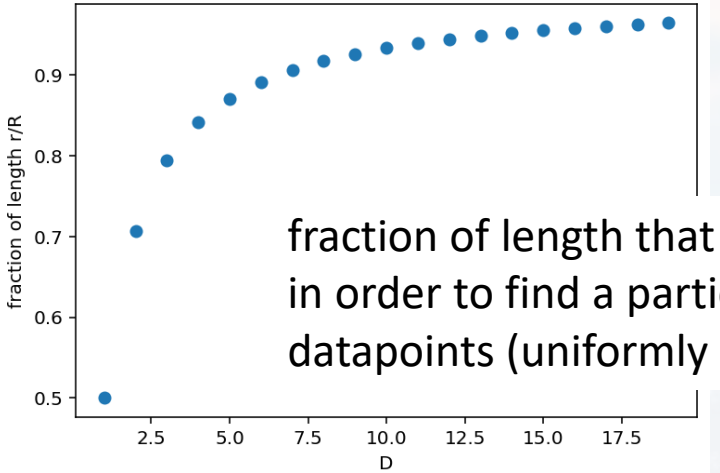




- fraction of length that we need to explore in order to find a particular fraction of datapoints **(uniformly distributed in V)**

1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?





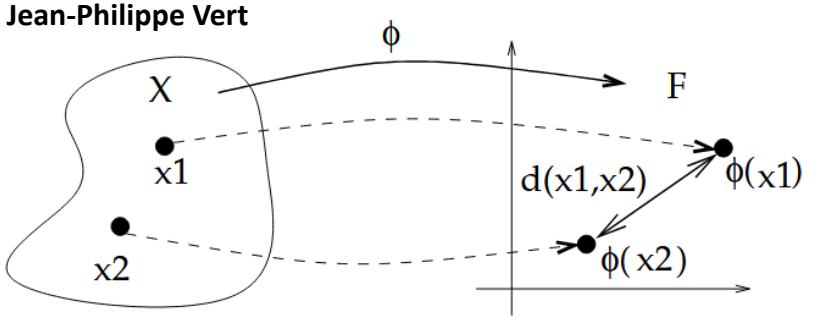
fraction of length that we need to explore in order to find a particular fraction of datapoints (uniformly distributed in R)

- for **large D**, one has to explore a **larger fraction $\frac{\rho}{R}$ of the data space** in order to get the **same fraction of data points**
- many algorithms get **less efficient** for large D
- for D→∞, the entire volume is located on the surface of the hyperspace

1) What if linear separation is not possible?

2) Is there a multiclass SVM?

Jean-Philippe Vert



problems: - computationally intensive
- ϕ usually unknown

idea: - entire mathematical framework not needed
- for separation: need distances *d* in data space and feature space

$$d^2(x,y) = \langle x-y, x-y\rangle$$

$$d_\phi^2\big(\phi(x),\phi(y)\big) = \langle \phi(x)-\phi(y), \phi(x)-\phi(y)\rangle$$

$$= \langle \phi(x),\phi(x)\rangle - 2\langle \phi(x),\phi(y)\rangle + \langle \phi(y),\phi(y)\rangle$$

kernel $\mathrm{K}(x,y) := \langle \phi(x),\phi(y)\rangle$

$$d_\phi^2\big(\phi(x),\phi(y)\big) = K(x,x) - 2K(x,y) + K(y,y)$$

kernel trick: - we don't know *K* either: **we guess it!**

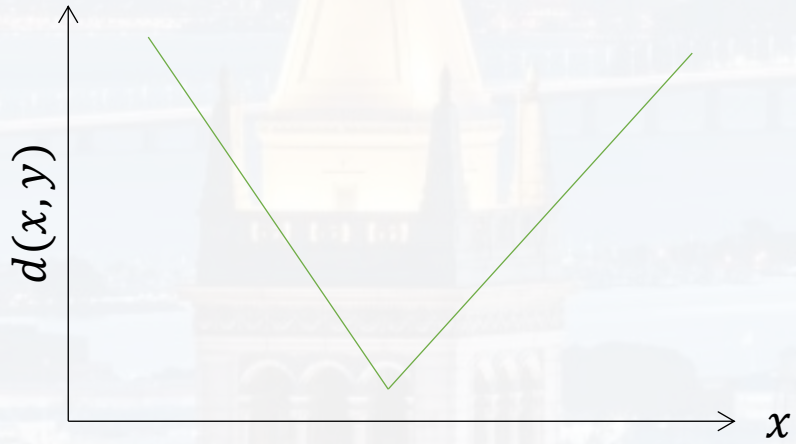
1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

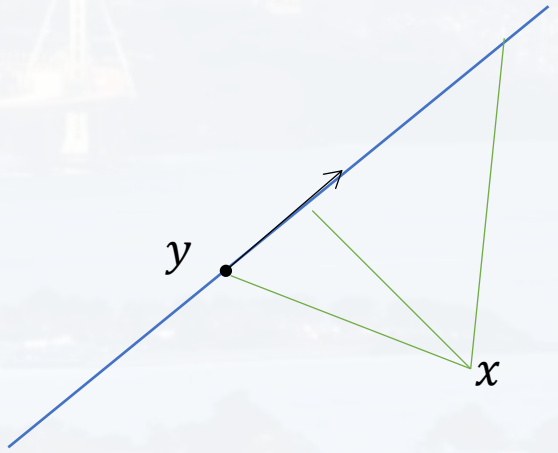
$$d^2\big(\phi(x),\phi(y)\big) = K(x,x) - 2K(x,y) + K(y,y)$$

example: linear kernel ϕ: x ↦ x $\phi(x) = x$

$$d_\phi^2(x,y) = \langle x,x\rangle - 2\langle x,y\rangle + \langle y,y\rangle = x^2 - 2xy + y^2 = (x-y)^2$$

$$d_\phi^2(x,y) = |x-y|$$

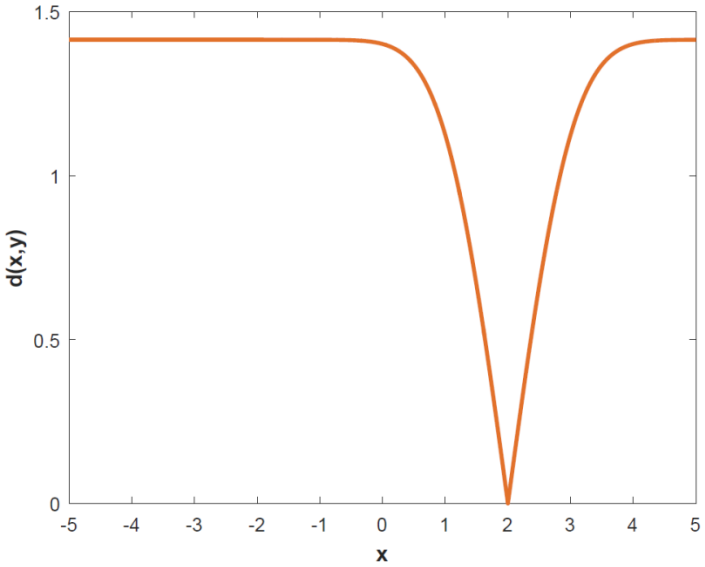


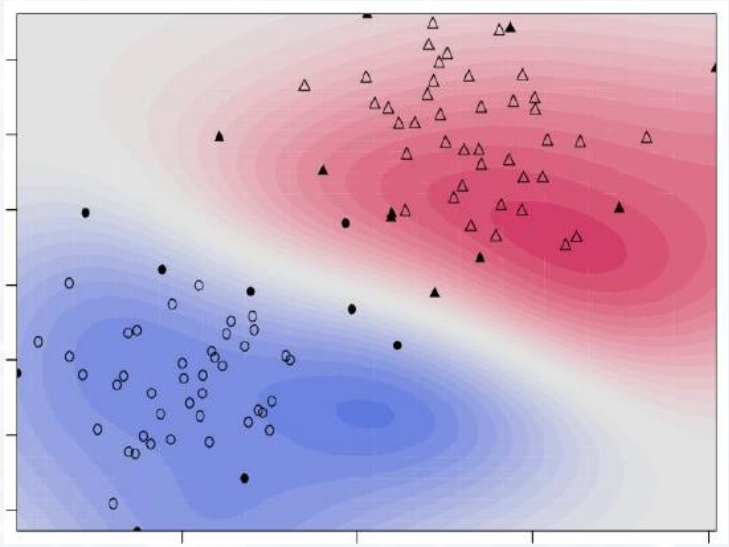


1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

example: : RBF (**r**adial **b**asis **f**unction) $\phi: x \mapsto \varsigma$ $\phi(x) = \sim\exp(\sim x)$

$$K(x,y) = exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \qquad d_\phi^2(\phi(x),\phi(y)) = 2\left[1 - exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)\right]$$





1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

kernels available in `sklearn`:

- linear: $K(x,y) = \|x - y\|$

- Gaussian aka RBF (**r**adial **b**asis **f**unction): $K(x,y) = exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$

in `sklearn` we can adjust $\boldsymbol{\gamma} := \frac{1}{2\sigma^2}$

- polynomial: $K(x,y) = \sum_{n=1}^{N} \|x - y\|^n$

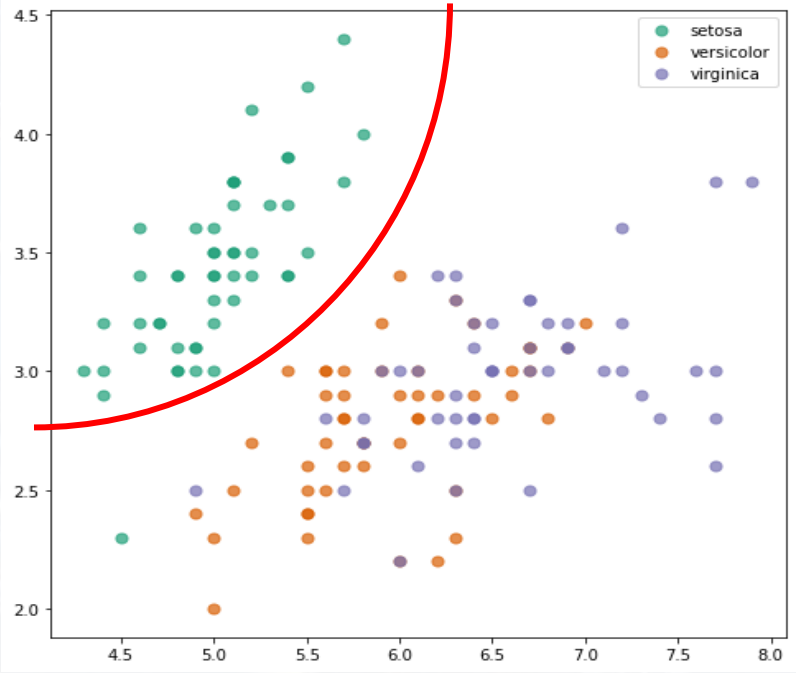
in `sklearn` we can adjust $\boldsymbol{N}$

