Berkeley MSSE
MOLECULAR SCIENCE & SOFTWARE ENGINEERING

Lecture 13:

Machine Learning Fundamentals

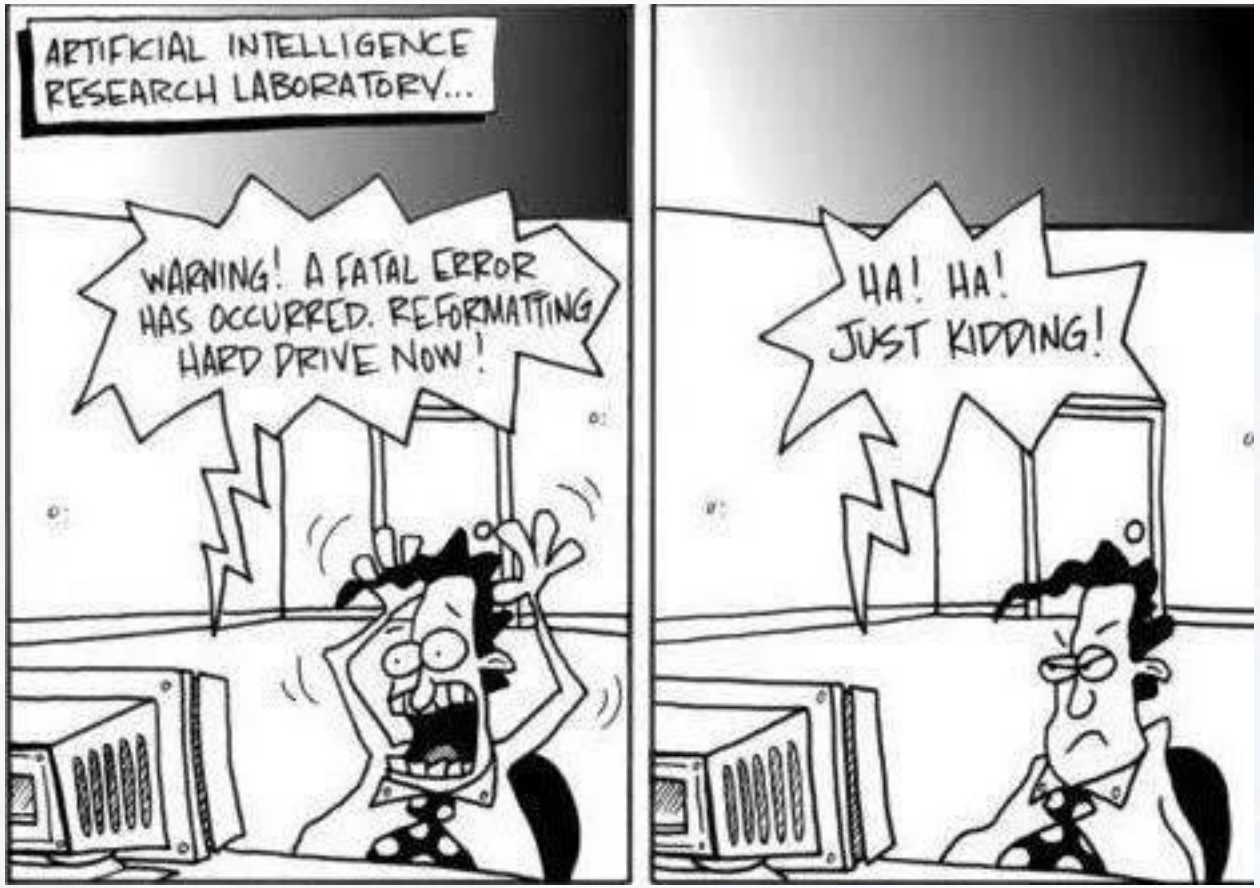
Markus Hohle

University California, Berkeley

Numerical Methods for Computational Science

Course Map

Week 1:	Introduction to Scientific Computing and Python Libraries
Week 2:	Linear Algebra Fundamentals
Week 3:	Vector Calculus
Week 4:	Numerical Differentiation and Integration
Week 5:	Solving Nonlinear Equations
Week 6:	Probability Theory Basics
Week 7:	Random Variables and Distributions
Week 8:	Statistics for Data Science
Week 9:	Eigenvalues and Eigenvectors
Week 10:	Simulation and Monte Carlo Method
Week 11:	Data Fitting and Regression
Week 12:	Optimization Techniques
Week 13:	**Machine Learning Fundamentals**



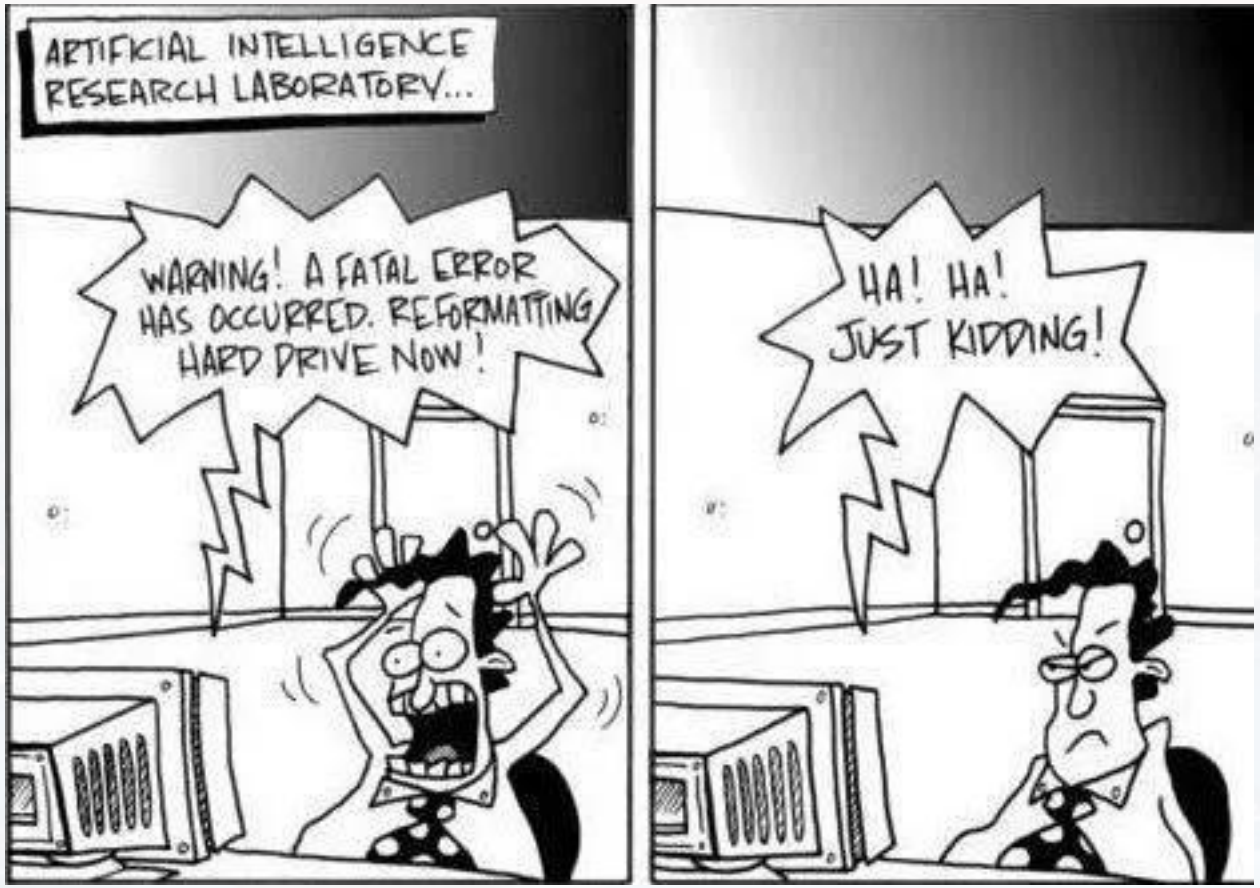
Concept

Supervised Learning

- **S**upport **V**ector **M**achine
- K-nearest

Unsupervised Learning

- K – means
- **G**aussian **M**ixture **M**odels



Concept

Supervised Learning

- **S**upport **V**ector **M**achine
- K-nearest

Unsupervised Learning

- K – means
- **G**aussian **M**ixture **M**odels

What is data mining?

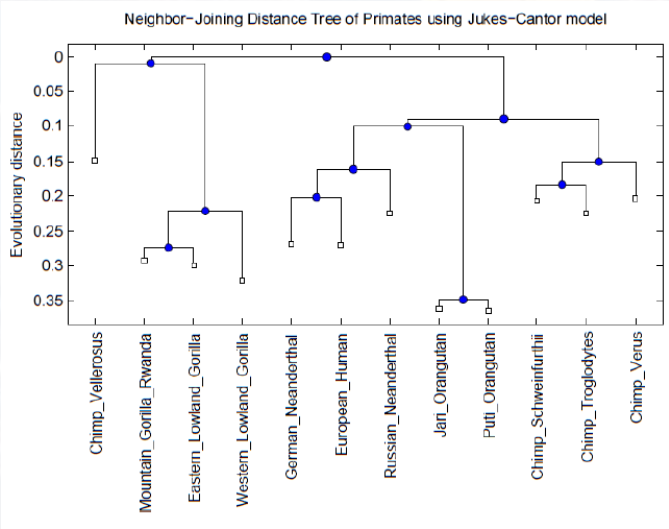
Wiki says:

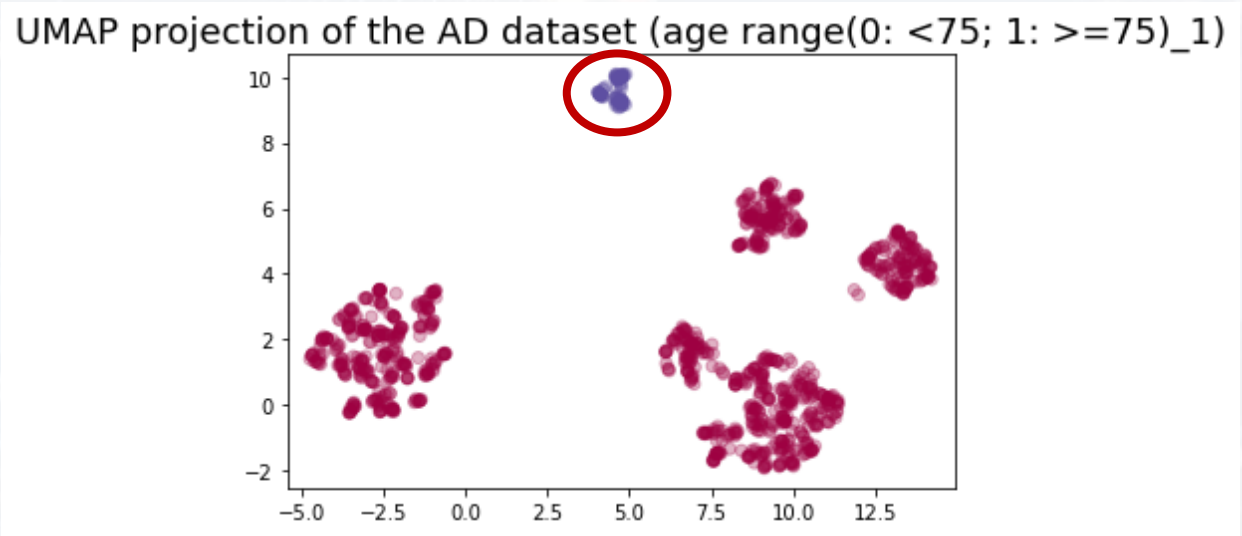
"***Data mining*** *is the computing process of discovering patterns in large data sets involving methods at the intersection of*

- *artificial intelligence,*
- *machine learning,*
- *statistics,*
- *and database systems."*

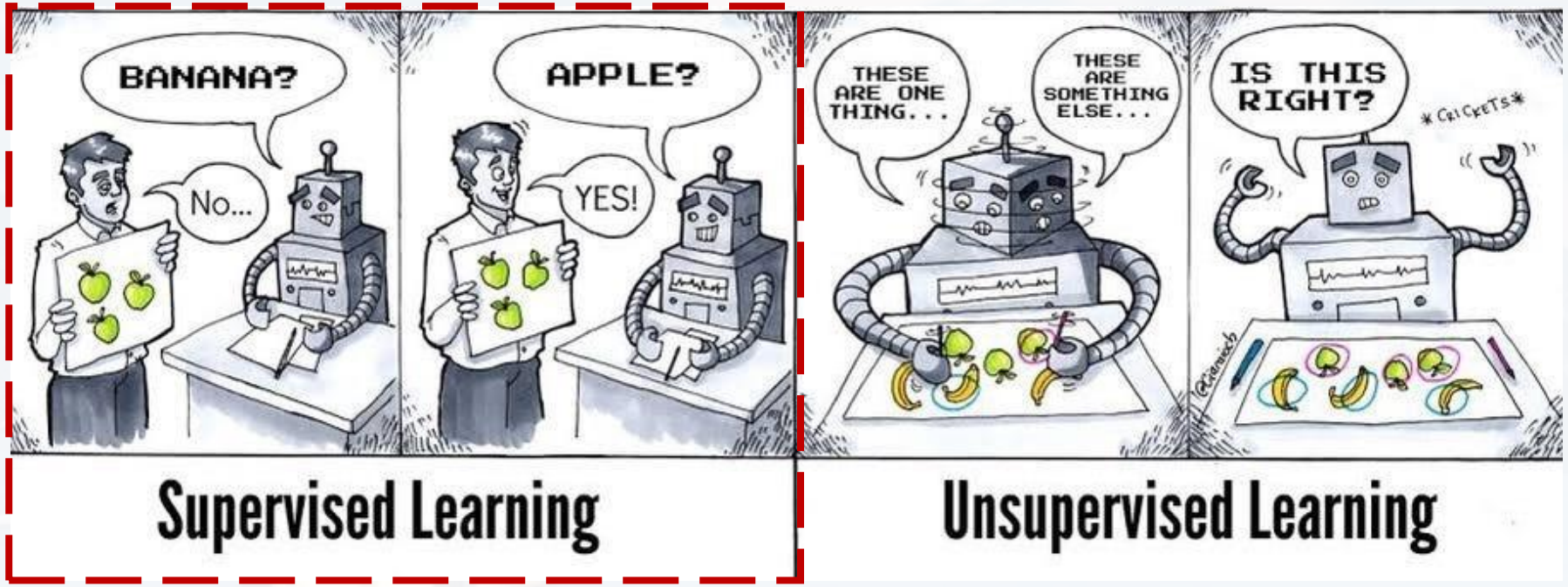
"computers have the ability to learn without being explicitly programmed"

cluster and/or categorize data and understand relations/interactions.