- sigmoidal: $K(x,y) = \frac{e^{\|x-y\|}}{1 + e^{\|x-y\|}}$

1) What if linear separation is not possible?
2) **Is there a multiclass SVM?**

→ one vs rest / one vs one

green vs the rest
→ storing probabilities

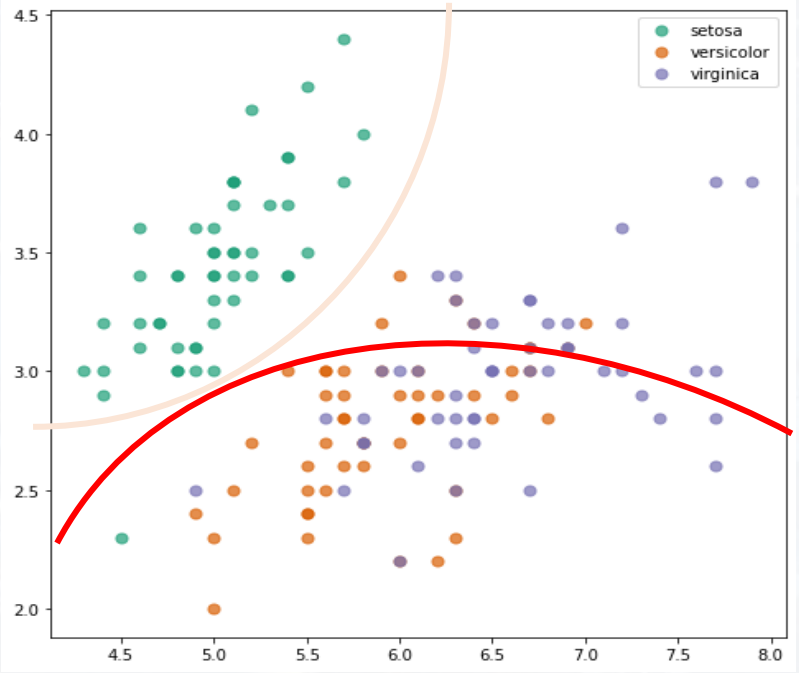


1) What if linear separation is not possible?

→ one vs rest / one vs one

2) Is there a multiclass SVM?

green vs the rest
→ storing probabilities

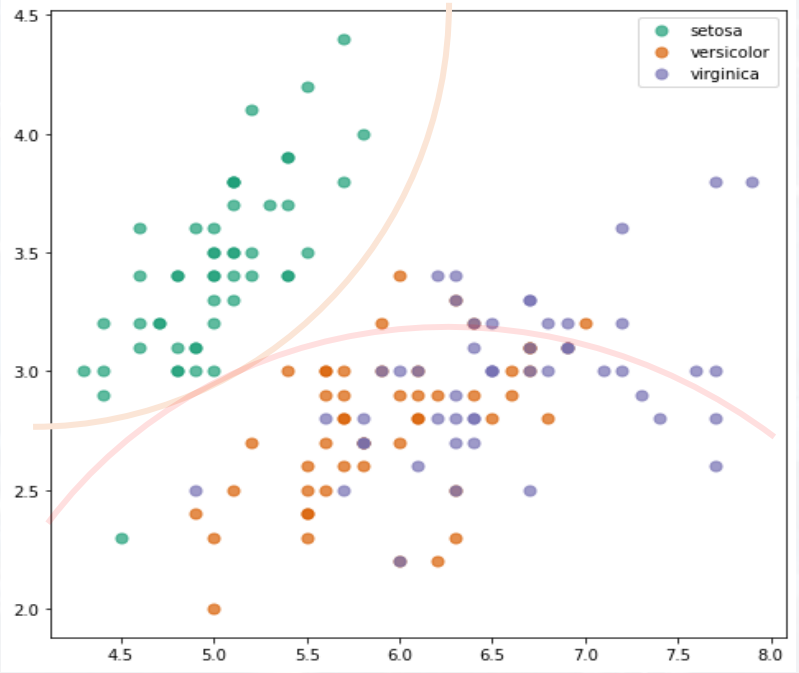


orange vs the rest
→ storing probabilities

1) What if linear separation is not possible?
2) **Is there a multiclass SVM?**

→ one vs rest / one vs one

green vs the rest
→ storing probabilities



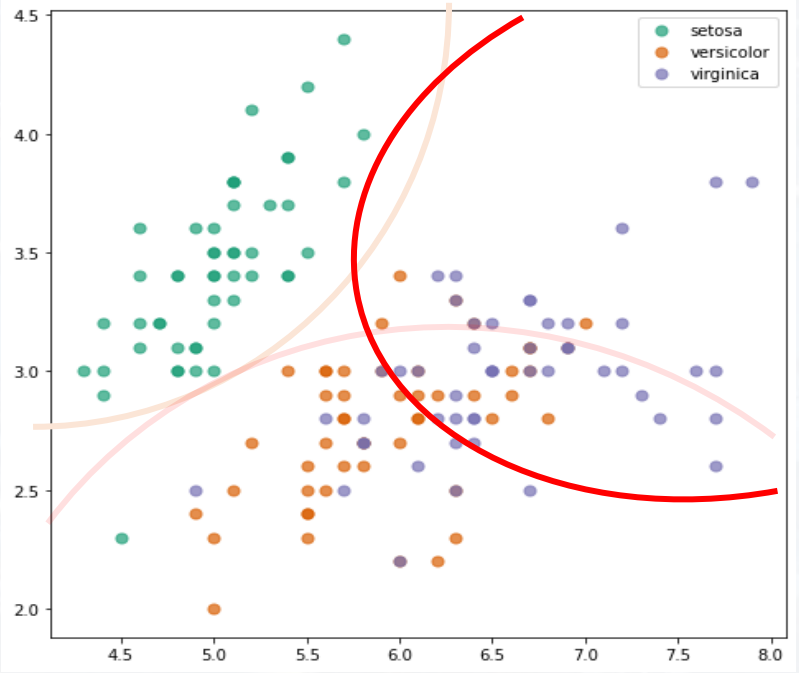
blue vs the rest
→ storing probabilities

orange vs the rest
→ storing probabilities

1) What if linear separation is not possible?
2) **Is there a multiclass SVM?**

→ one vs rest / one vs one

green vs the rest
→ storing probabilities



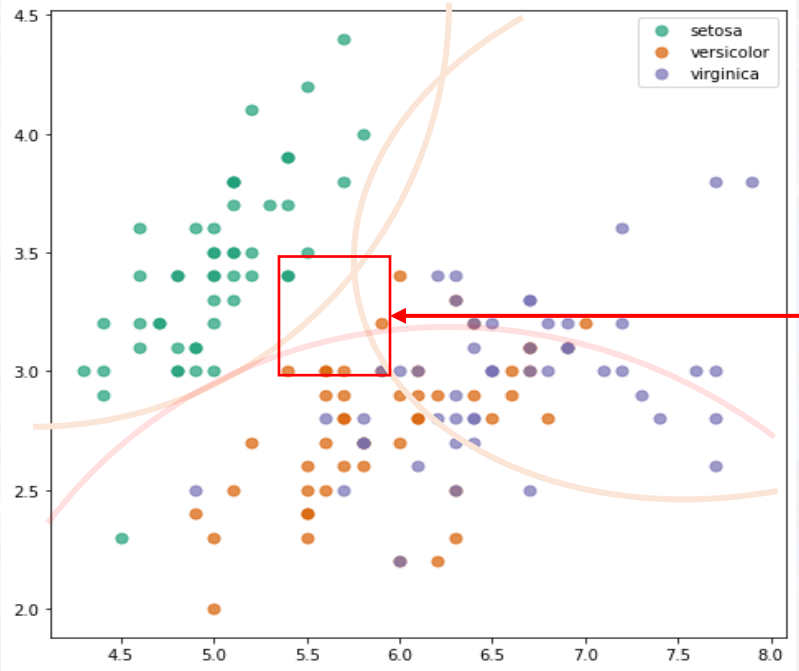
blue vs the rest
→ storing probabilities

orange vs the rest
→ storing probabilities

1) What if linear separation is not possible?
2) **Is there a multiclass SVM?**

→ one vs rest / one vs one

green vs the rest
→ storing probabilities



three probabilities for each data point
→ assign class to most probable value

k class classification with two-class discriminant functions is ambiguous!

```
from sklearn import svm
```

see **Walk_Through_SVM.ipynb**

1) setting up the model & 2) fitting the model

running analysis with different kernel

```
outlinear = svm.SVC(kernel = 'linear', C = 1, decision_function_shape = 'ovr')
linear    = outlinear.fit(X2D, Y)


outrbf    = svm.SVC(kernel = 'rbf', gamma = 1, C = 1, \
                    decision_function_shape = 'ovr')
rbf       = outrbf.fit(X2D, Y)


outpoly   = svm.SVC(kernel = 'poly', degree = 3, C = 1,\
                    decision_function_shape = 'ovr')
poly      = outpoly.fit(X2D, Y)


outsig    = svm.SVC(kernel = 'sigmoid', C = 1, decision_function_shape = 'ovr')
sig       = outsig.fit(X2D, Y)
```

one **v**ersus **r**est

L2 regularization parameter for error tolerance when calculating the classifier

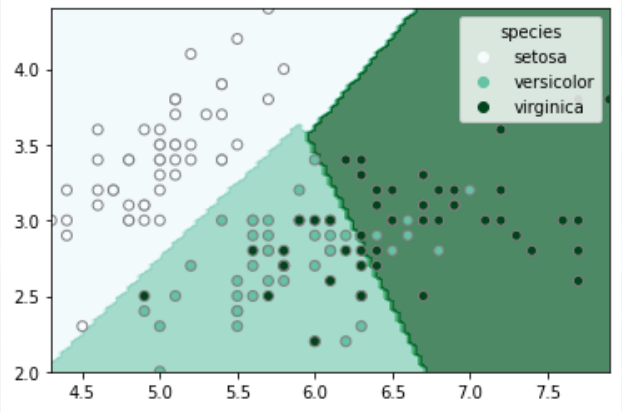
$\gamma := \frac{1}{2\sigma^2}$

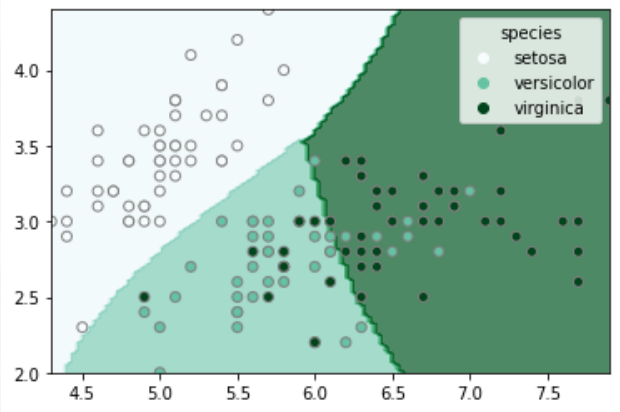
refers to *N* in $\sum_{n=1}^{N} \|x - y\|^n$

```
from sklearn import svm
```

see **Walk_Through_SVM.ipynb**

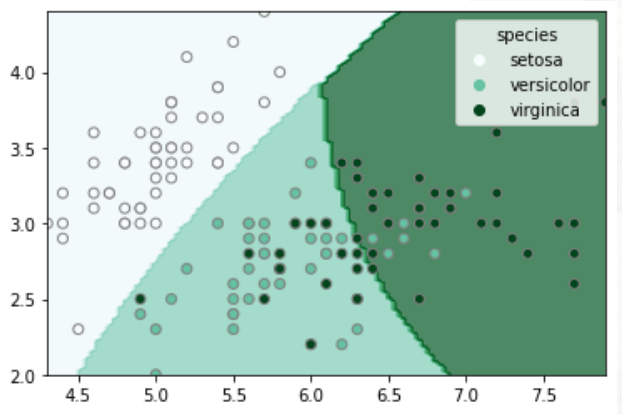
$K(x, y) = \|x - y\|$

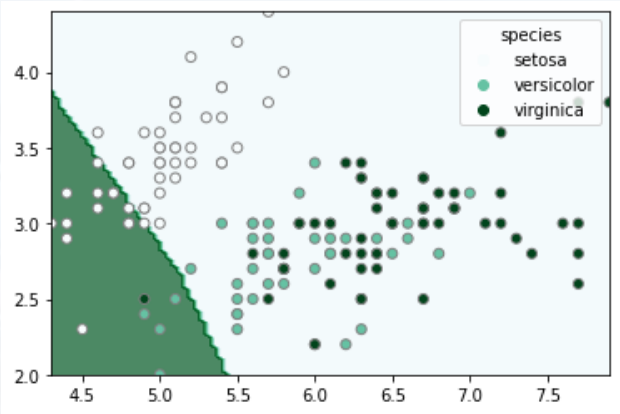




$K(x, y) = exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$

$K(x, y) = \sum_{n=1}^{N} \|x - y\|^n$



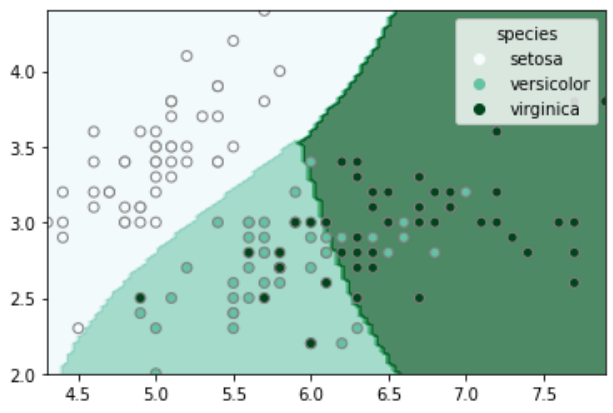


$K(x, y) = \frac{e^{\|x-y\|}}{1 + e^{\|x-y\|}}$

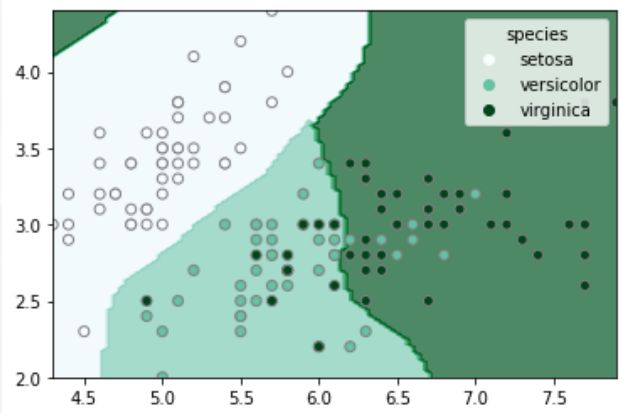
```
from sklearn import svm
```

see **Walk_Through_SVM.ipynb**

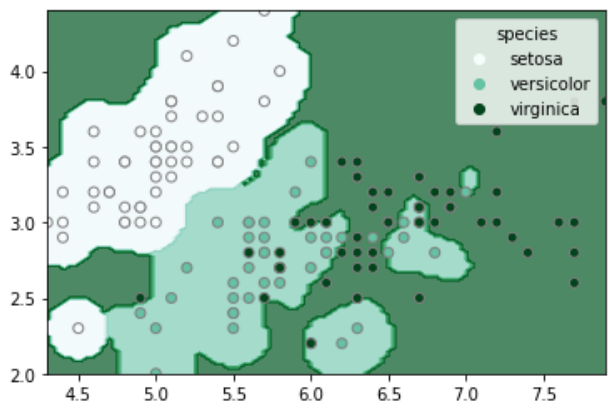
$$K(x, y) = exp\left(-\frac{1}{2\sigma^2}\|x - y\|^2\right)$$



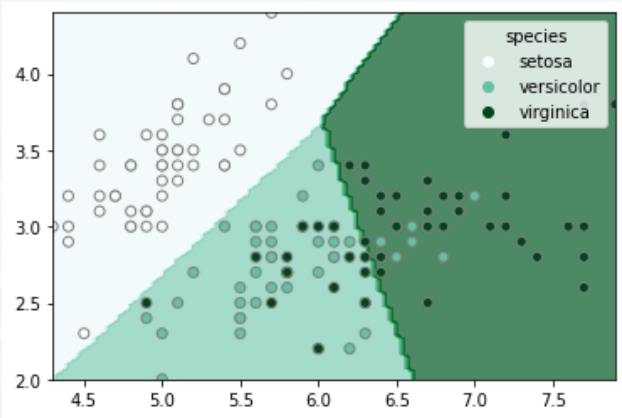
$$\gamma := \frac{1}{2\sigma^2} = 1$$



$$\gamma := \frac{1}{2\sigma^2} = 5$$



$$\gamma := \frac{1}{2\sigma^2} = 50$$

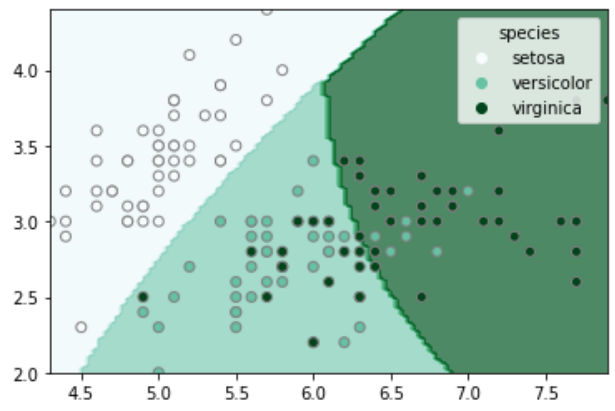


$$\gamma := \frac{1}{2\sigma^2} = 0.1$$

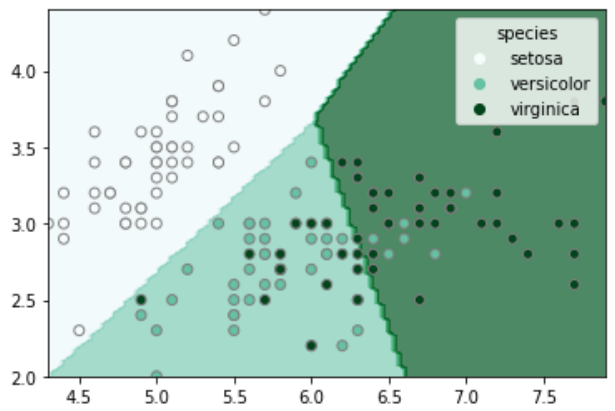
```
from sklearn import svm
```

see **Walk_Through_SVM.ipynb**

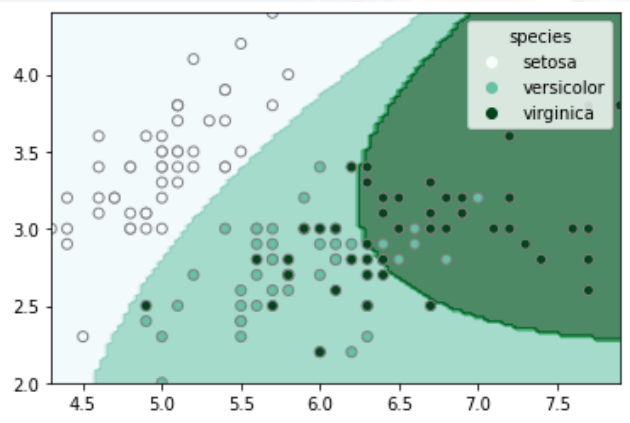
$$K(x, y) = \sum_{n=1}^{N} \|x - y\|^n$$