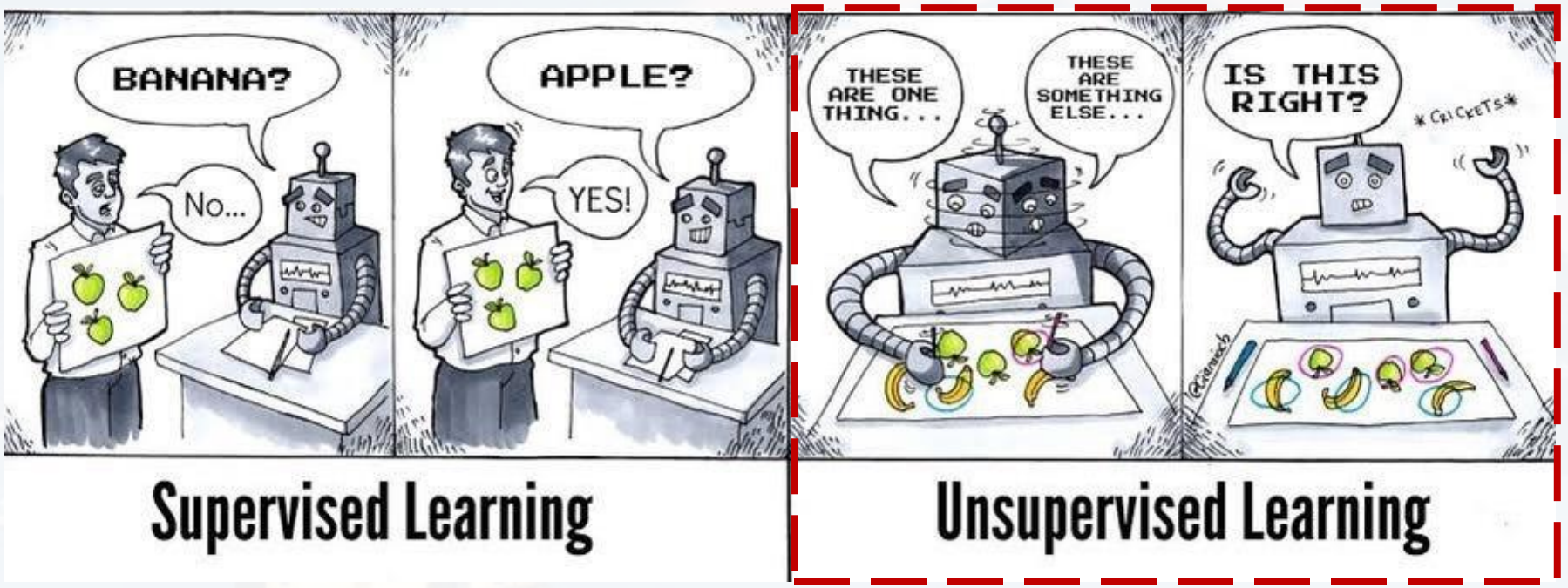
curve fitting, linear/logistic models:
training data set and a **test** data set…



- **S**upport **V**ector **M**achine
- K-nearest

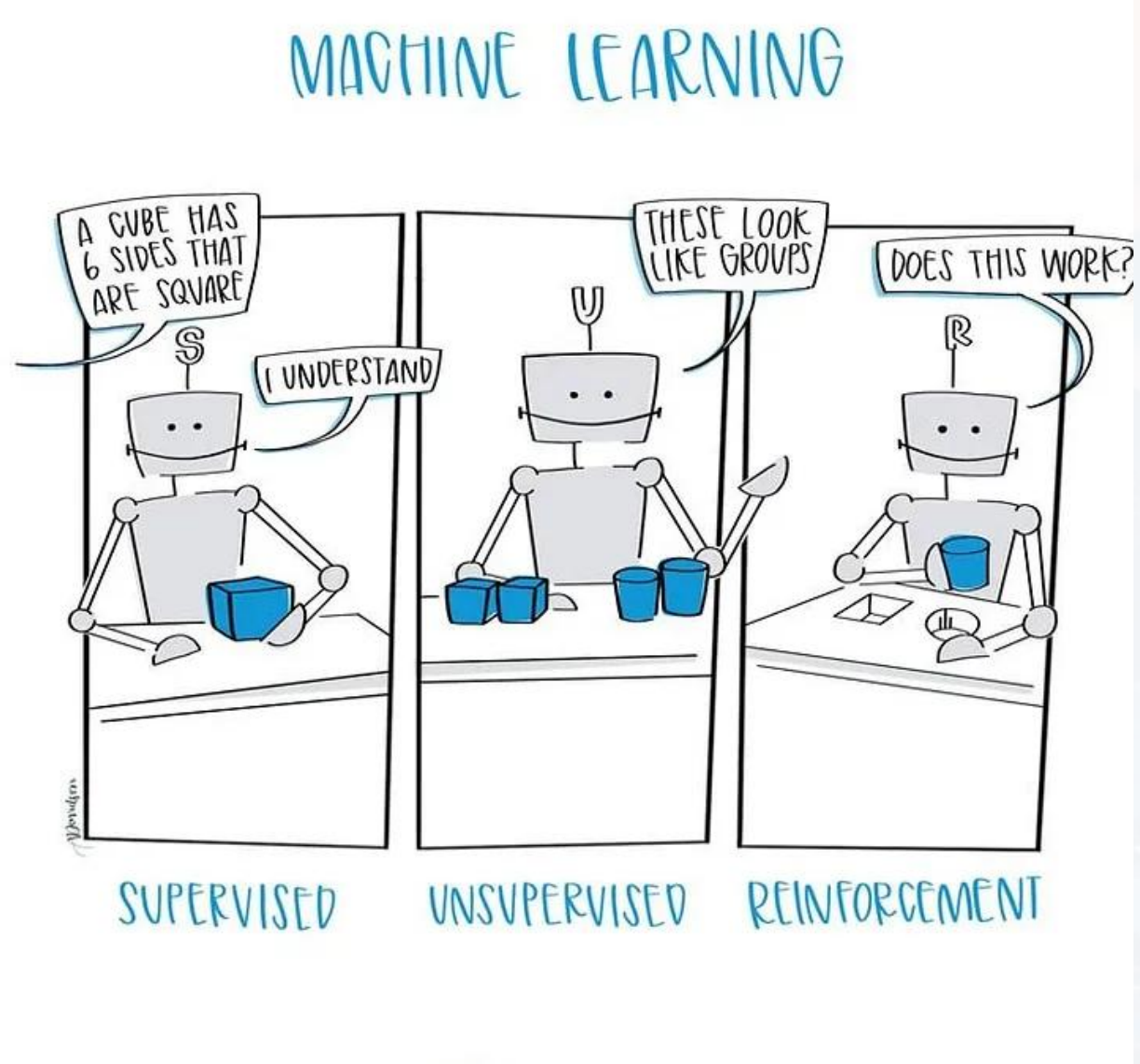
curve fitting, linear/logistic models:
training data set and a **test** data set…

no training data set required – we can start right away!

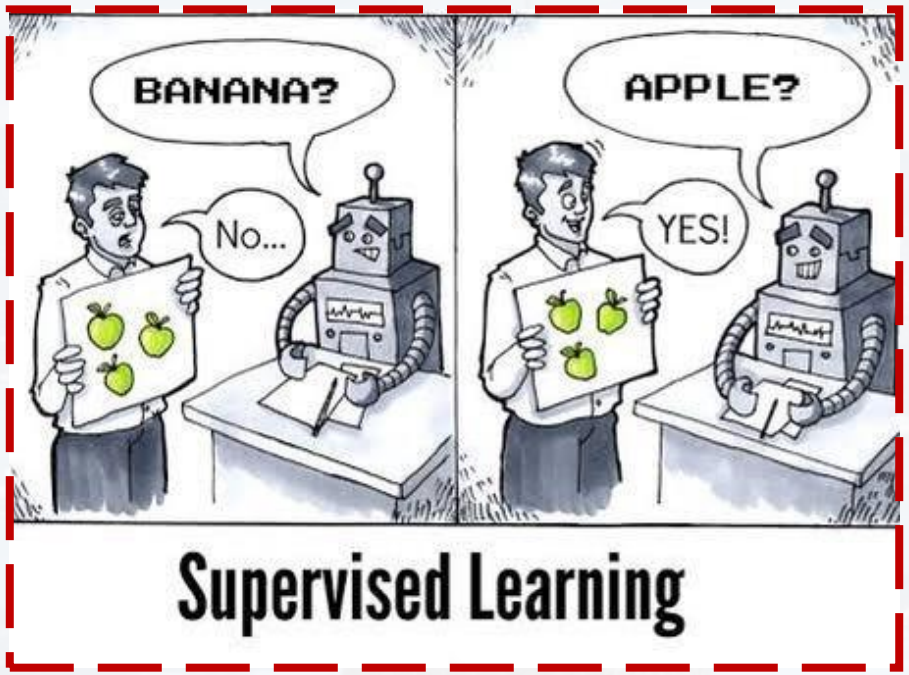


- **S**upport **V**ector **M**achine
- K-nearest

- K - means
- GMM


MACHINE LEARNING
A CUBE HAS 6 SIDES THAT ARE SQUARE
S
I UNDERSTAND
U
THESE LOOK LIKE GROUPS
DOES THIS WORK?
R
SUPERVISED
UNSUPERVISED
REINFORCEMENT

curve fitting, linear/logistic models:
training data set and a **test** data set…



1) creating the model:

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

2) training the model

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

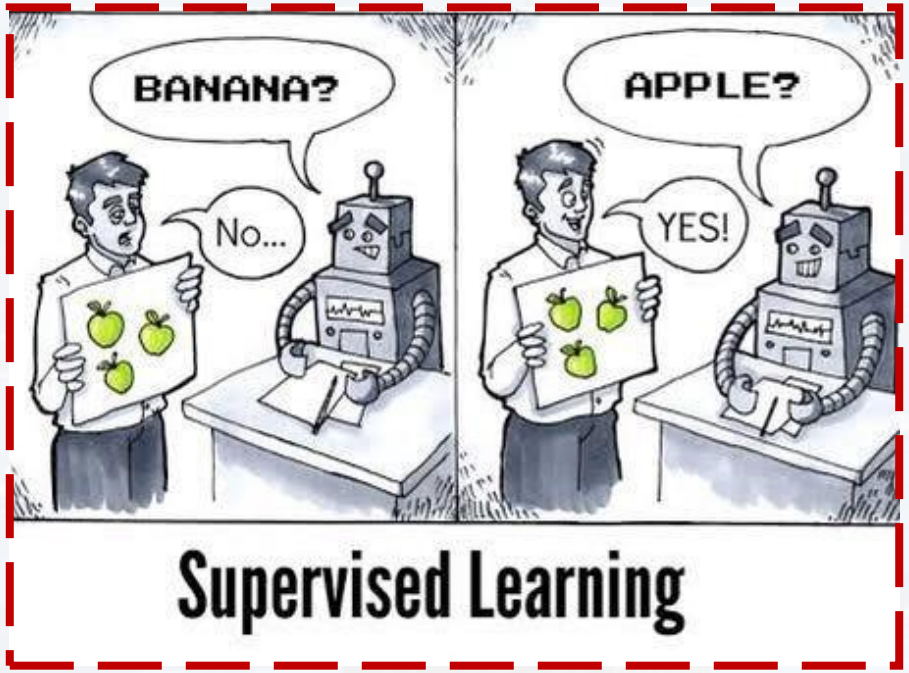
3) evaluation

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

4) prediction (actual application)

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

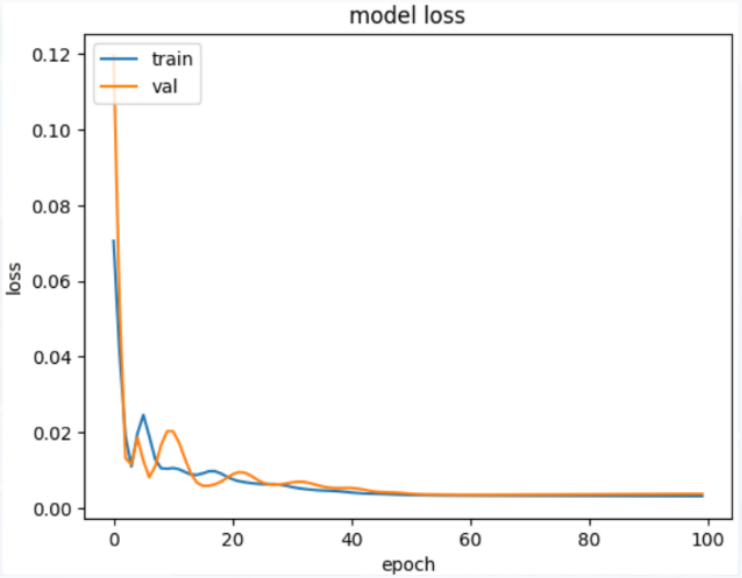
curve fitting, linear/logistic models:
training data set and a **test** data set…



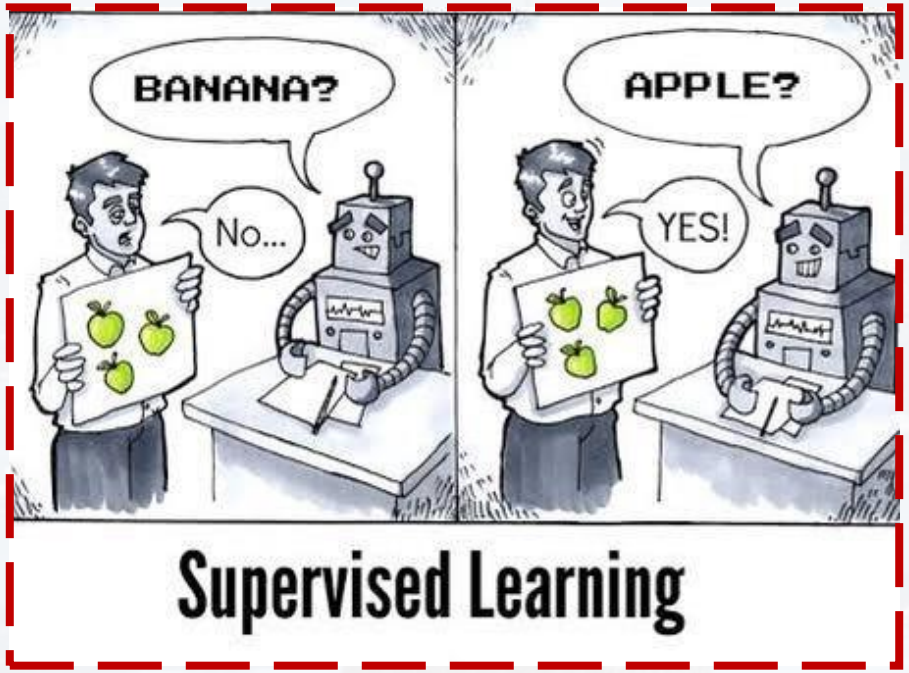
<u>How to split the data?</u>

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

evaluation while training!



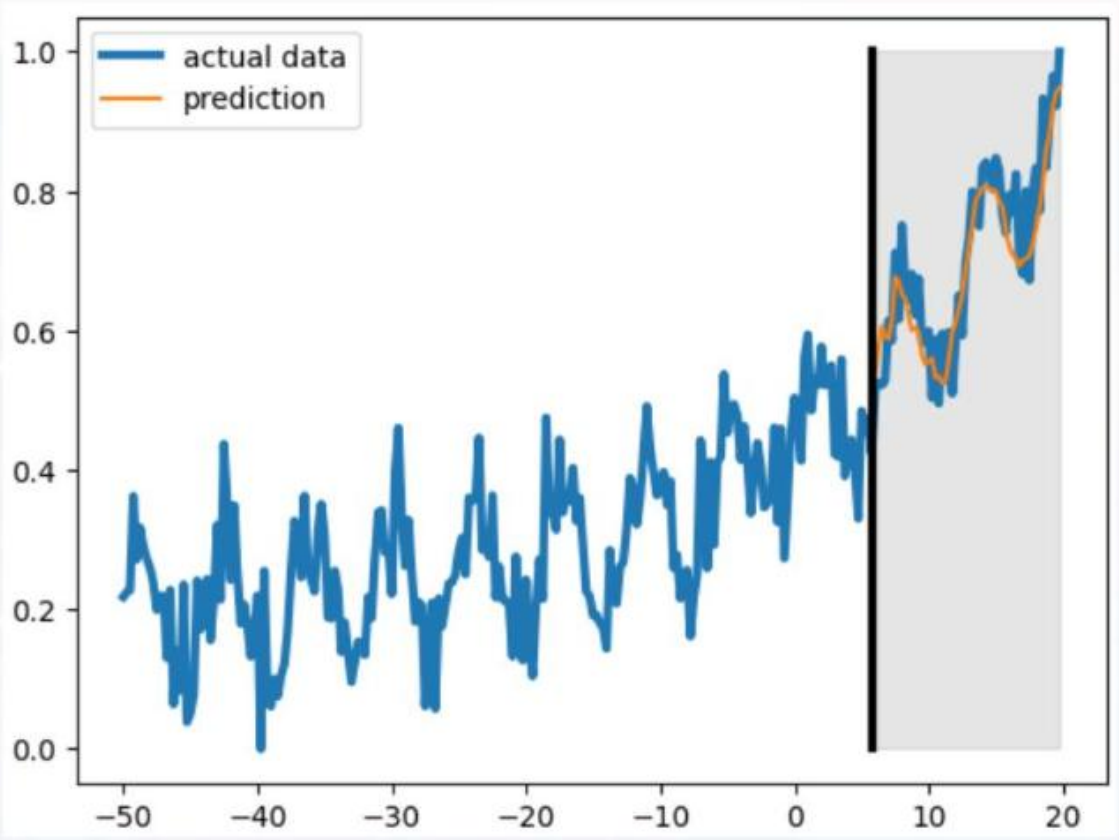
curve fitting, linear/logistic models:
training data set and a **test** data set…