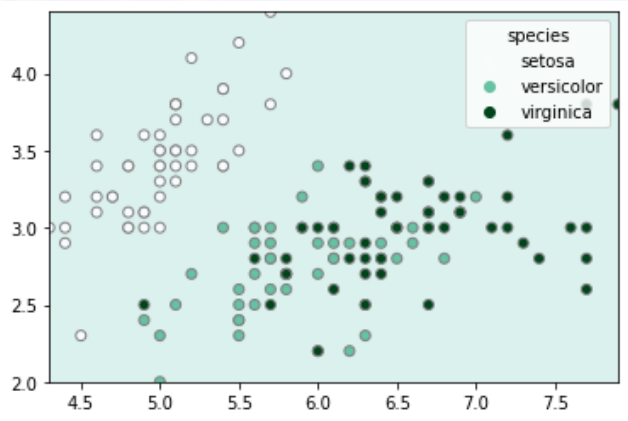
N = 3



N = 1



N = 5



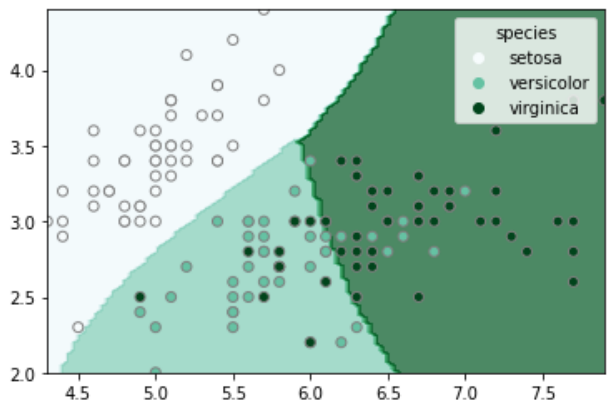
N = 0

```
from sklearn import svm
```

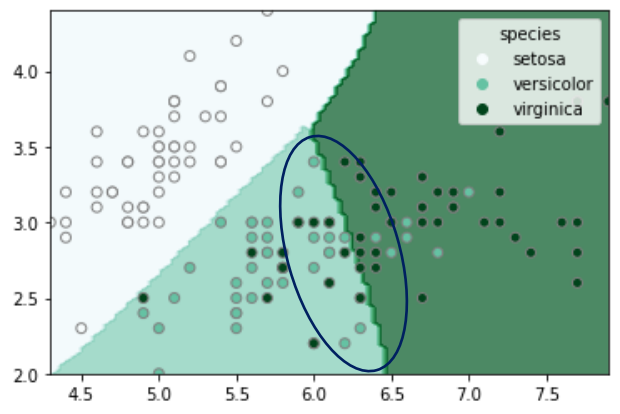
see **Walk_Through_SVM.ipynb**

$$K(x, y) = exp\left(-\frac{1}{2\sigma^2}\|x - y\|^2\right)$$

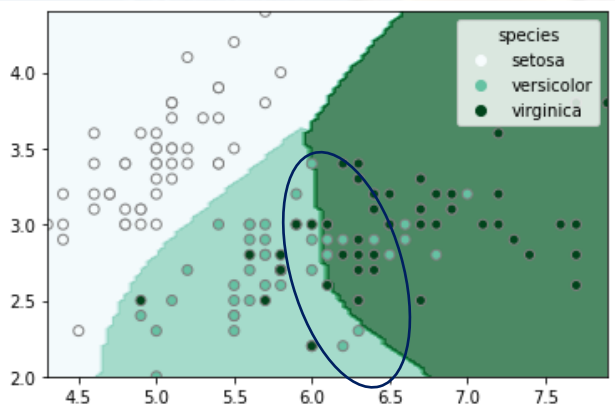
C is a L2 regularization parameter



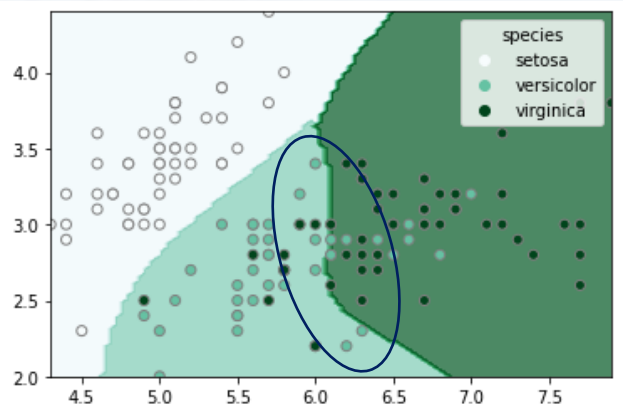
$C = 1$



$C = 0.1$



$C = 10$



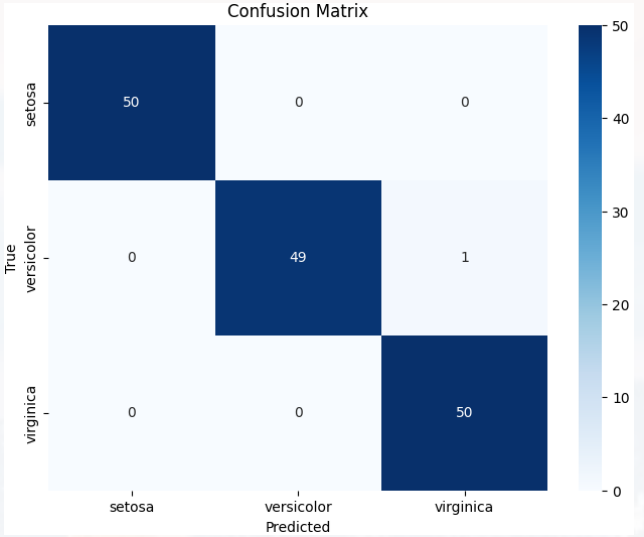
$C = 20$

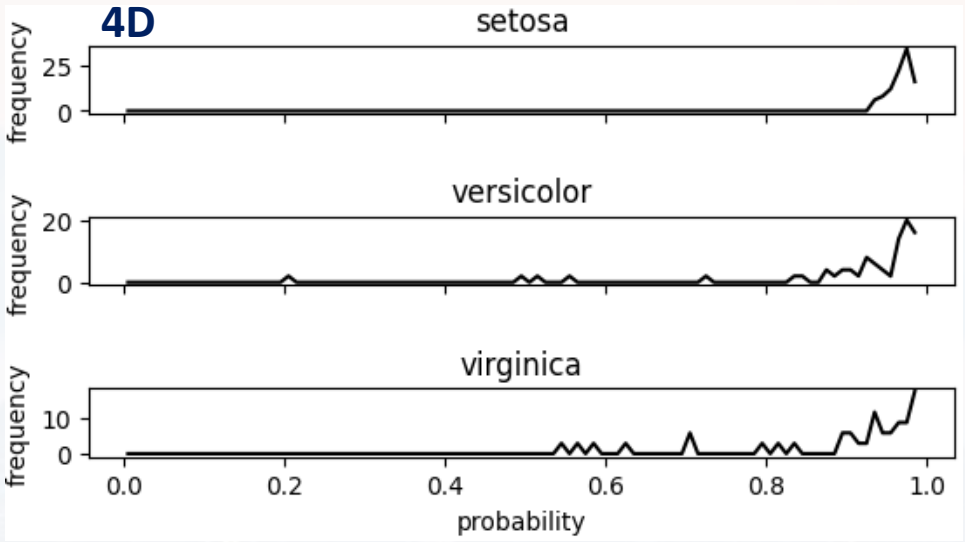
full **4D** data set

3) evaluating the model

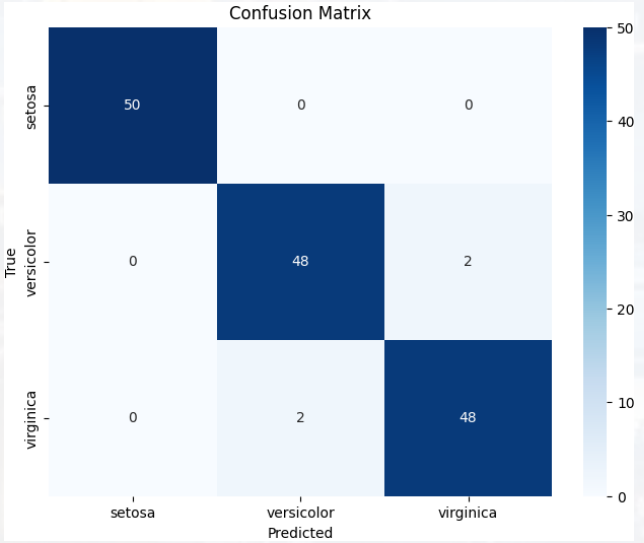
see **`Walk_Through_SVM.ipynb`**

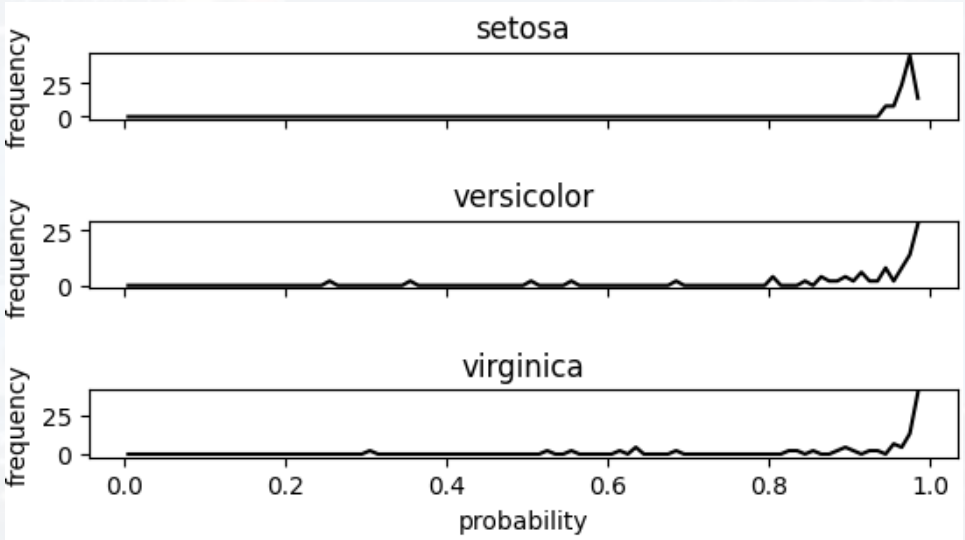
linear
accuracy: 99.3%





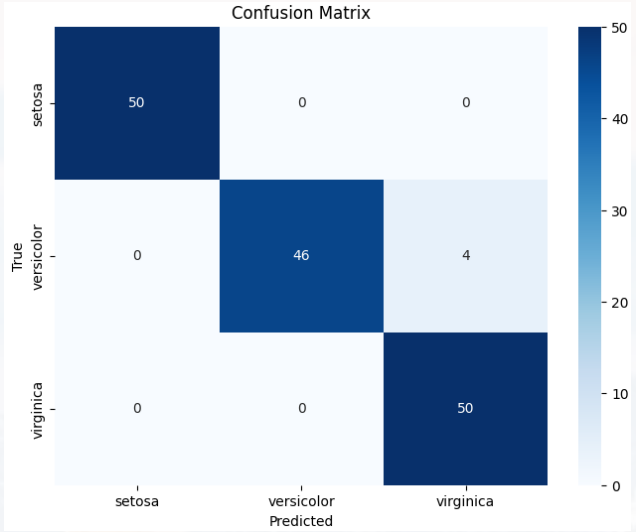
Gaussian ($\gamma = 1$)
accuracy: 97.3%