<u>How to split the data?</u>

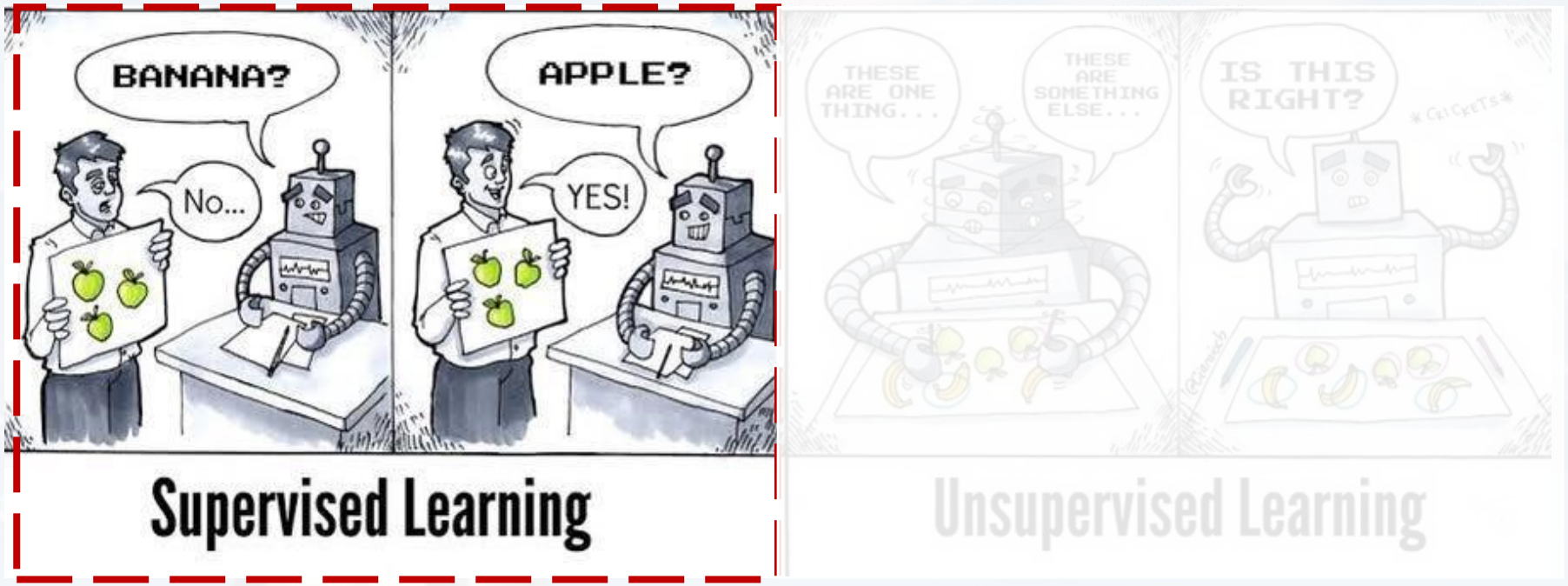
- 80% training (60% actual training 20% evaluation)
- 20% test

prediction after training/evaluation completed



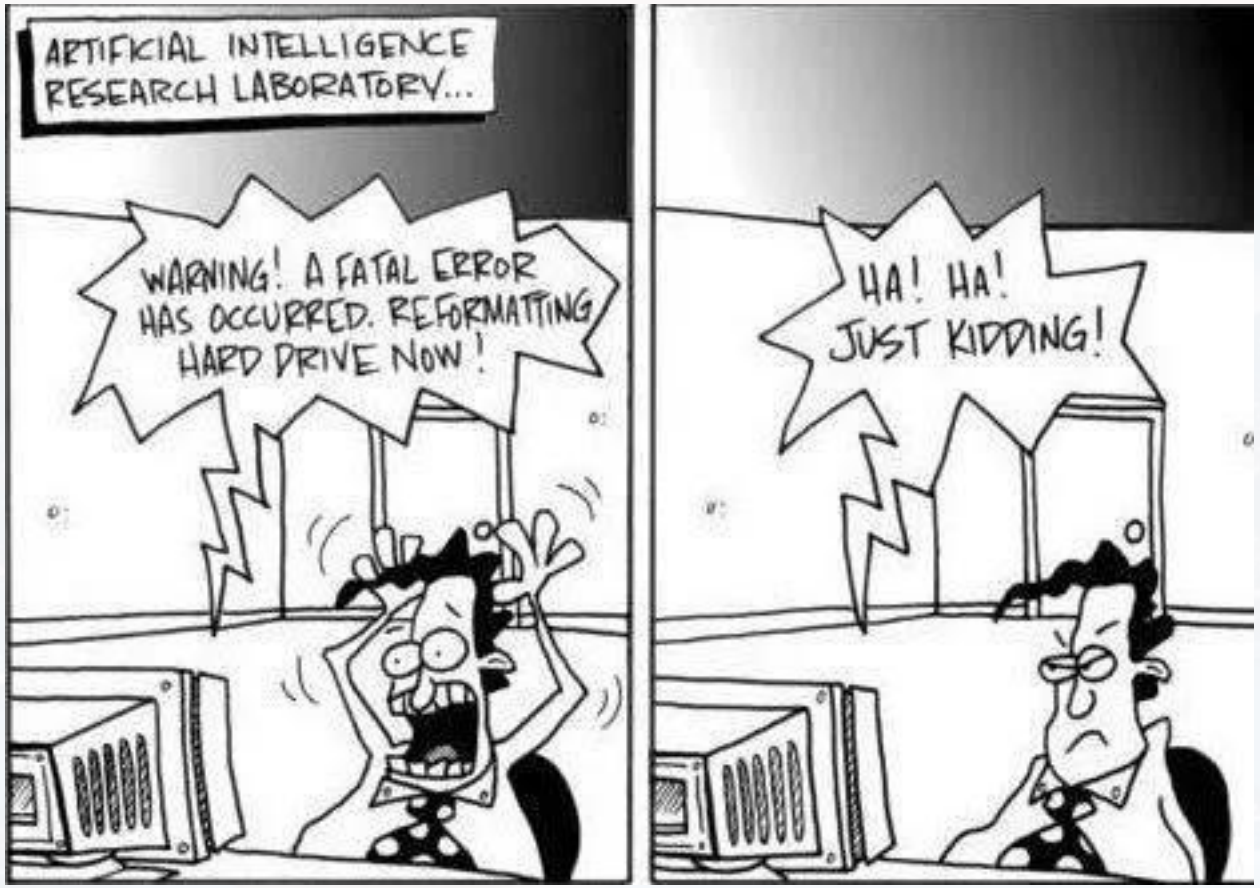
curve fitting, linear/logistic models:
training data set and a **test** data set…

no training data set required – we can start right away!



- **Support Vector Machine**
- **K-nearest**

- K - means
- GMM



Concept

Supervised Learning

- **S**upport **V**ector **M**achine
- K-nearest

Unsupervised Learning

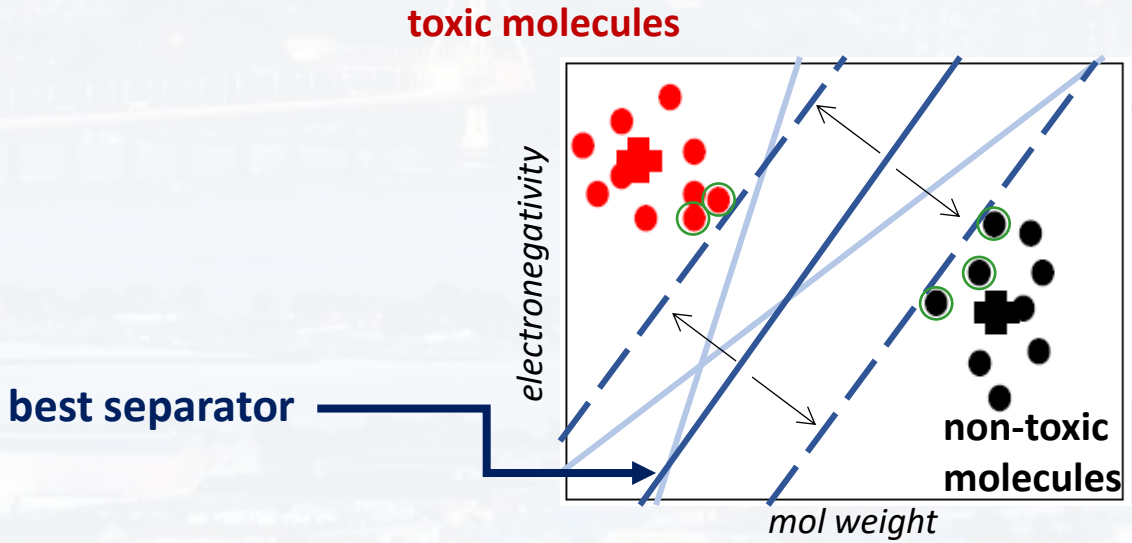
- K – means
- **G**aussian **M**ixture **M**odels

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

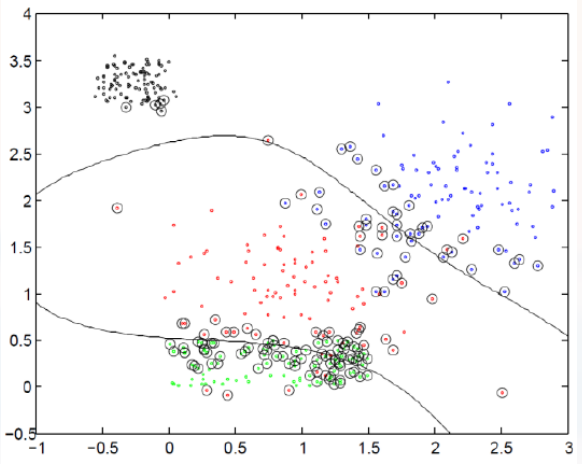
idea:

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)

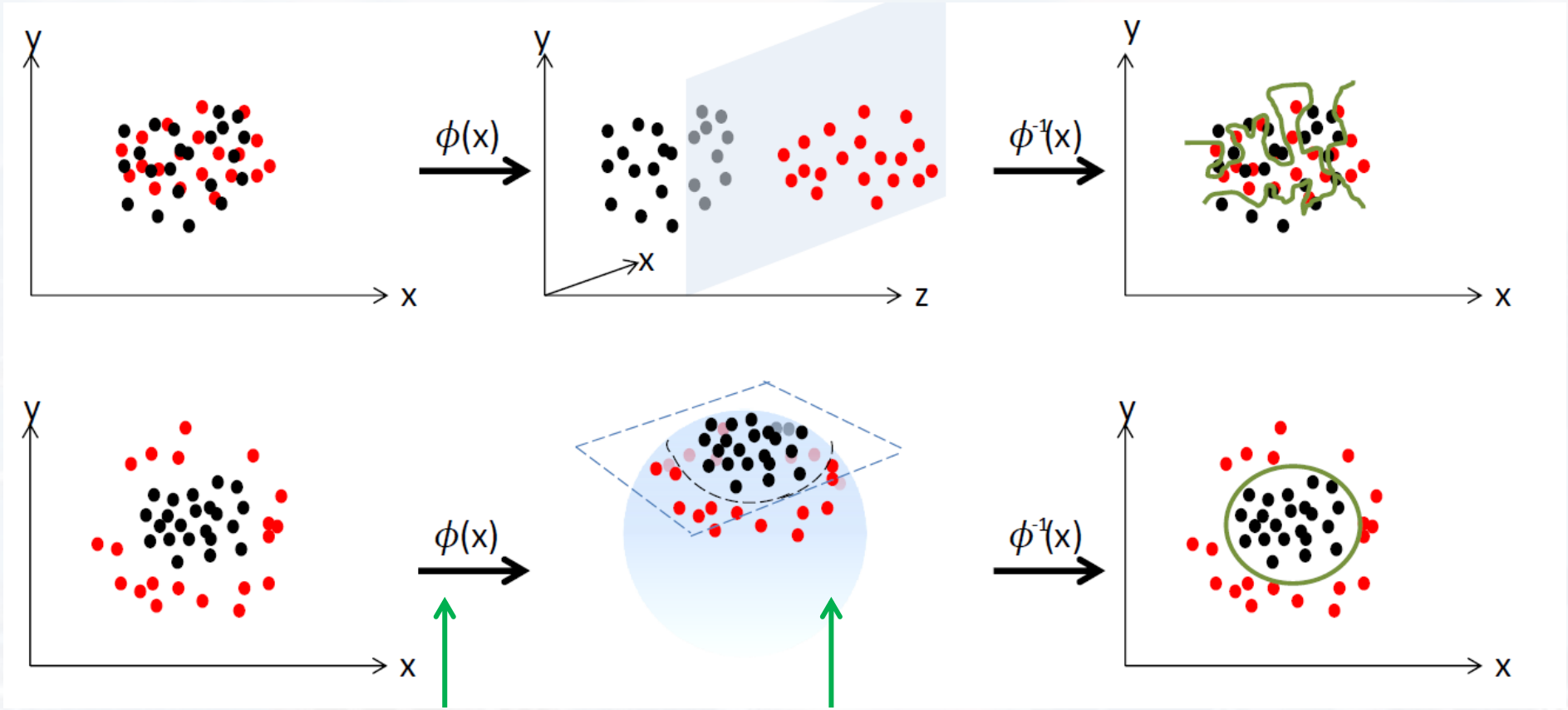


1) What if linear separation is not possible?
2) Is there a multiclass SVM?



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

Theorem: "There exists always a (linear) hyperplane in higher dimensions."



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

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

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

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

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

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

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

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

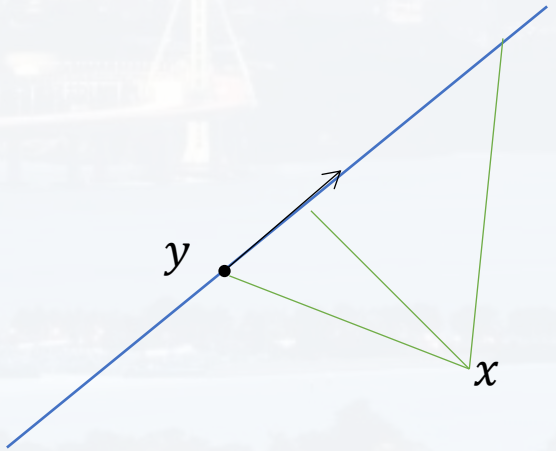
1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

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

example: linear kernel $\phi: x \mapsto x$ $\phi(x) = x$

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

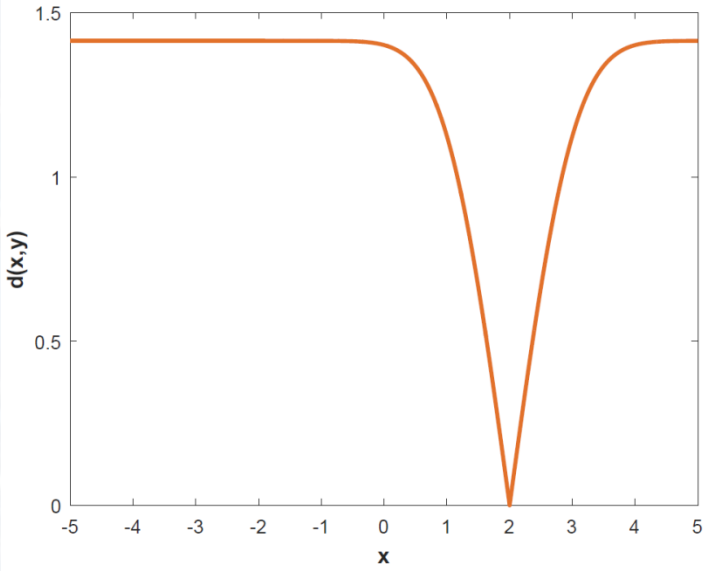
$$d(x,y) = |x-y|$$

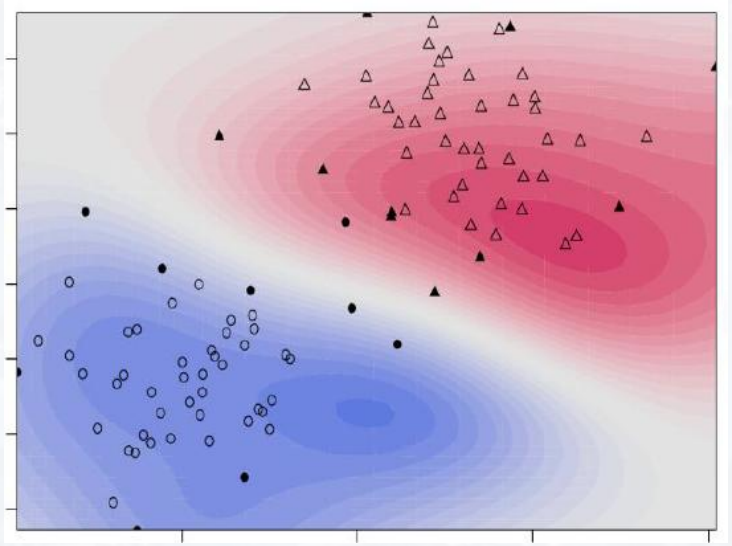


1) **What if linear separation is not possible?**
2) Is there a multiclass SVM?

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

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





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

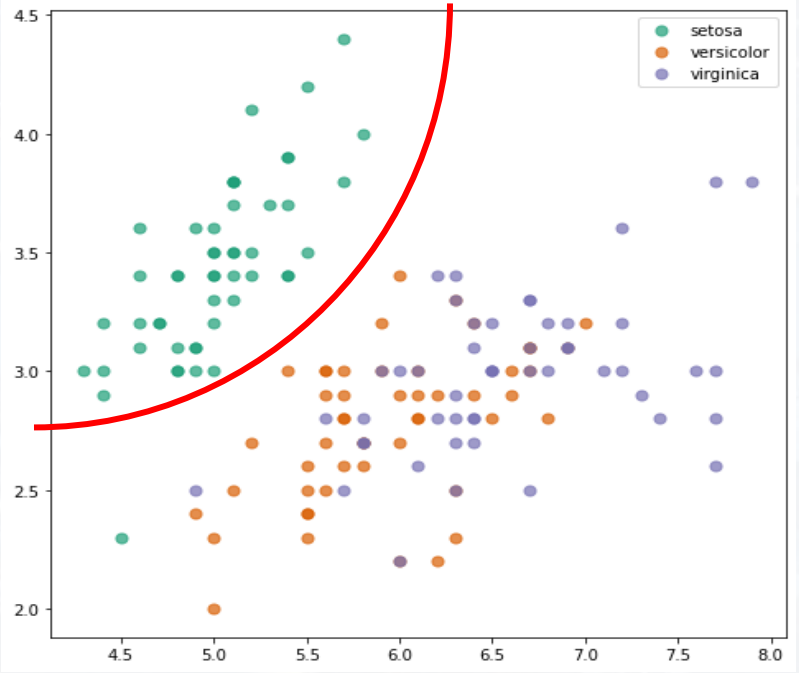
in `sklearn` we can adjust $\boldsymbol{N}$

- 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

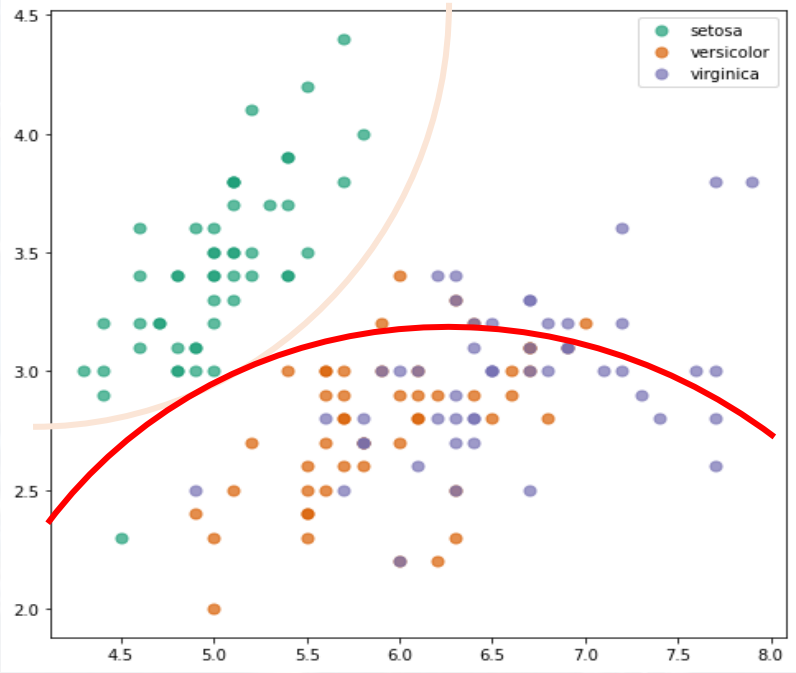
green vs the rest
→ storing probabilities



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

→ one vs rest / one vs one

green vs the rest
→ storing probabilities

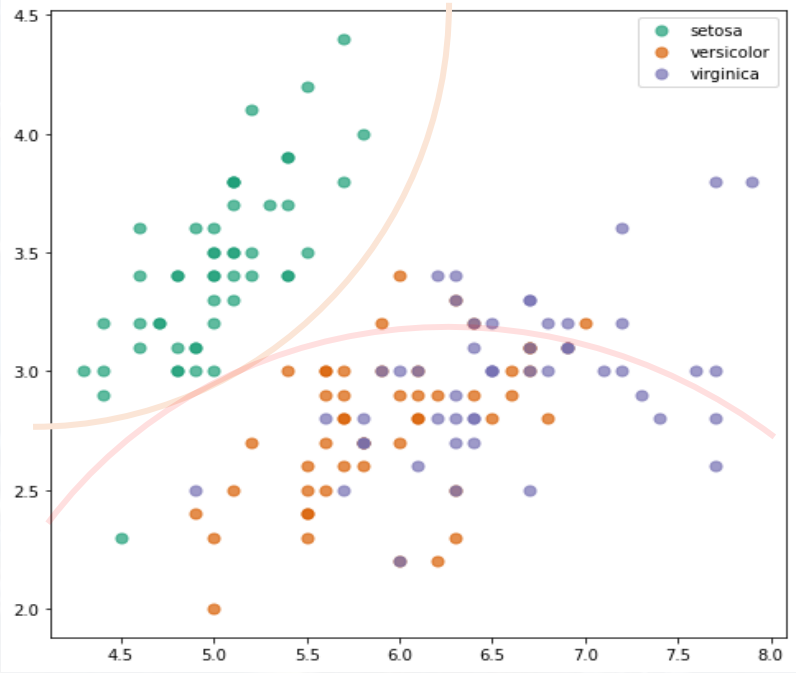


orange vs the rest
→ storing probabilities

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



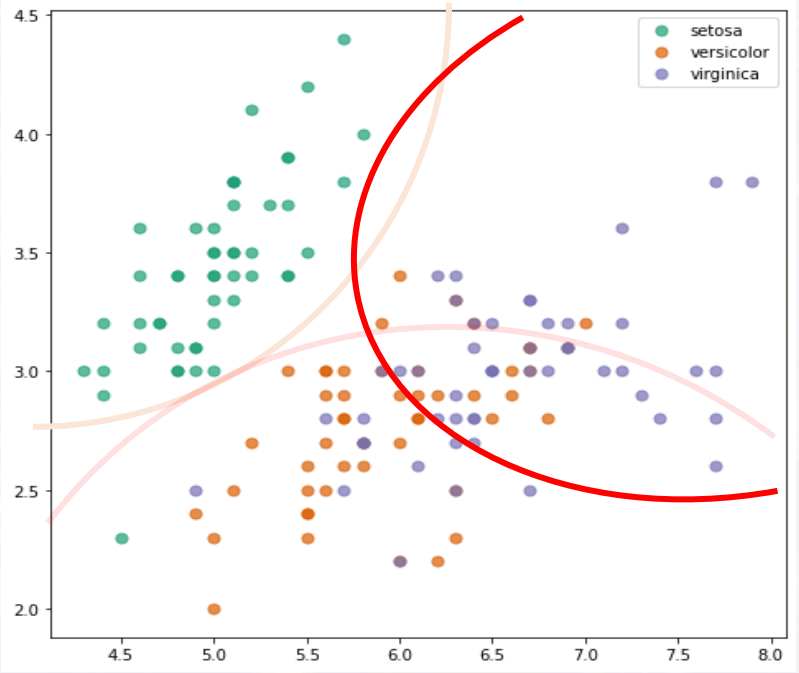
blue vs the rest
→ storing probabilities

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



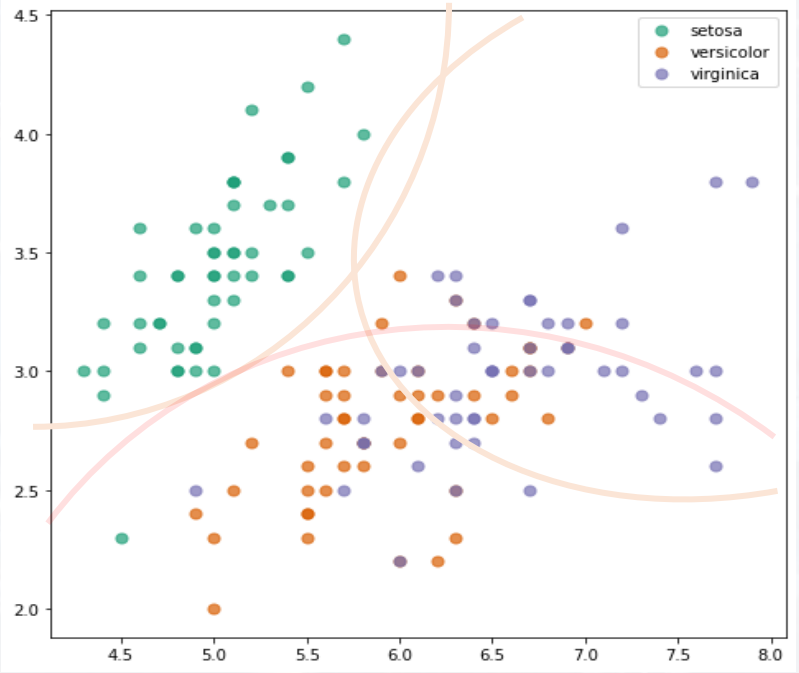
blue vs the rest
→ storing probabilities

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



blue vs the rest
→ storing probabilities

orange vs the rest
→ storing probabilities

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

our standard libraries

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

from sklearn import datasets

iris = datasets.load_iris()

iris.DESCR
```

```
from sklearn import datasets
```

see **Walk_Through_SVM.ipynb**

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']
```

```
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

```
from sklearn import datasets
```

see **`Walk_Through_SVM.ipynb`**

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

```
from sklearn import svm
```

running analysis with different kernel

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

```
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

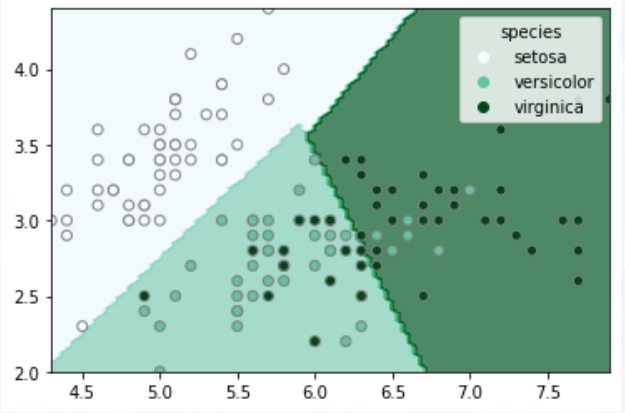
parameter for error tolerance when calculating the classifier

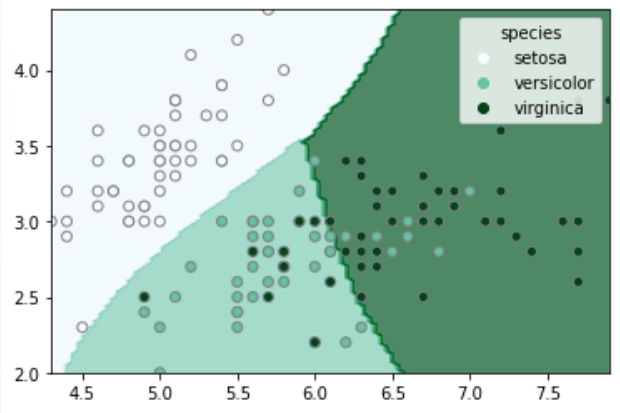
$\gamma := \frac{1}{2\sigma^2}$

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

run: SVM.py to visualize the prediction 3) evaluating the model

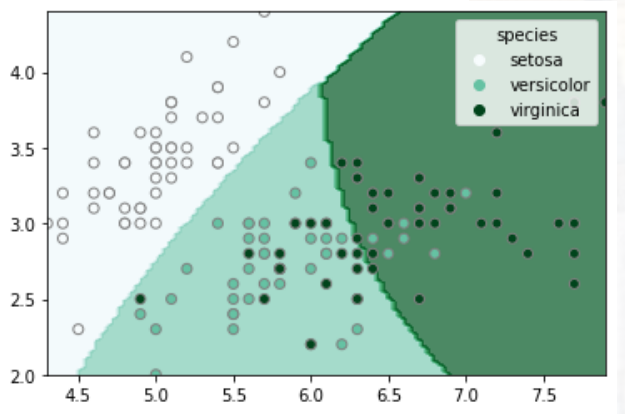
$K(x, y) = \|x - y\|$

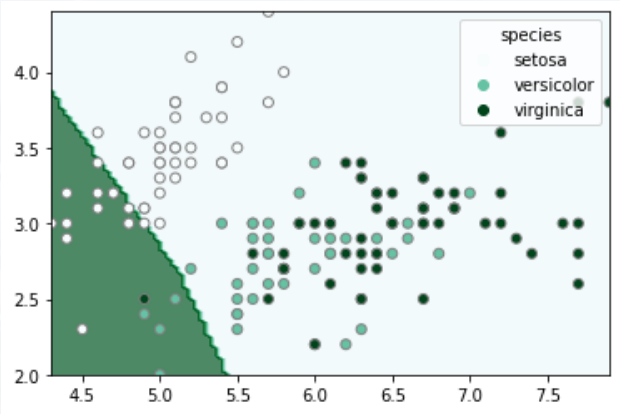




$K(x, y) = exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$

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

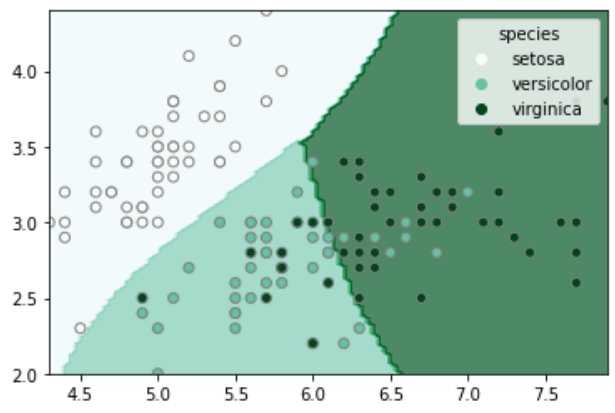




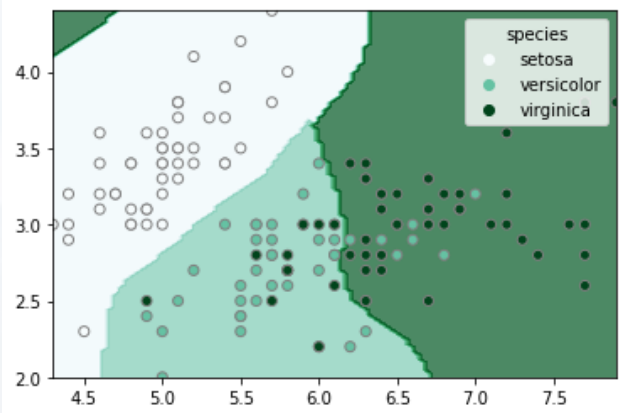
$K(x, y) = \frac{e^{\|x-y\|}}{1 + e^{\|x-y\|}}$

run: SVM.py to visualize the prediction 3) evaluating the model

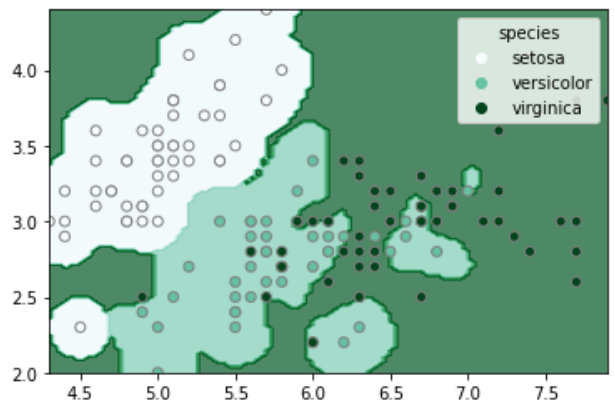
$$K(x, y) = exp\left(-\frac{1}{2\sigma^2}\|x - y\|^2\right)$$