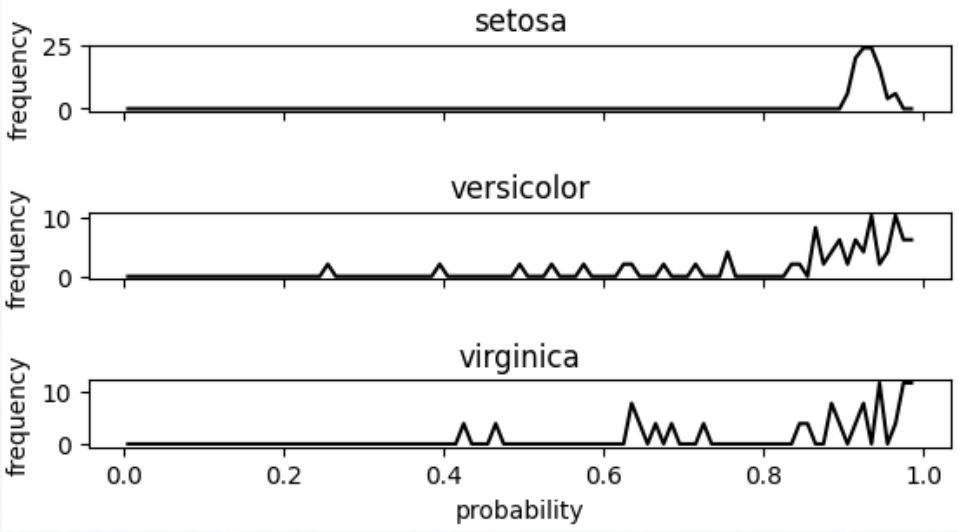
full **4D** data set

polynomial (n = 3)
accuracy: 97.3%



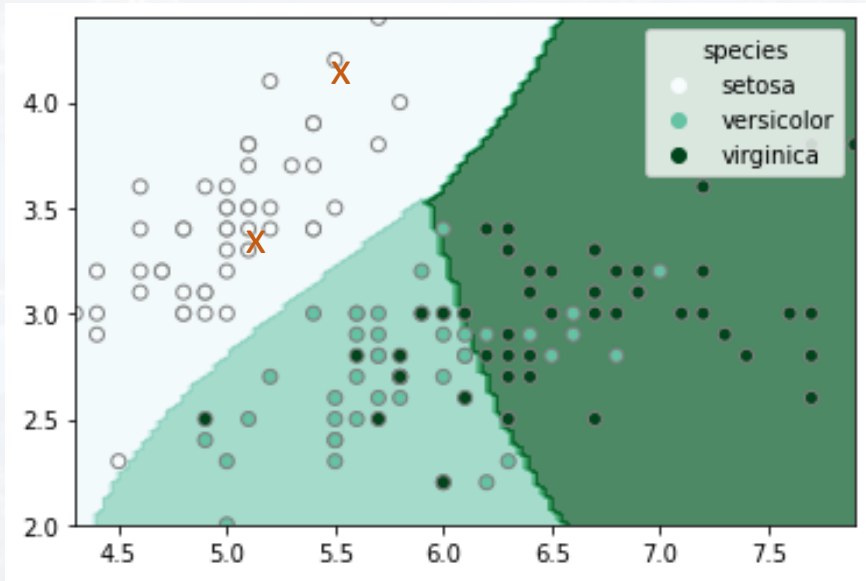
3) evaluating the model

see **`Walk_Through_SVM.ipynb`**

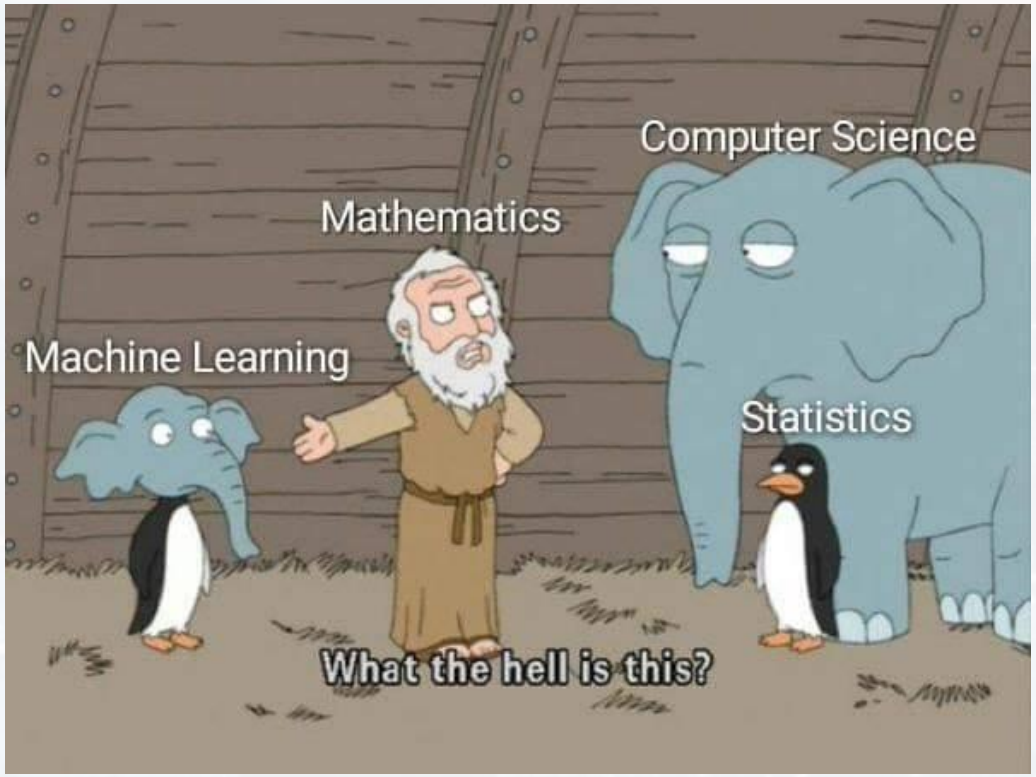


4) applying the model to a new data set

```
ypred = outpoly.predict([[6, 3.5],[6.3, 4.5]])
```



Machine Learning Overview, Supervised Methods & Unsupervised Methods

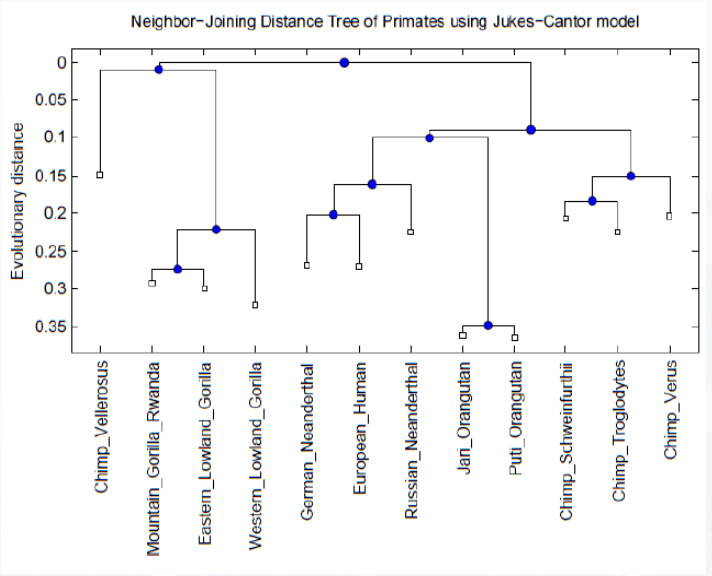


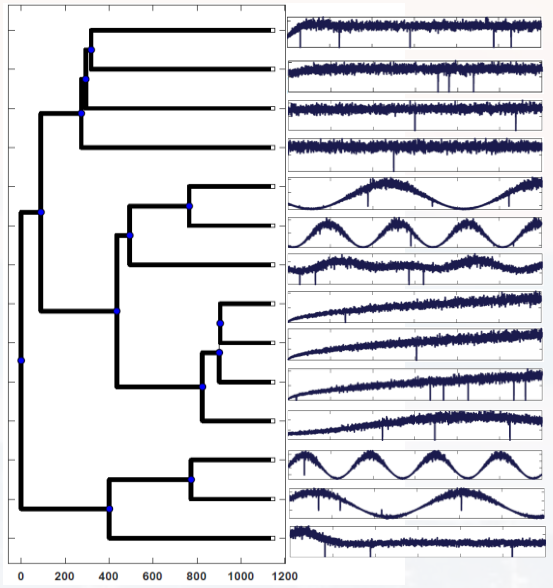
Outline

General Overview

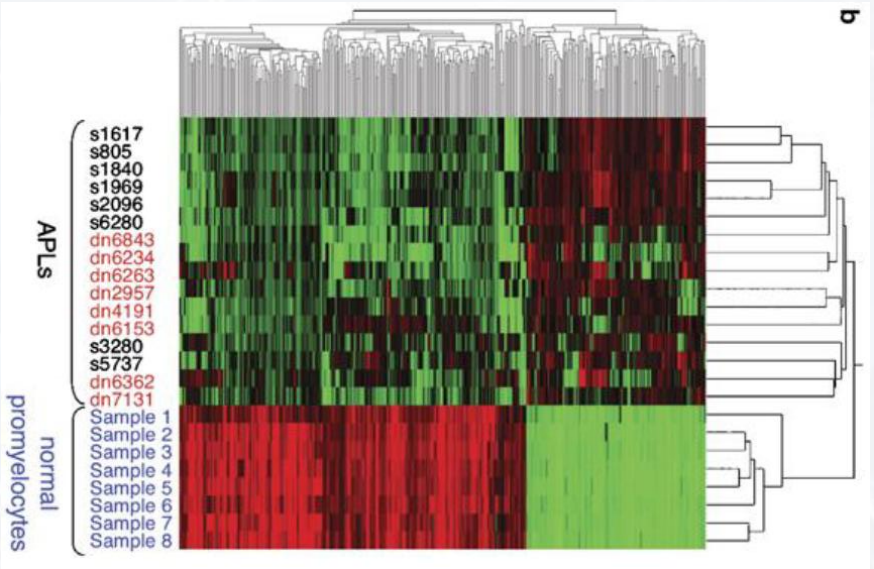
Methods we haven't discussed yet

- Support Vector Machine
- Trees

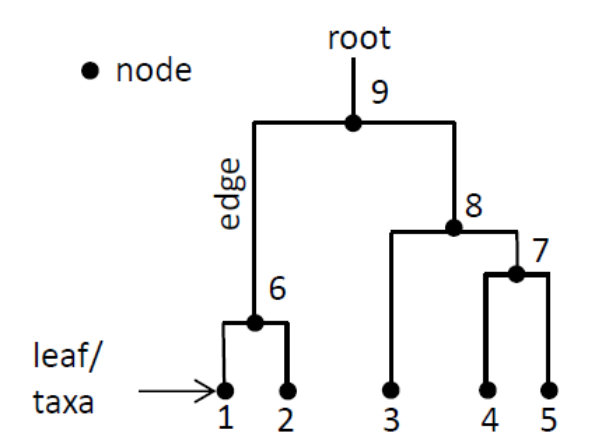




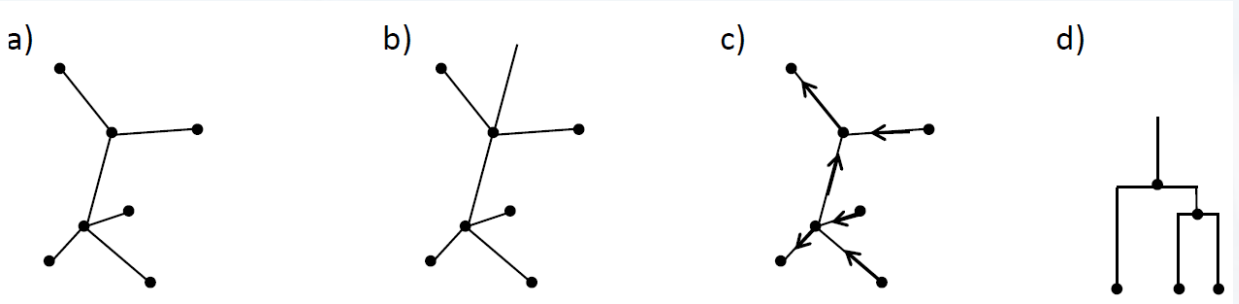
- What is a tree?
- Different kinds of trees...?
- How to build a tree?
- Why do we need trees?
- Examples...