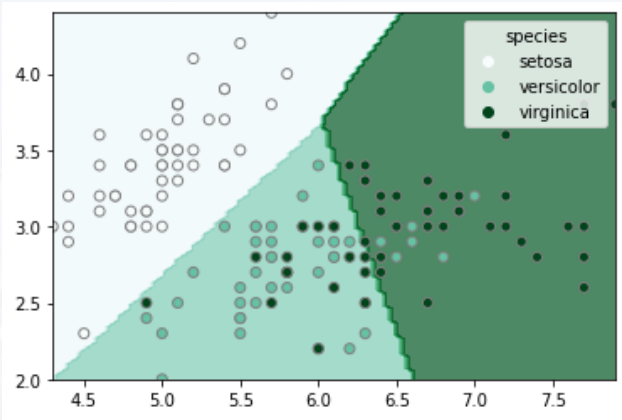
$\gamma := \frac{1}{2\sigma^2} = 1$



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



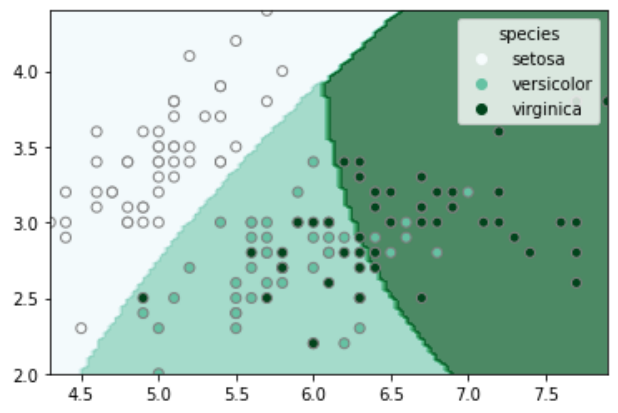
$\gamma := \frac{1}{2\sigma^2} = 50$



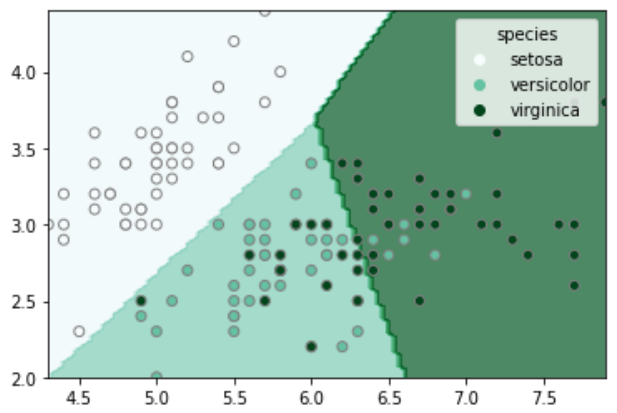
$\gamma := \frac{1}{2\sigma^2} = 0.1$

run: SVM.py to visualize the prediction 3) evaluating the model

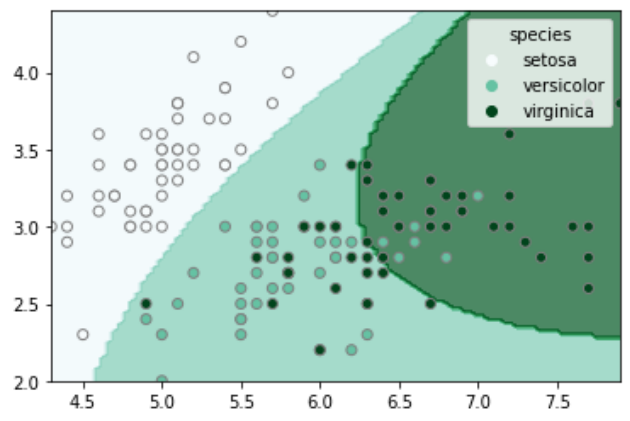
$$K(x, y) = \sum_{n=1}^{N} \|x - y\|^n$$



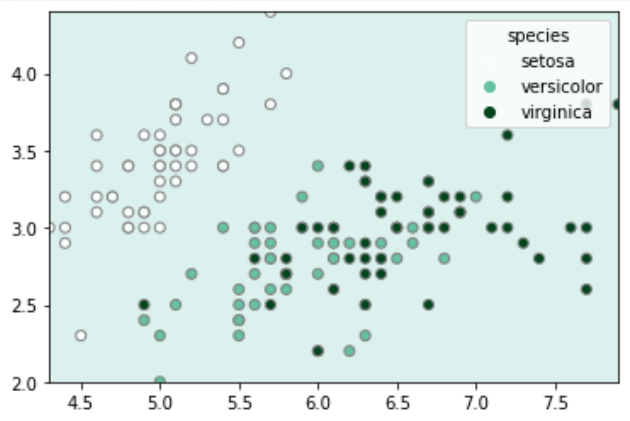
N = 3



N = 1



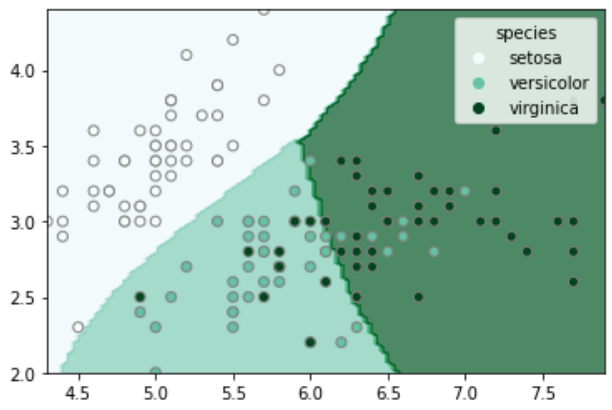
N = 5



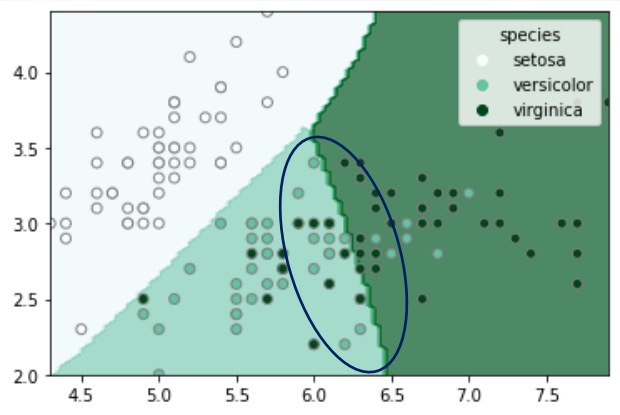
N = 0

run: SVM.py to visualize the prediction 3) evaluating the model

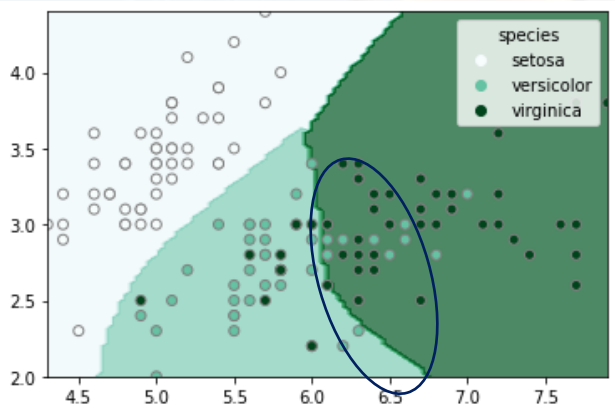
$$K(x, y) = exp\left(-\frac{1}{2\sigma^2}\|x - y\|^2\right)$$



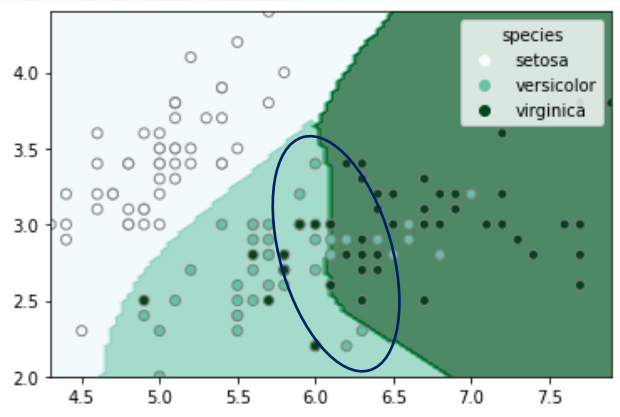
C = 1



C = 0.1



C = 10

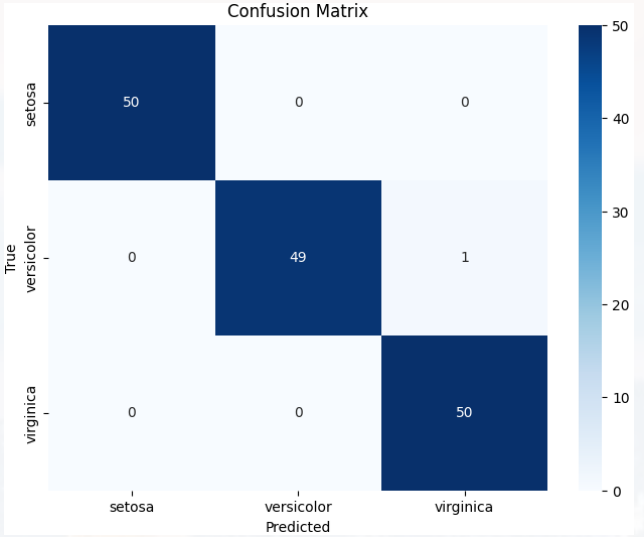


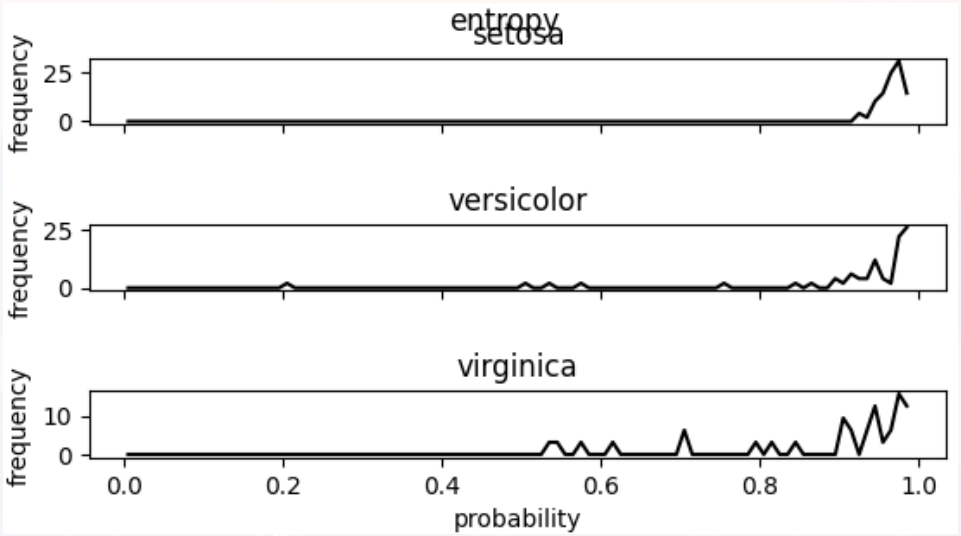
C = 20

full **4D** data set

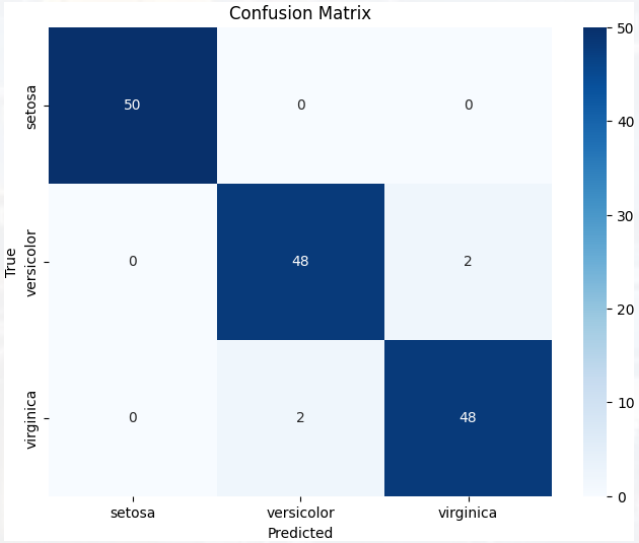
3) evaluating the model

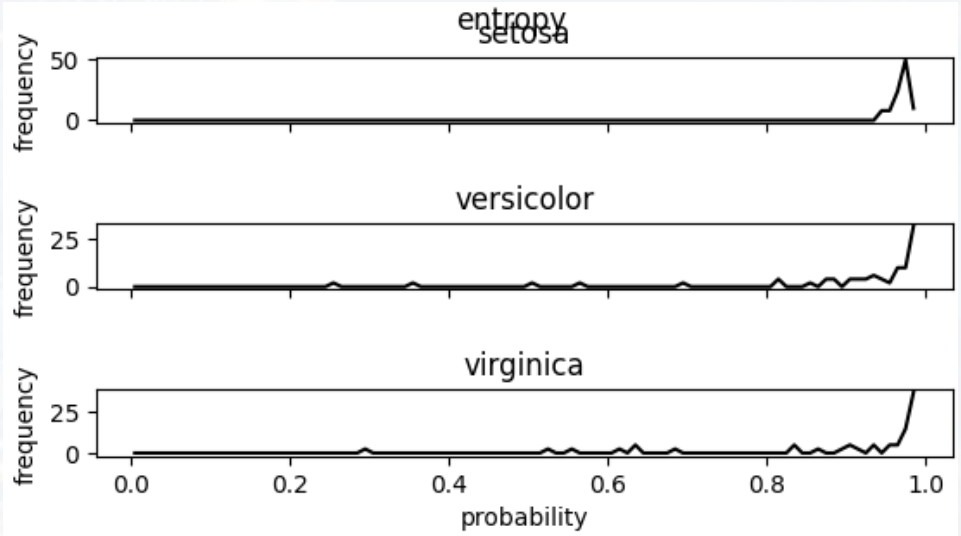
linear
accuracy: 99.3%





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

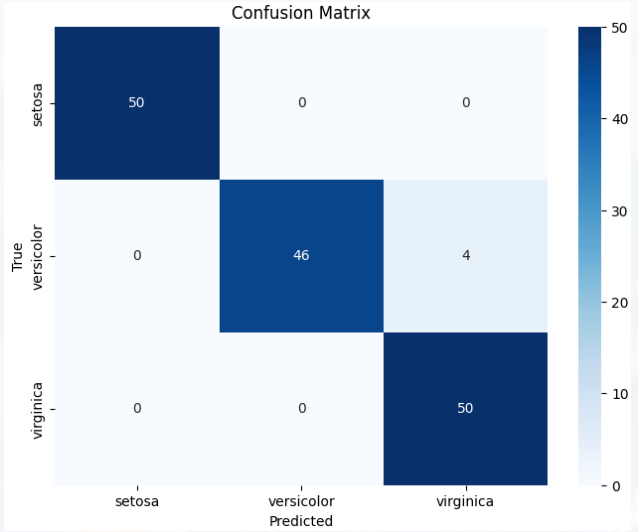


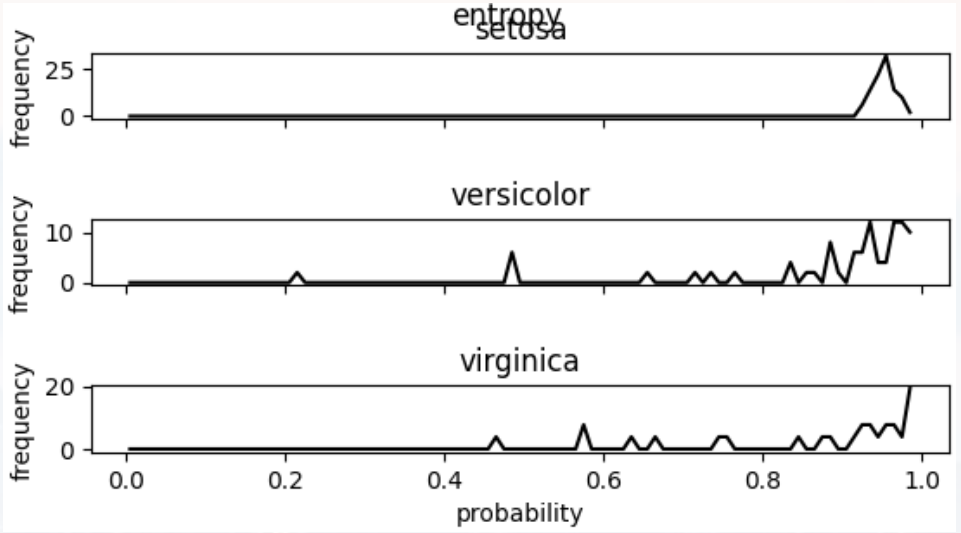


full **4D** data set

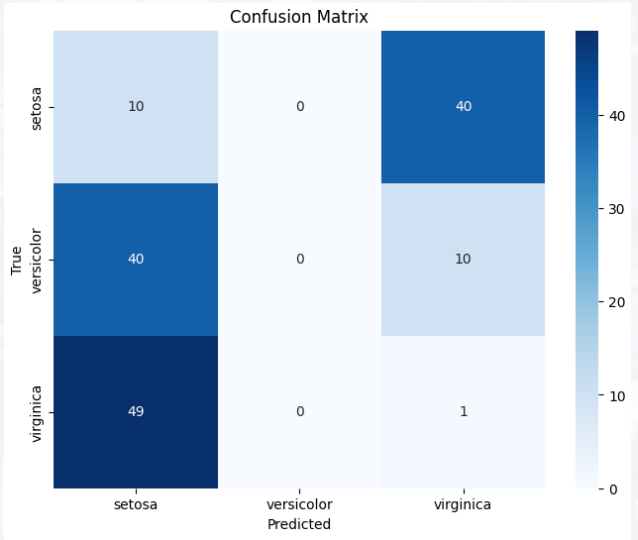
3) evaluating the model

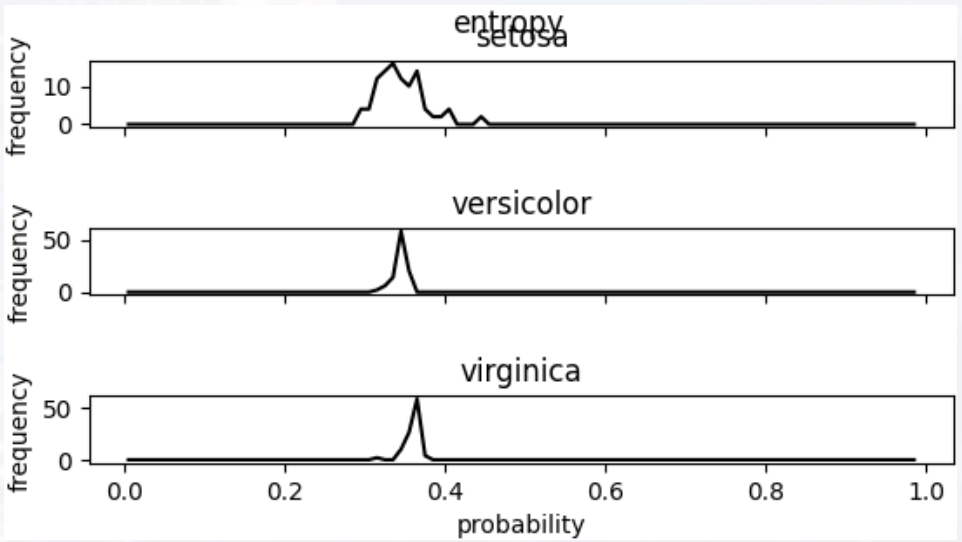
polynomial (n = 3)
accuracy: 97.3%





sigmoid
accuracy: 7.3%

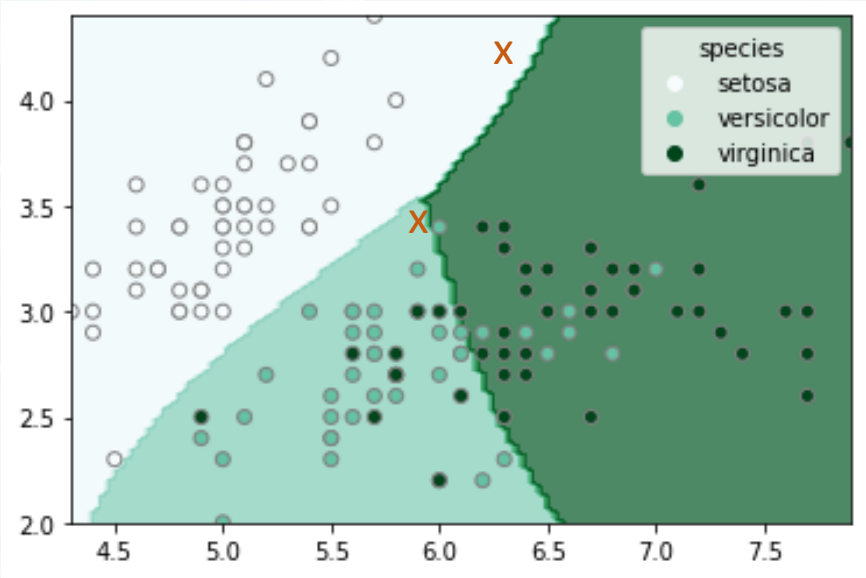




run: SVM.py to visualize the prediction

4) applying the model to a new data set

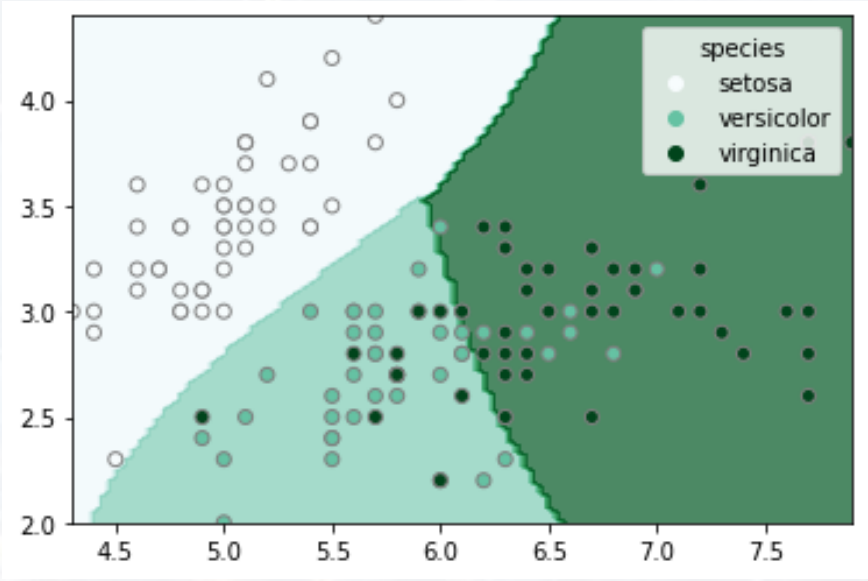
```
ypred = outpoly.predict([[6, 3.5],[6.3, 4.5]])
```

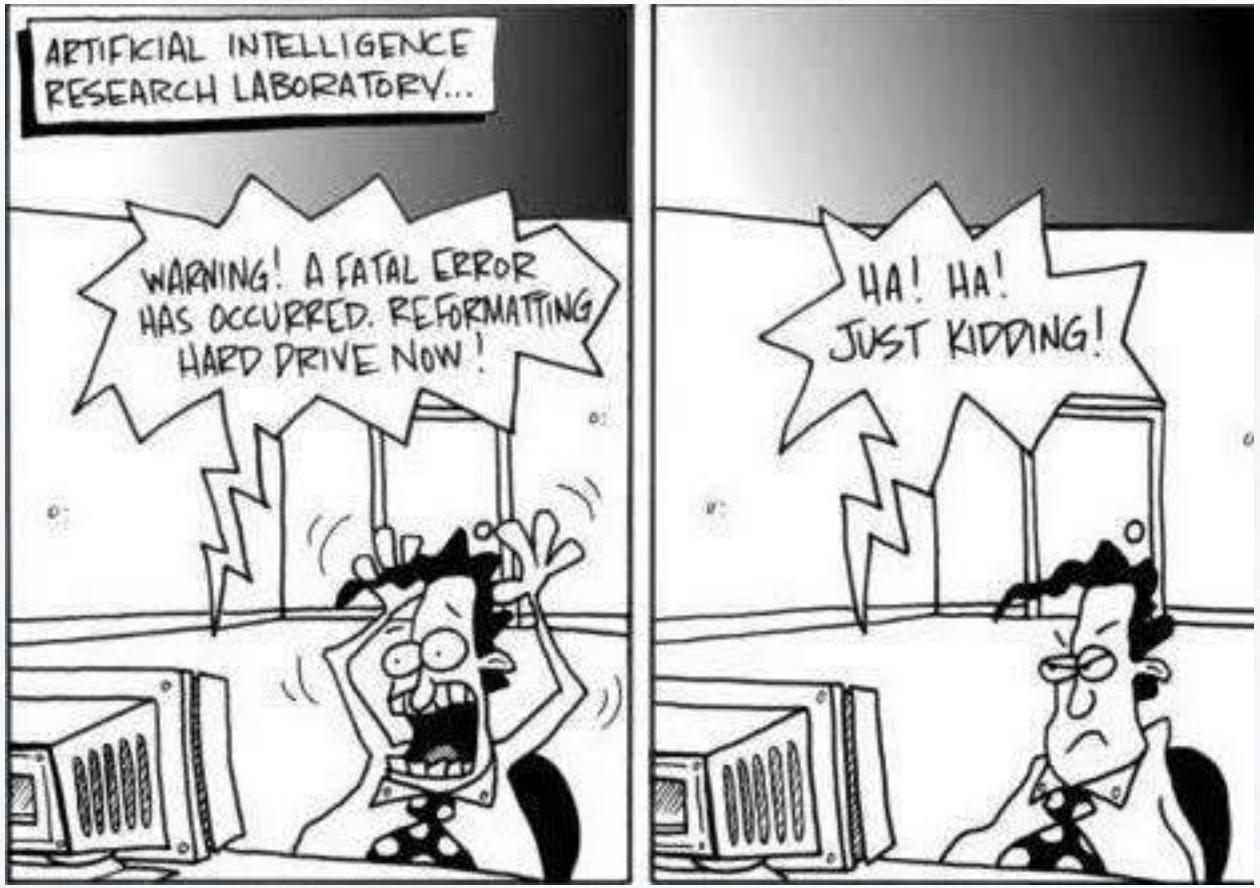


summary:

- simple and fast
- supervised
- *kernel* has to be given
- problems if cluster have unusual shapes

(elongated, hollow inside, scattered)





Concept

Supervised Learning

- Support Vector Machine
- K-nearest

Unsupervised Learning

- K – means
- Gaussian Mixture Models

idea: 1) determine the **mode** of the categories of the *k* nearest neighbors
2) assign new data points to this category
3) **supervised** learning
4) has to be given: ***k***

most *k* "neighbors" are trees
→ a new object must be a tree too

most *k* "neighbors" are buildings
→ a new object must be a building too

idea:

1) determine the **mode** of the categories of the *k* nearest neighbors
2) assign new data points to this category
3) **supervised** learning
4) has to be given: ***k***

$$\hat{Y} = mode\left\{Y_i\right\}_{x_i \in N_k(x)}$$

k neighborhood of *x* $N_k(x)$

idea:

1) determine the **mode** of the categories of the *k* nearest neighbors
2) assign new data points to this category
3) **supervised** learning
4) has to be given: ***k***

idea:

1) determine the **mode** of the categories of the *k* nearest neighbors
2) assign new data points to this category
3) **supervised** learning
4) has to be given: ***k***

idea:

1) determine the **mode** of the categories of the *k* nearest neighbors
2) assign new data points to this category
3) **supervised** learning
4) has to be given: ***k***

```
import matplotlib.pyplot as plt
import numpy as np
```

our standard libraries

for having different distances available

```
from pyclustering.utils.metric import *
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
```

performing knears classification

calling the “iris” data set

1) + 2) running and fitting the model **(again: 2D here only for visualization!)**

```
X2D = X[:,0:2]
```

check out `Walk_Through_Knearest.ipynb`

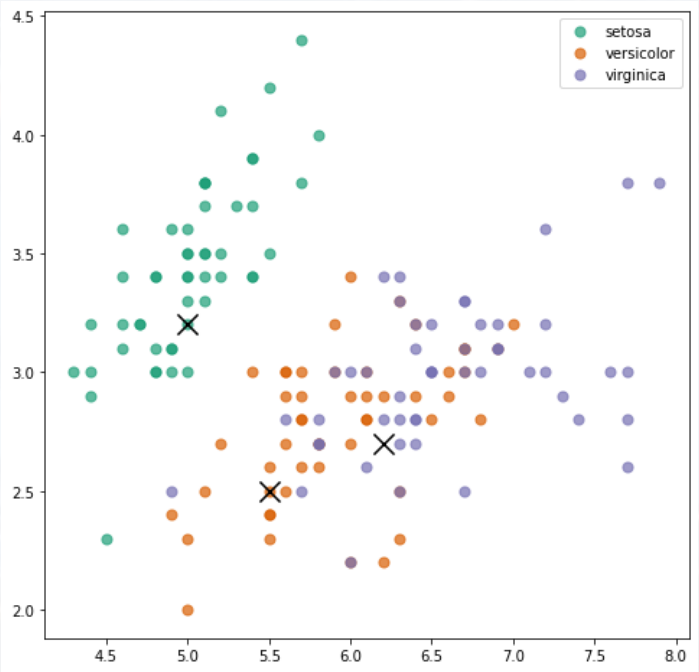
```
k   = 5
out = KNeighborsClassifier(k, \
                           metric = distance_metric(type_metric.EUCLIDEAN))
out.fit(X2D, Y)
```

3) + 4) after evaluation: predicting the classes of new data

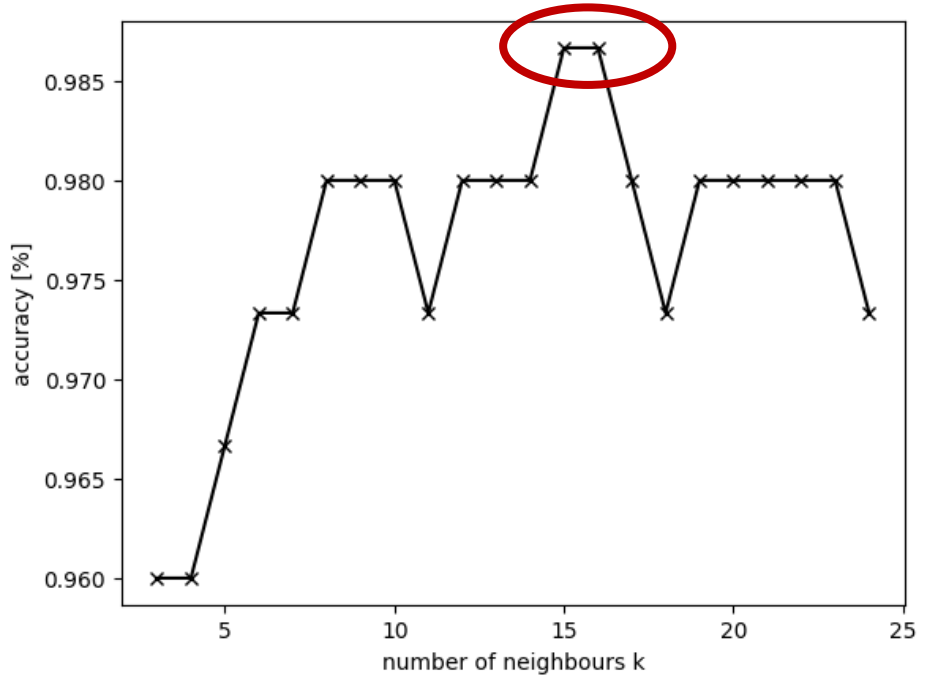
```
data_new = np.array( [[5, 3.2], [5.5, 2.5], [6.2, 2.7]] )
```