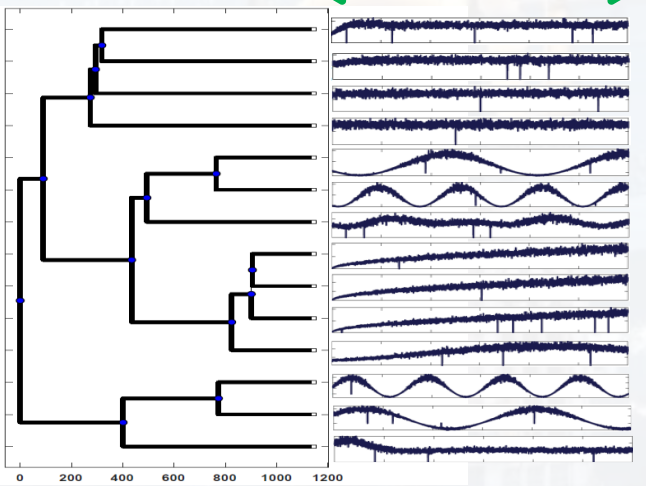
trees are a subclass of graphs **(but: not fully connected → "hierarchy", no loops)**:



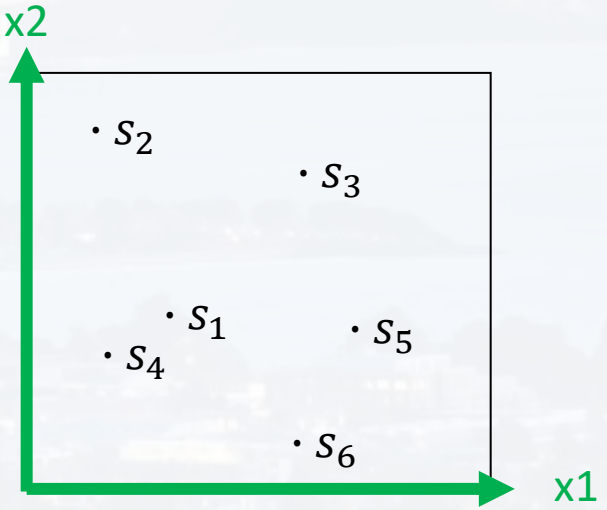
a) unrooted, undirected multinary tree
b) rooted, undirected multinary tree
c) unrooted, directed multinary tree
d) rooted, undirected binary tree



N timepoints

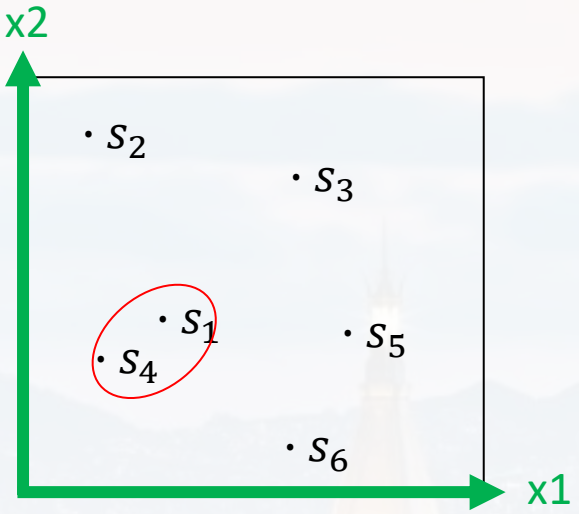


each sample s is a vector of N rows, hence, a data point in N-D



constructing trees:

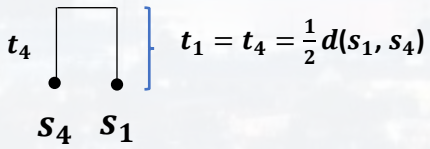
- calculating a distance between each pair of samples



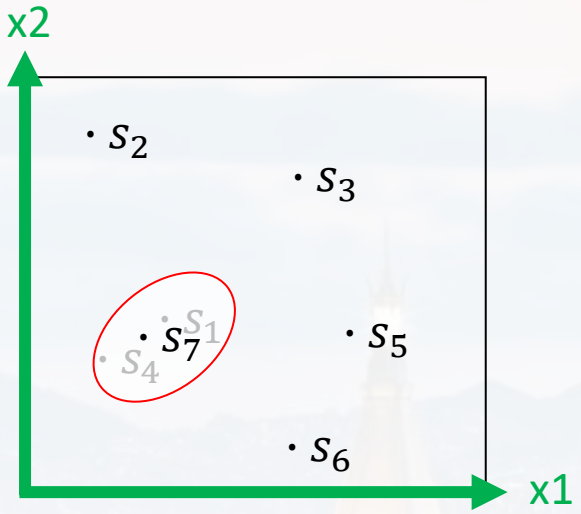
	s_1	s_2	s_3	s_4	s_5	s_6
s_1	0	$d(s_1, s_2)$	$d(s_1, s_3)$	$d(s_1, s_4)$	$d(s_1, s_5)$	$d(s_1, s_6)$
s_2		0	$d(s_2, s_3)$	$d(s_2, s_4)$	$d(s_2, s_5)$	$d(s_2, s_6)$
s_3			0	$d(s_3, s_4)$	$d(s_3, s_5)$	$d(s_3, s_6)$
s_4				0	$d(s_4, s_5)$	$d(s_4, s_6)$
s_5					0	$d(s_5, s_6)$
s_6						0

...once a distance has been defined (see soon)...

→ find the closest pair



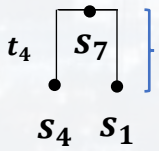
constructing trees:



- calculating a distance between each pair of samples

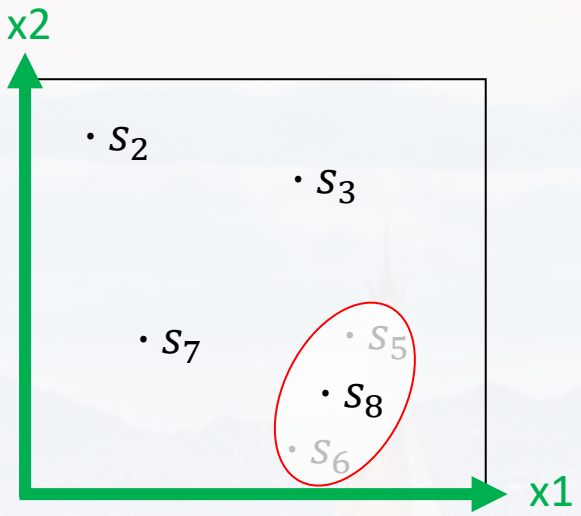
	s_1	s_2	s_3	s_4	s_5	s_6
s_1	0	$d(s_1, s_2)$	$d(s_1, s_3)$	$d(s_1, s_4)$	$d(s_1, s_5)$	$d(s_1, s_6)$
s_2		0	$d(s_2, s_3)$	$d(s_2, s_4)$	$d(s_2, s_5)$	$d(s_2, s_6)$
s_3			0	$d(s_3, s_4)$	$d(s_3, s_5)$	$d(s_3, s_6)$
s_4				0	$d(s_4, s_5)$	$d(s_4, s_6)$
s_5					0	$d(s_5, s_6)$
s_6						0

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



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

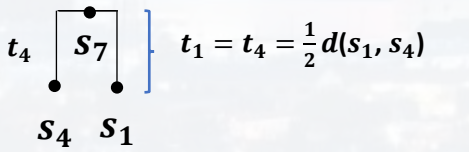
constructing trees:

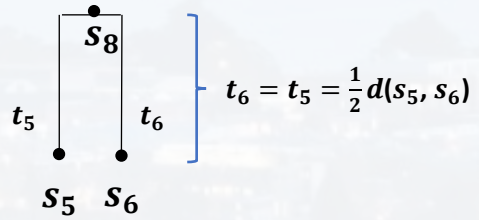


- calculating a distance between each pair of samples

	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

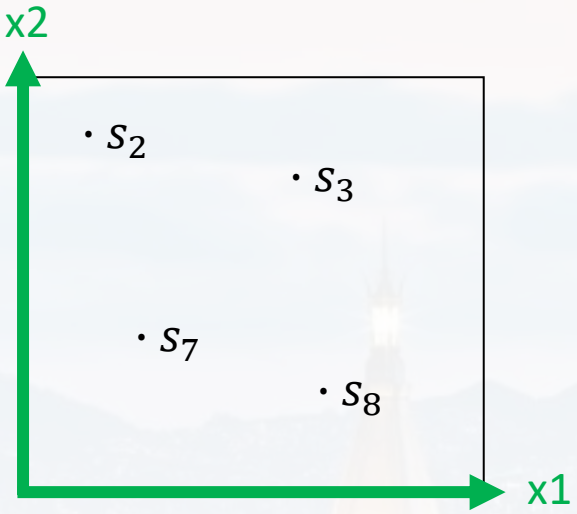
→ find the closest pair
(now including the cluster)





constructing trees:

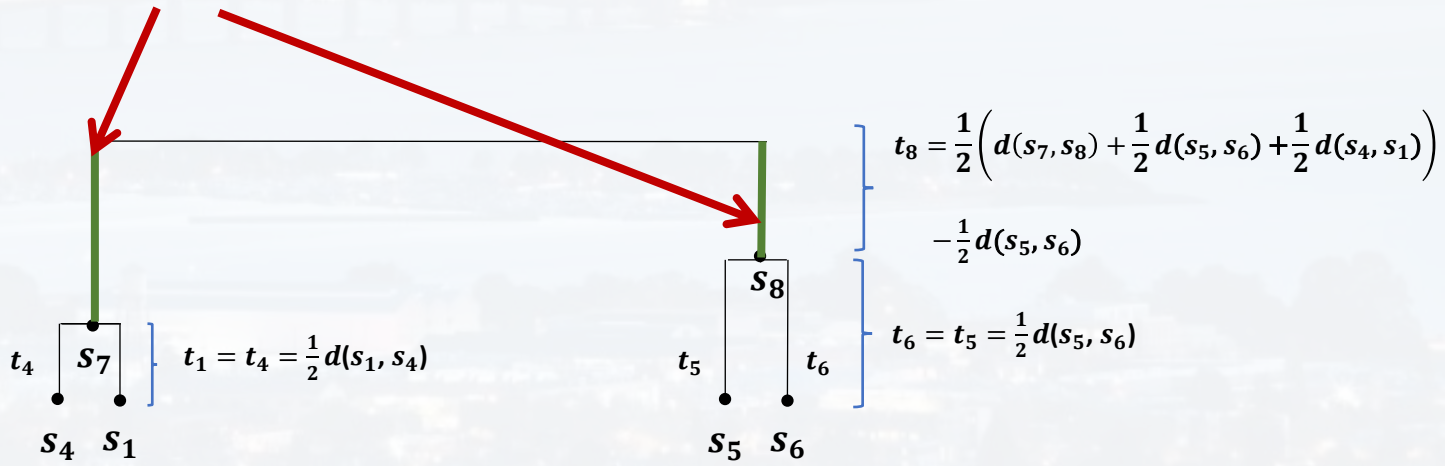
- calculating a distance between each pair of samples



	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

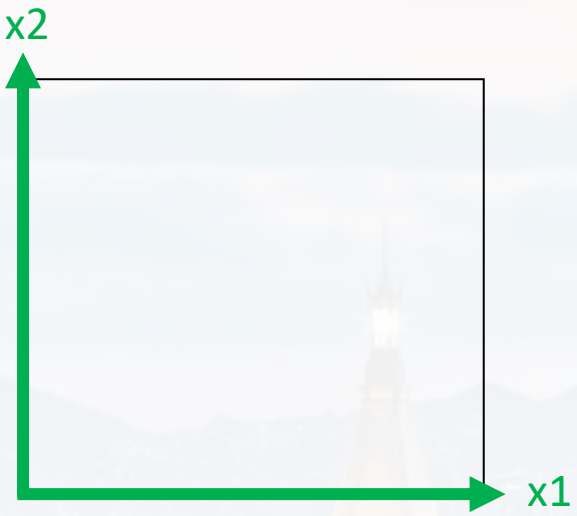
→ and so on....