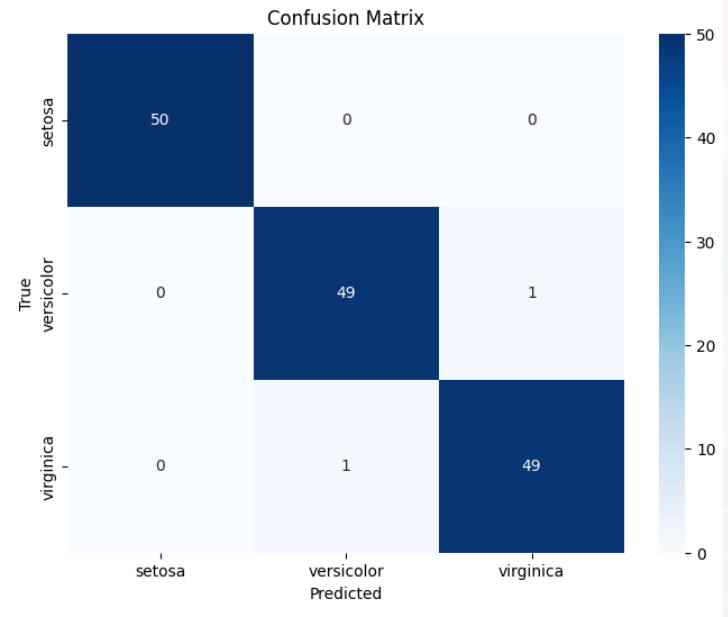
```
Pred      = out.predict(data_new)
Prob      = out.predict_proba(data_new)
print(Prob)
print(Pred)
```

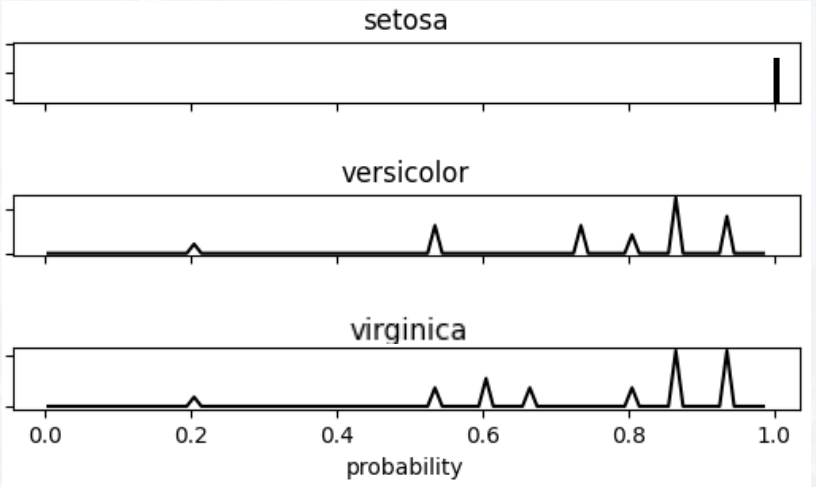
```
'setosa'   'versicolor'   'virginica'

  1.0          0.0            0.0
  0.0          1.0            0.0
  0.0          0.4            0.6
```



4D dataset and different number of neighbors **k**

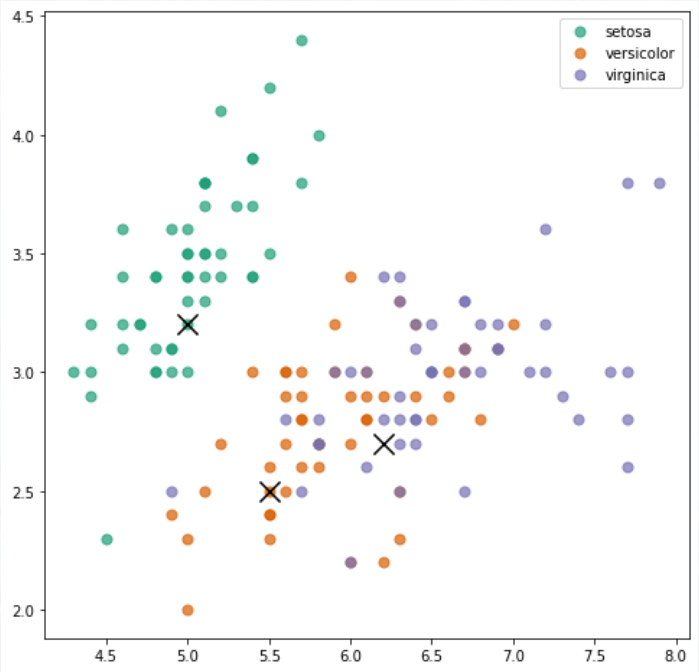






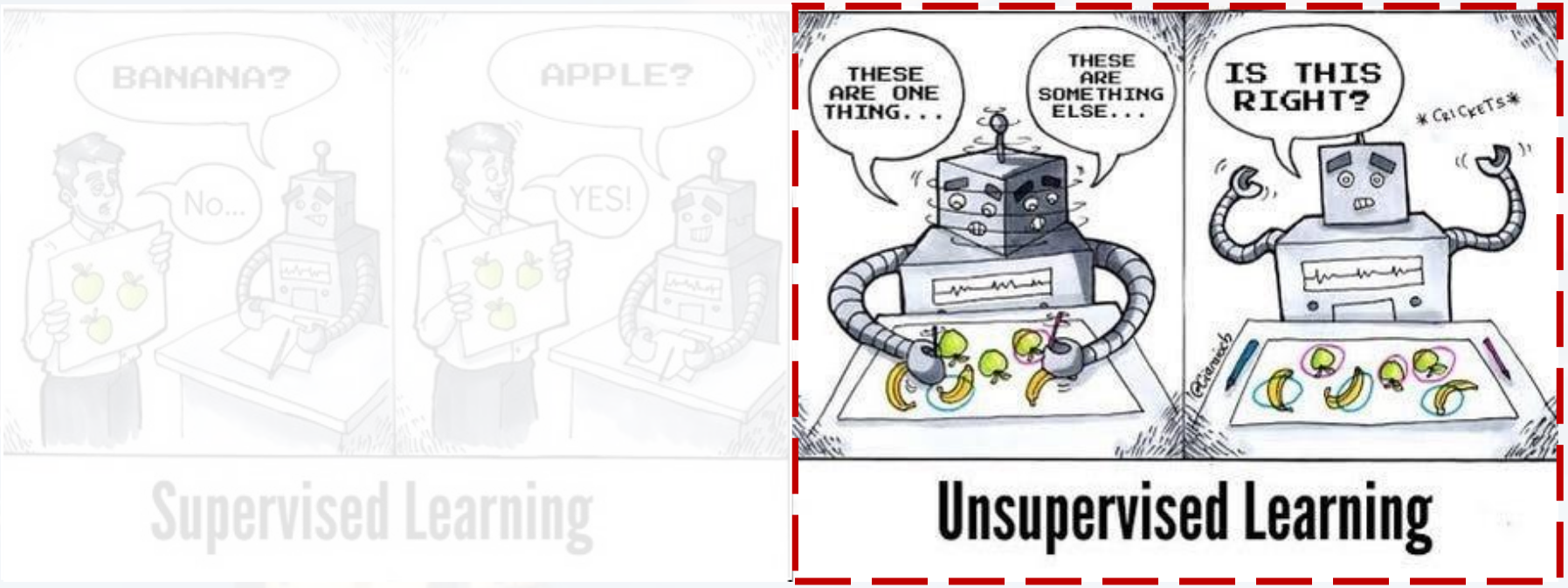
summary:

- simple and fast
- supervised
- *k* has to be given
- problems if cluster are overlapping



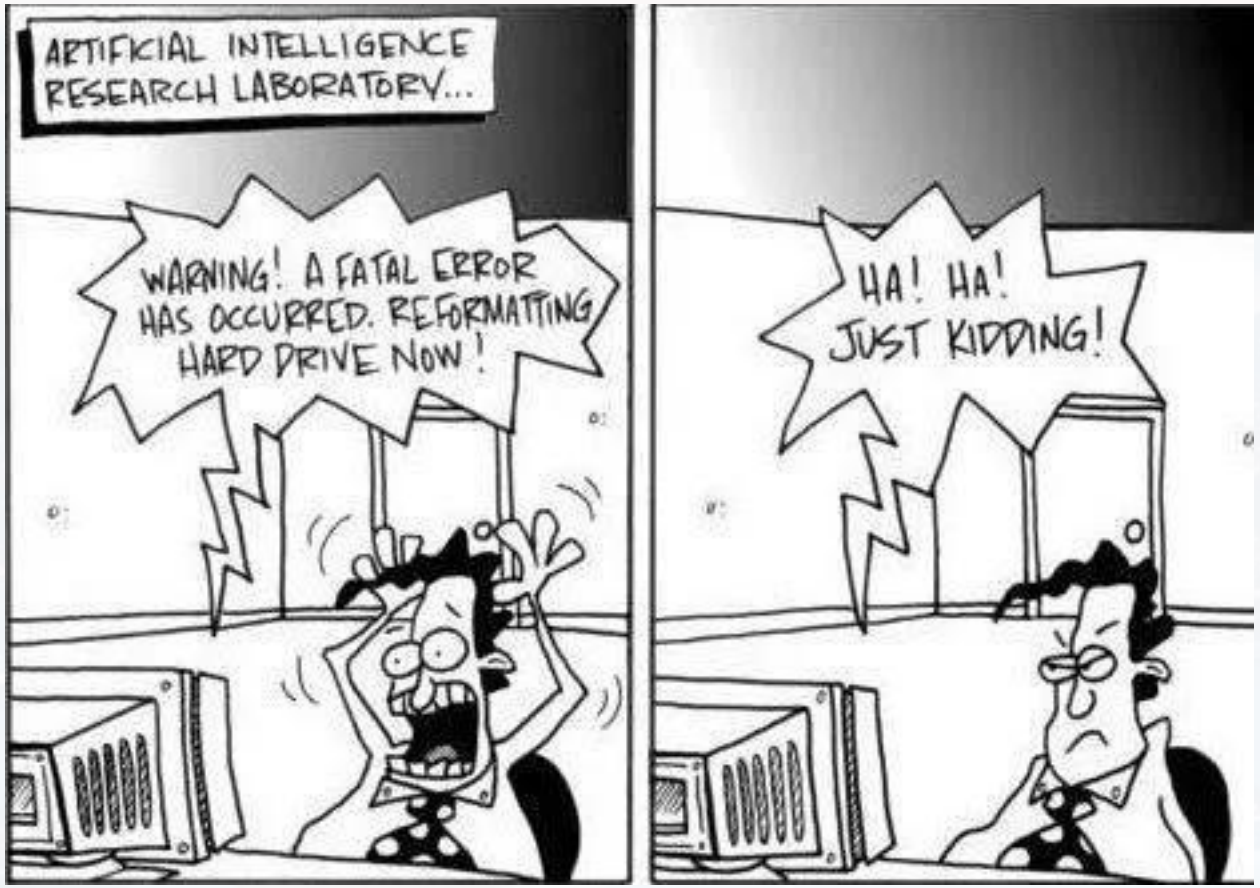
curve fitting, linear/logistic models:
training data set and a **test** data set…

no training data set required – we can start right away!



- Support Vector Machine
- K-nearest

- K - means
- GMM



Concept

Supervised Learning

- Support Vector Machine
- K-nearest

Unsupervised Learning

- K – means
- Gaussian Mixture Models

Workflow

1) setting up the model

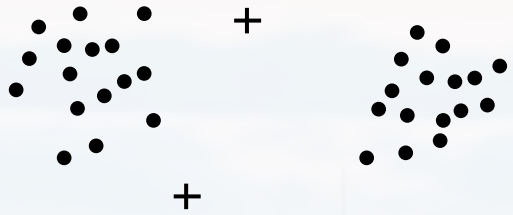
2) fitting the model: **X** are the **features** (sepal lengths/widths and petal lengths/widths),
~~**Y** are the **classes** (setosa, versicolor, virginica)~~

3) evaluating the model

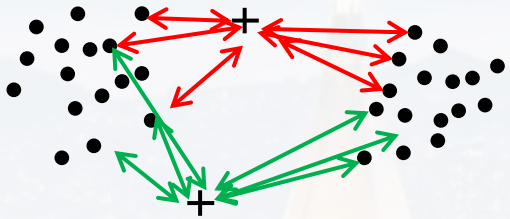
4) applying the model to a new data set

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

idea:



a) assign *k* means randomly

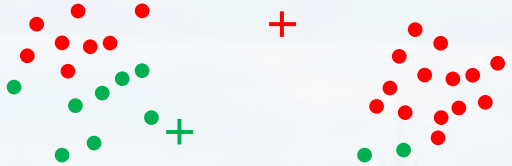


b) calculate *distance* from each point to each mean

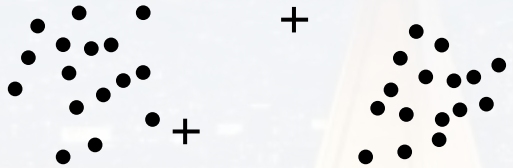
c) assign each point to its closest mean

d) update the means accordingly

idea:



d) update the means accordingly



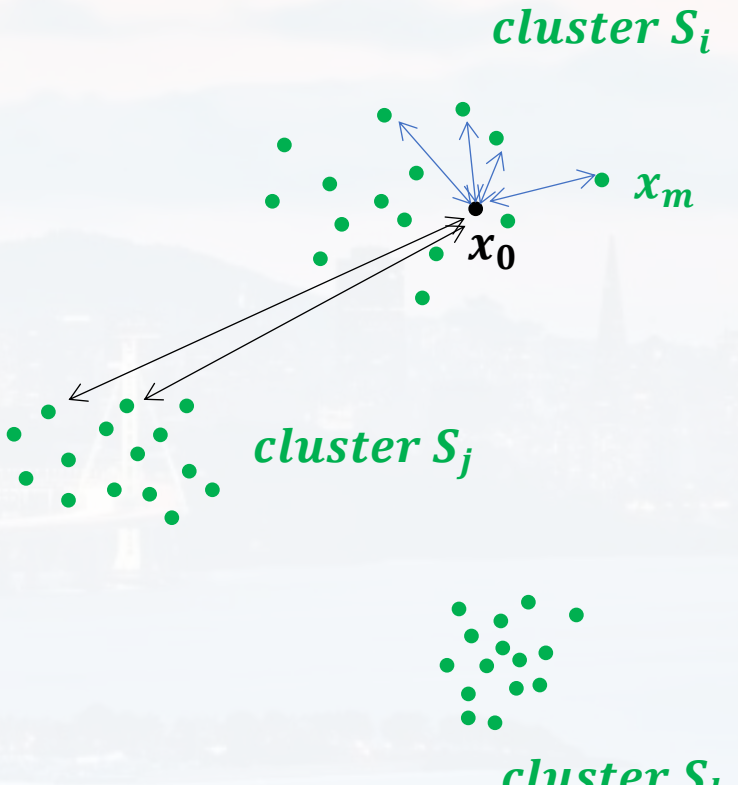
e) go back to b)

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*** $\boldsymbol{S_j}$***)***

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



- average over all points → ψ_{tot}

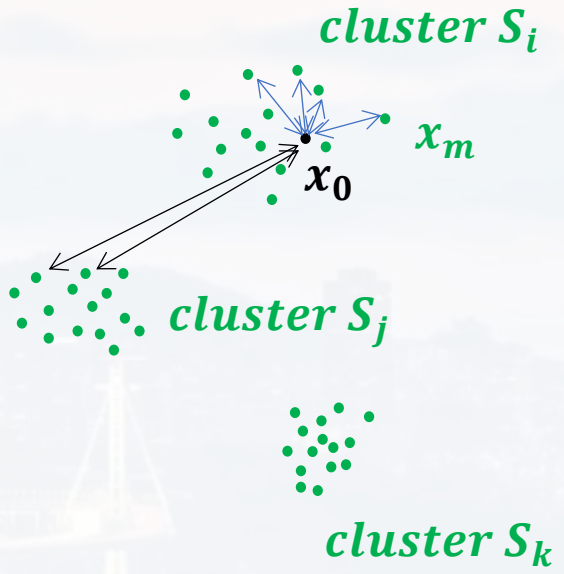
if	$\psi_{tot} = 0.75 \ldots 1.00$	→ well clustered
	$\psi_{tot} = 0.50 \ldots 0.75$	→ medium clustered
	$\psi_{tot} = 0.25 \ldots 0.50$	→ poorly clustered
	$\psi_{tot} < 0.25$	→ data has no structure

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*** S_i
  vs distance d_2 to ***closest cluster (here*** S_j***)***