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



constructing trees:

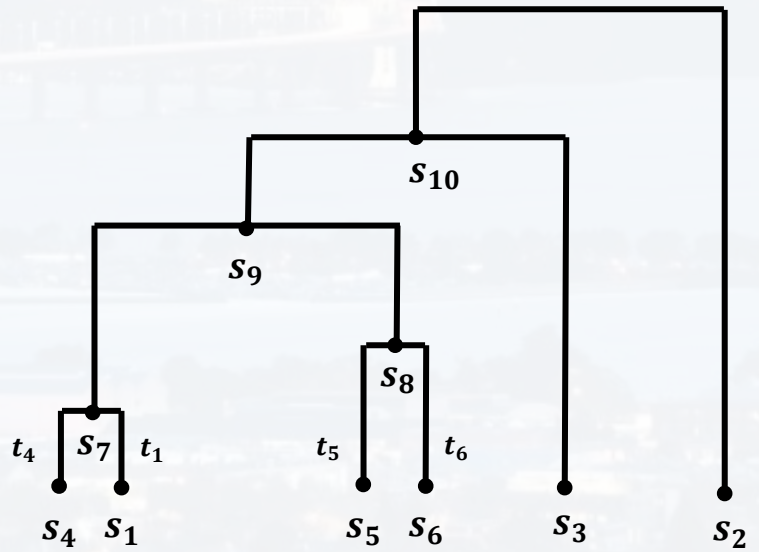
- calculating a distance between each pair of samples



	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

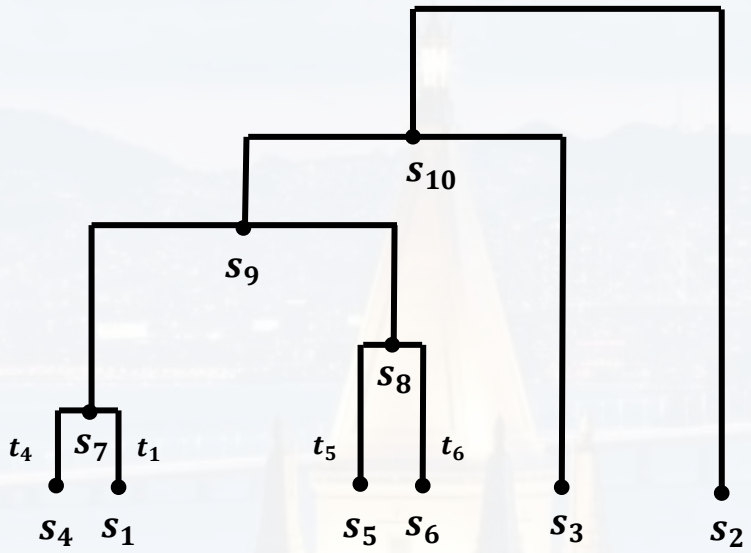
....finally

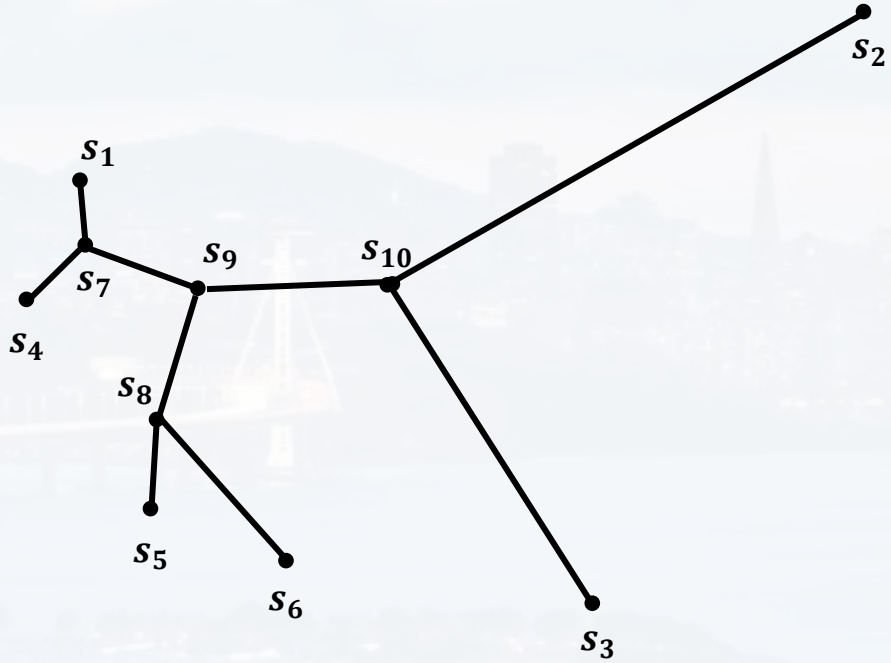
→ **U**nweighted **P**air **G**roup **M**ethod Using **A**rithmetic **A**verages (UPGMA)



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

distances:

"A distance ***d*** is a function that assigns a **positive real number** to a set and/or to elements of a set."

properties:

- **d(x , y) ≥ 0**
- **d(x , y) = 0 $\Leftrightarrow x = y$**
- **d(x , y) = d(y , x)**
- **d(x , z) ≤ d(x , y) + d(y , z)**

frequently used distances:

- Euclidean
- cityblock
- cosine
- correlation
- hamming
- Jukes Cantor

distances:

"A distance ***d*** is a function that assigns a **positive real number** to a set and/or to elements of a set."

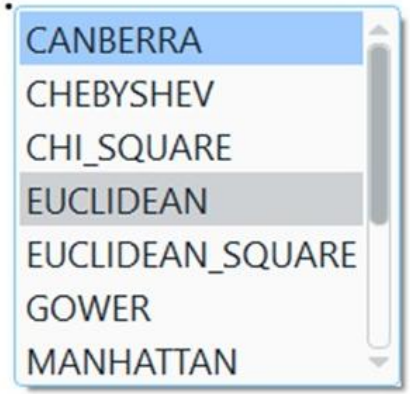
properties:

- **d(x , y) ≥ 0**
- **d(x , y) = 0 ⇔ $x = y$**
- **d(x , y) = d(y , x)**
- **d(x , z) ≤ d(x , y) + d(y , z)**

- Euclidean
- cityblock
- cosine
- correlation
- hamming
- Jukes Cantor

```
from pyclustering.utils.metric import *
```

```
type_metric.
```



distances:

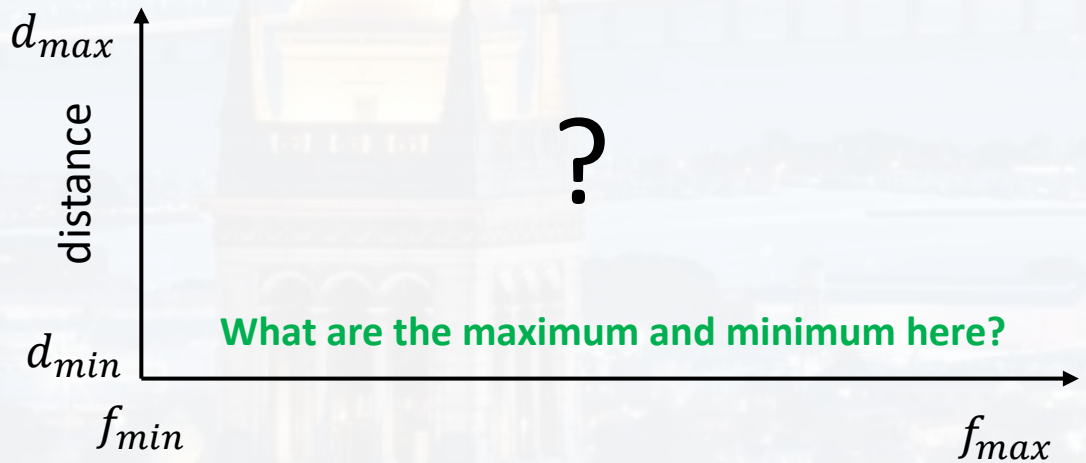
The Jukes-Cantor distance

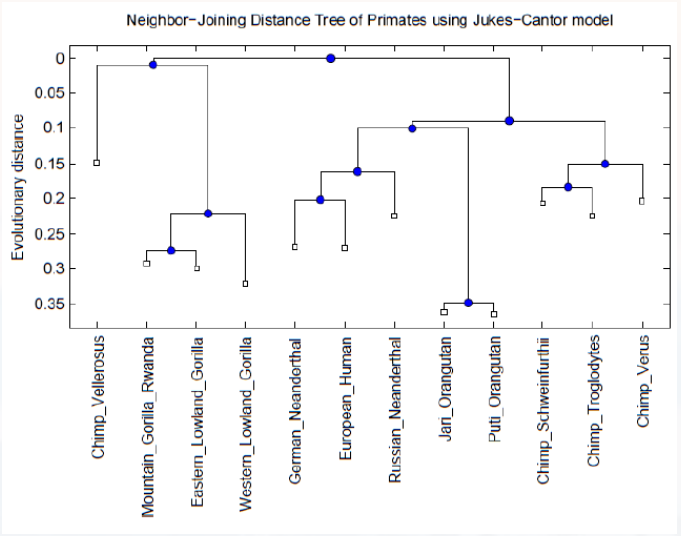
question: What could be a measure of distance here?

Seq1 = {AAGTCTTTGAATCGTATGGGCGCGGCTTTCAAAA}
Seq2 = {AGGTCTATGAATCGTATGGGTTCGGCTTTCAAAA}

fraction ***f*** of different sites

if ***f*** is large → ***d*** is large
if ***f*** is small → ***d*** is small

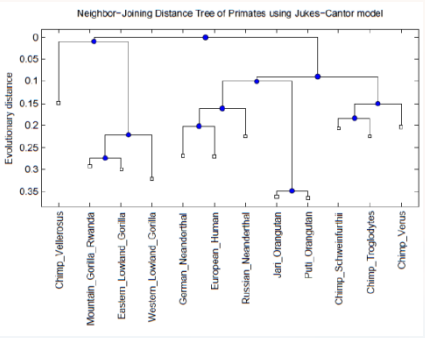




distances:

f : fraction of different sites

Seq1 = {AAGTCTTTGAATCGTATGGGCGCGGCTTTCAAAA}
Seq2 = {AGGTCTATGAATCGTATGGGTTCGGCTTTCAAAA}



The Jukes-Cantor distance

$f_{min} = 0 \rightarrow d_{min} = 0$ (sequences are identical)

for ***N*** letters (***N = 4*** nucleotides, ***N = 20*** amino acids)

$$f_{max} = 1 - \frac{1}{N} \rightarrow d_{\max} = \infty$$

(sequences have no relations, matches are coincidence)

- the probability that a site **has been** mutated, given a mutation rate λ equals:

$$P(any|\lambda) = 1 - e^{-\lambda t}$$

- the probability that this mutation will lead to **a particular letter** is

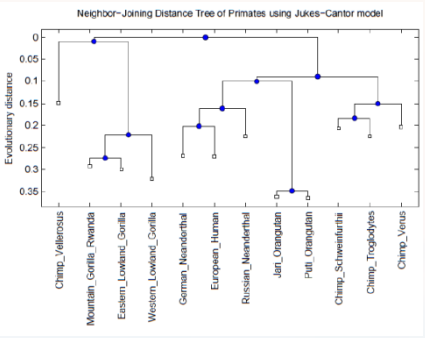
$$P(part|\lambda) = \frac{1}{N}(1 - e^{-\lambda t})$$

- the probability that this mutation will lead to a **letter other than the previous one** is

$$P(part|\lambda) = \frac{N-1}{N}(1 - e^{-\lambda t})$$

distances:

f : fraction of different sites



The Jukes-Cantor distance

Seq1 = {AAGTCTTTGAATCGTATGGGCGCGGCTTTCAAAA}
Seq2 = {AGGTCTATGAATCGTATGGGTTCGGCTTTCAAAA}

$f_{min} = 0 \rightarrow d_{min} = 0$ (sequences are identical)

for ***N*** letters (***N = 4*** nucleotides, ***N = 20*** amino acids)

- the probability that this mutation will lead to a **letter other than the previous one** is

$$P(part|\lambda) = \frac{N-1}{N}(1 - e^{-\lambda t})$$

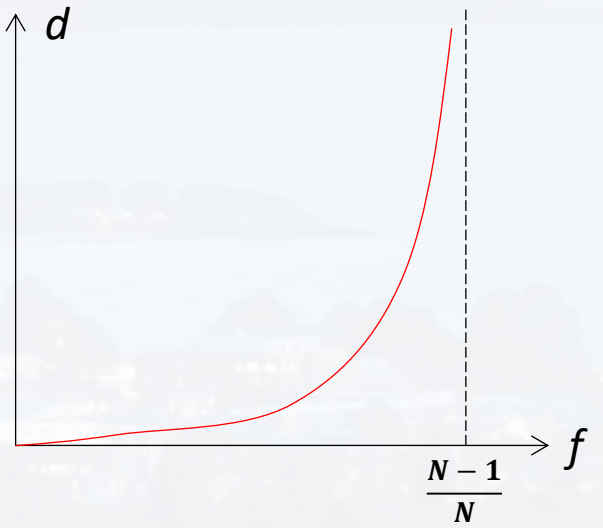
$\lambda t \sim d$ since cumulative mutations is what makes species different!

$$d \sim \lambda t = -ln\left(1 - \frac{N}{N-1} f\right)$$

often:

$$d = -\frac{N-1}{N} ln\left(1 - \frac{N}{N-1} f\right)$$

Jukes – Cantor distance



distances:

Jukes – Cantor distance $d(x,\bar{x}) = -\frac{N-1}{N} ln\left(1 - \frac{N}{N-1} f(x,\bar{x})\right)$

Euclidean distance $d(x,\bar{x})^2 = (x_1 - \bar{x}_1)^2 + (x_2 - \bar{x}_2)^2 \dots (x_N - \bar{x}_N)^2$

cityblock $d(x,\bar{x}) = |x_1 - \bar{x}_1| + |x_2 - \bar{x}_2| \dots |x_N - \bar{x}_N|$

cosine $d(x,\bar{x}) = 1 - \frac{x \cdot \bar{x}}{\sqrt{(x \cdot x)(\bar{x} \cdot \bar{x})}}$

correlation $d(x,\bar{x}) = 1 - \frac{cov(x,\bar{x})}{\sqrt{var(x)var(\bar{x})}}$

...and many others

note: for sequence alignment and phylogenetic trees, chemical properties of the AA or NT have to be taken into account → **score matrices**

Python libraries:

see **`Walk_Through_Tree.ipynb`**

libraries from the *Bio* package

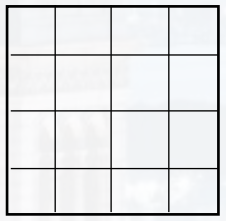
```
from Bio.Phylo.TreeConstruction import DistanceCalculator, DistanceTreeConstructor
from Bio import Phylo
from scipy import spatial
```

for plotting a phylogenetic tree

for rearranging a distance matrix

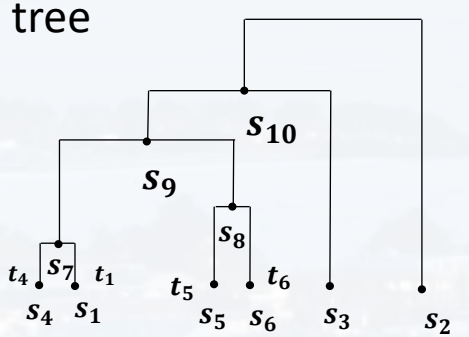
general idea:

distance matrix



linkage algorithm (e.g. UPGMA)

tree



see **`Walk_Through_Tree.ipynb`**

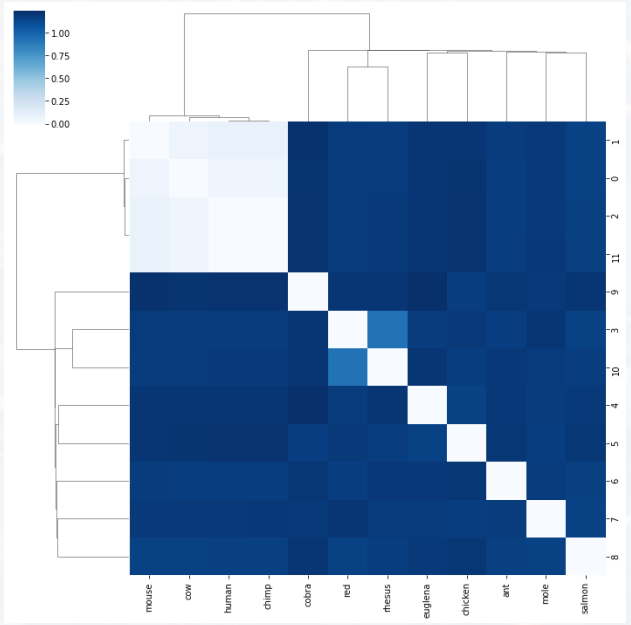
Python libraries:

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

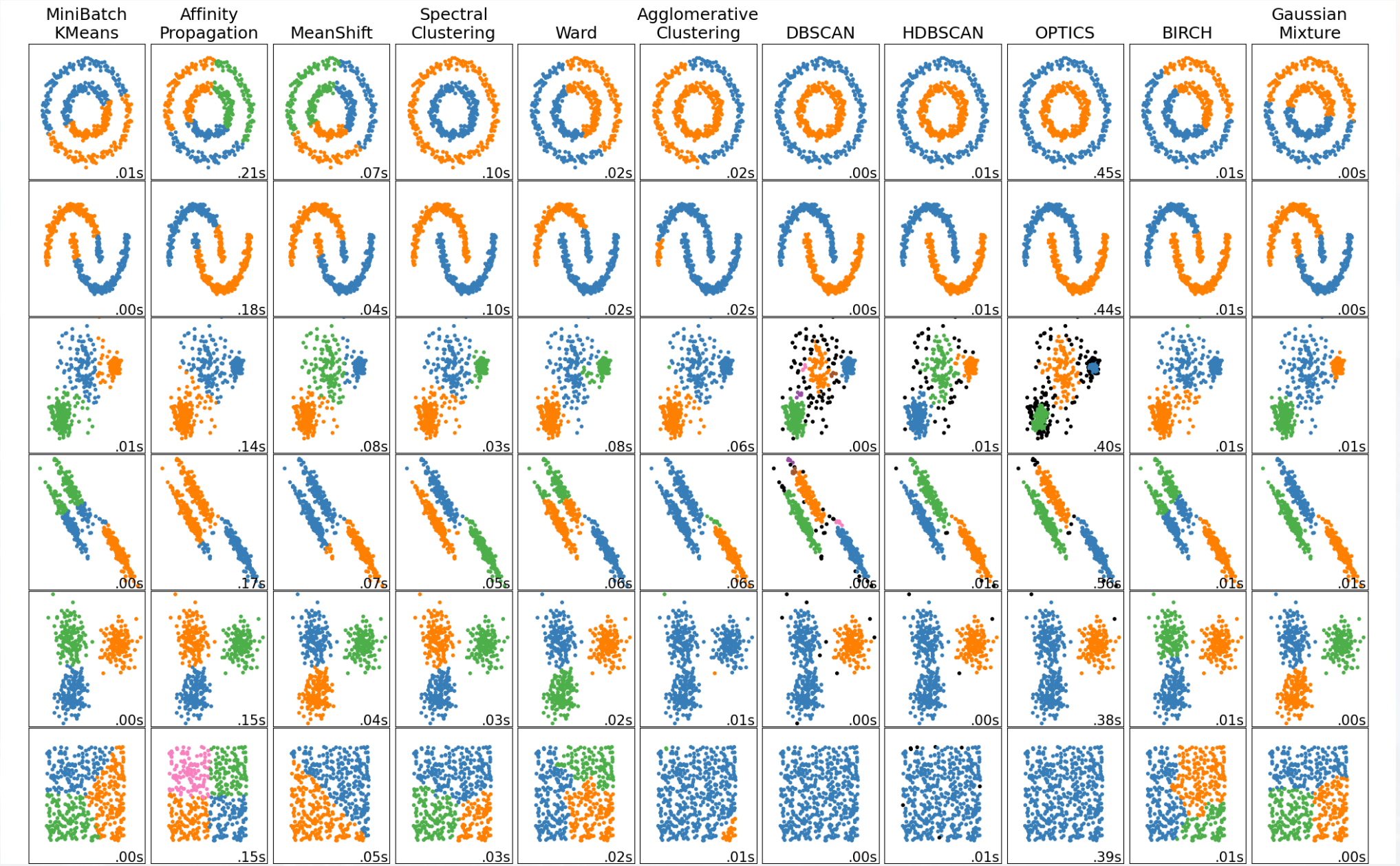
```
sns.clustermap(D_df, cmap = 'Blues', \
                     row_cluster = True, col_cluster = True, \
                     metric = 'euclidean', method = 'average', \
                     yticklabels = True)

plt.show()
```

similar to UPGMA



there is a lot more...



Thank you very much for your attention!