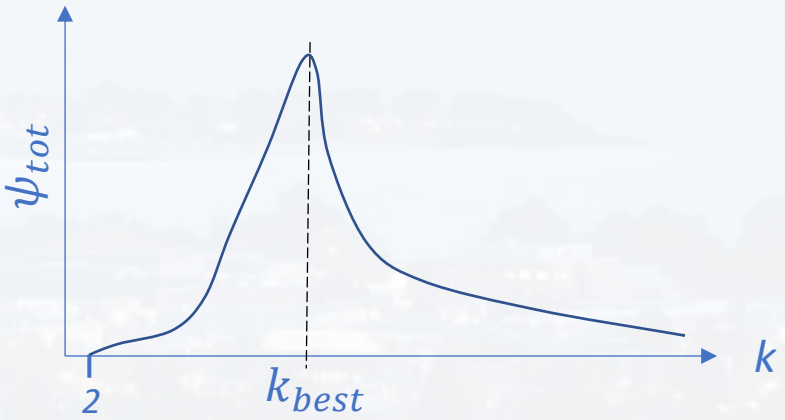
$$\Psi(x_0) = \begin{cases} 0 & if\ d_1 = 0 \\ \dfrac{d_2 - d_1}{max[d_1; d_2]} & \end{cases}$$



- average over all points → ψ_{tot}

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

ideal world →



```
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 *
```

for having different distances available

```
from nltk.cluster.kmeans import KMeansClusterer
```

performing k-means

```
from sklearn.metrics import silhouette_samples, silhouette_score
```

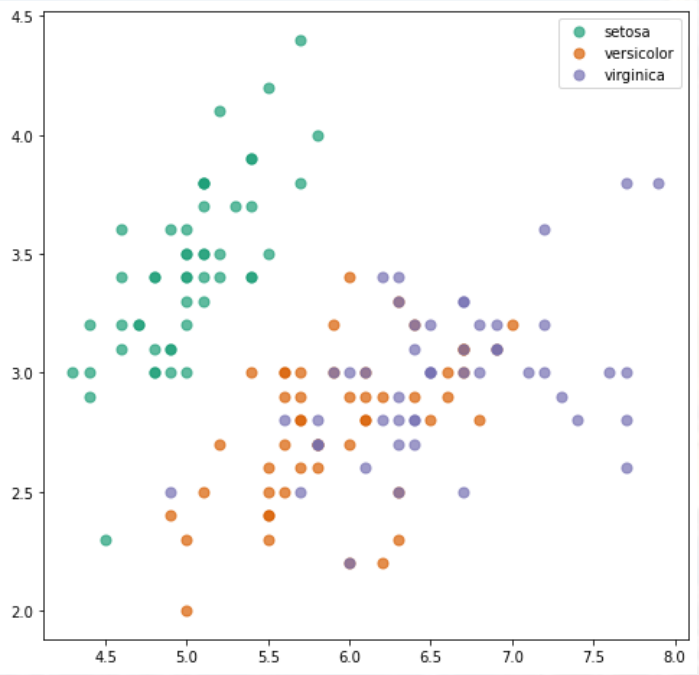
calculating silhouette coefficient for different k

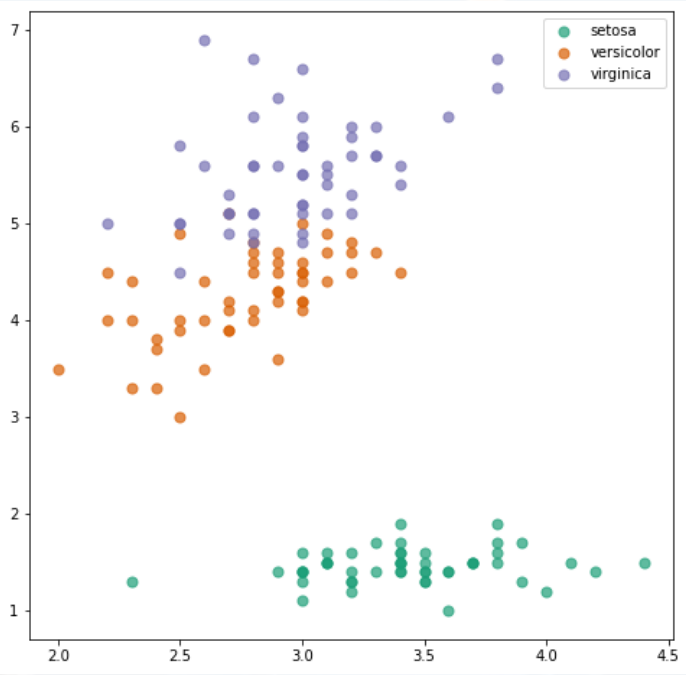
check out the Jupyter Notebook **`Walk_Through_Kmeans.ipynb`**

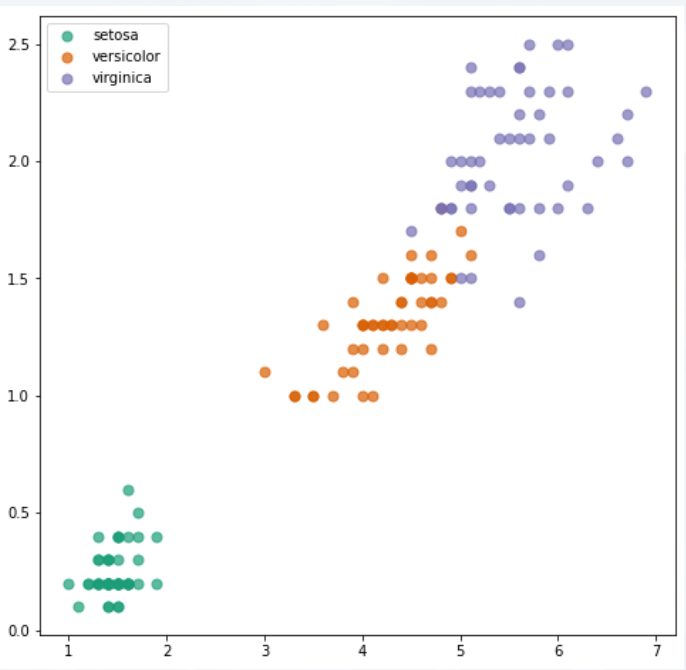
- **plotting the data**
- running k-means
- evaluating the result

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

4D dataset → plotting two components







```
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()
```

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

we need to “guess” the number of cluster

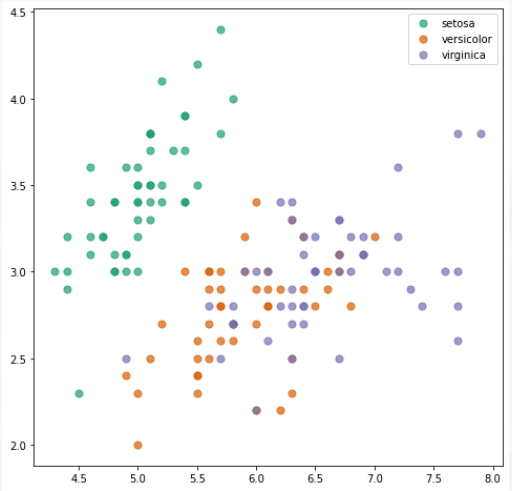
the **initial** means are assigned randomly.
→ repeat the procedure 25 times
→ avoiding local minimum,

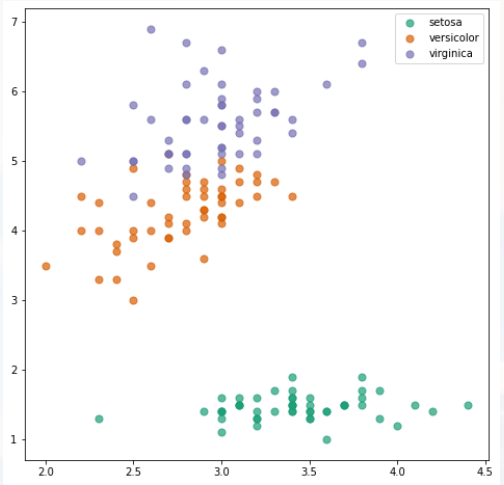
the features are measured in cm, i. e. the correct distance to pick here is Euclidean

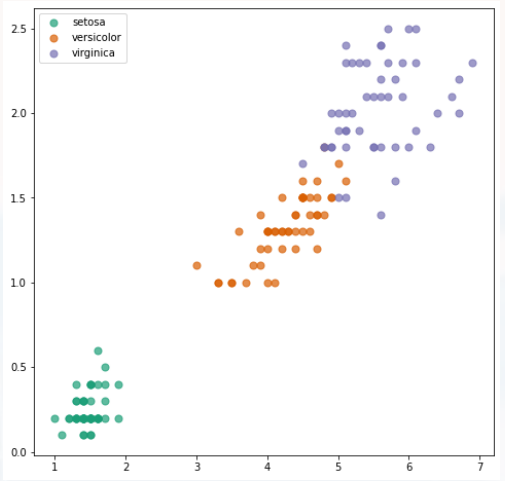
Machine Learning Fundamentals:

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

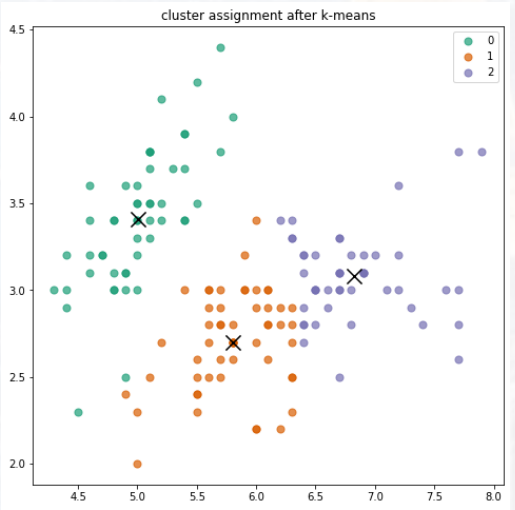
true classes

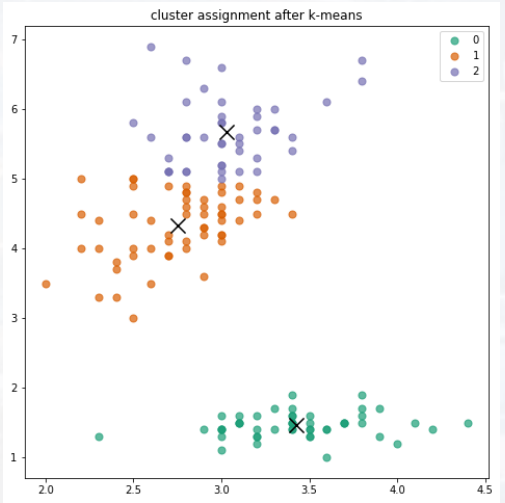


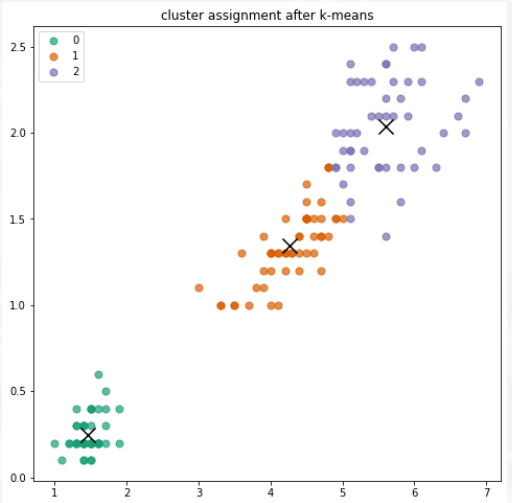




assigned classes

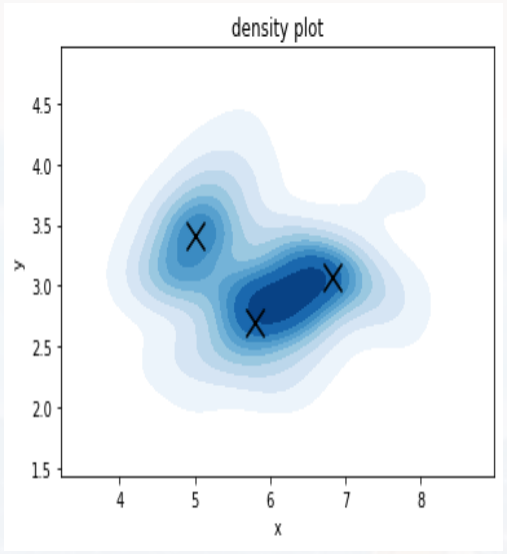


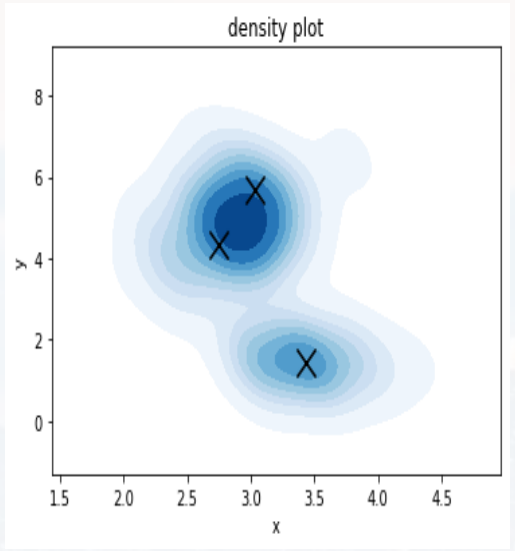


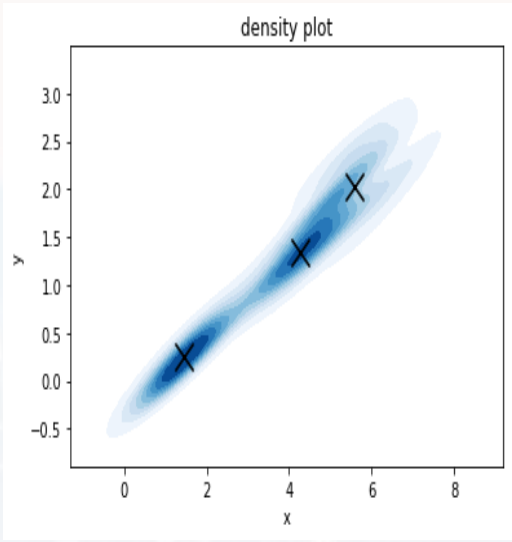


- plotting the data
- **running k-means**
- evaluating the result

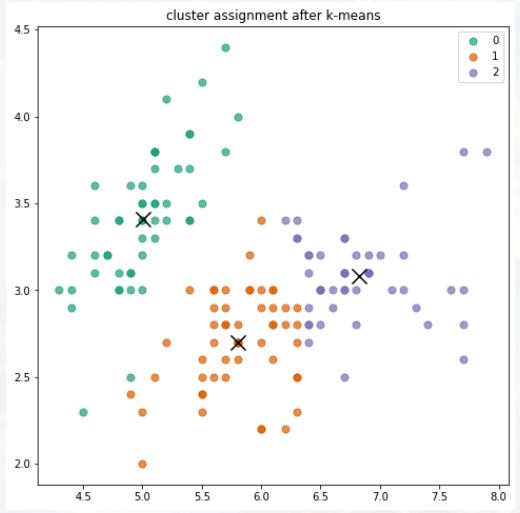
density

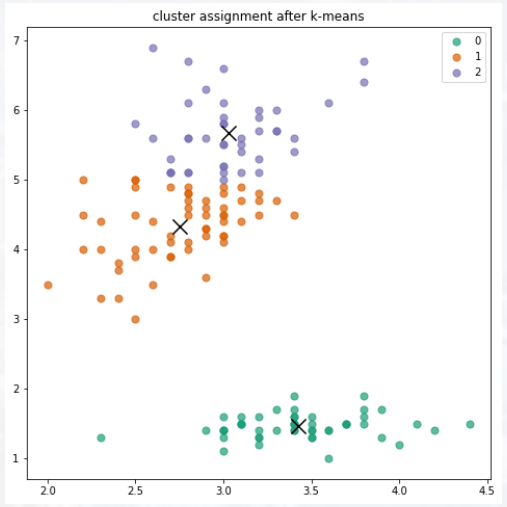


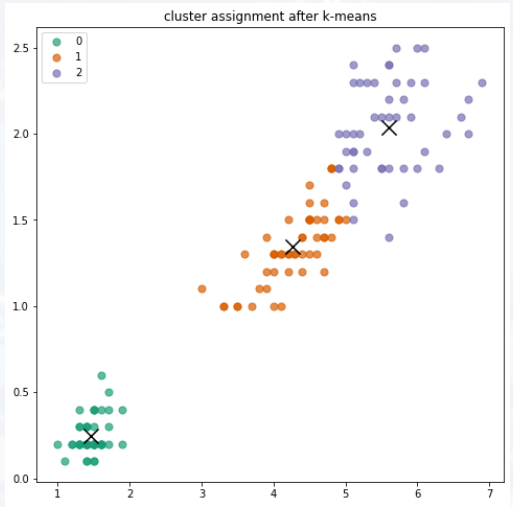




assigned classes



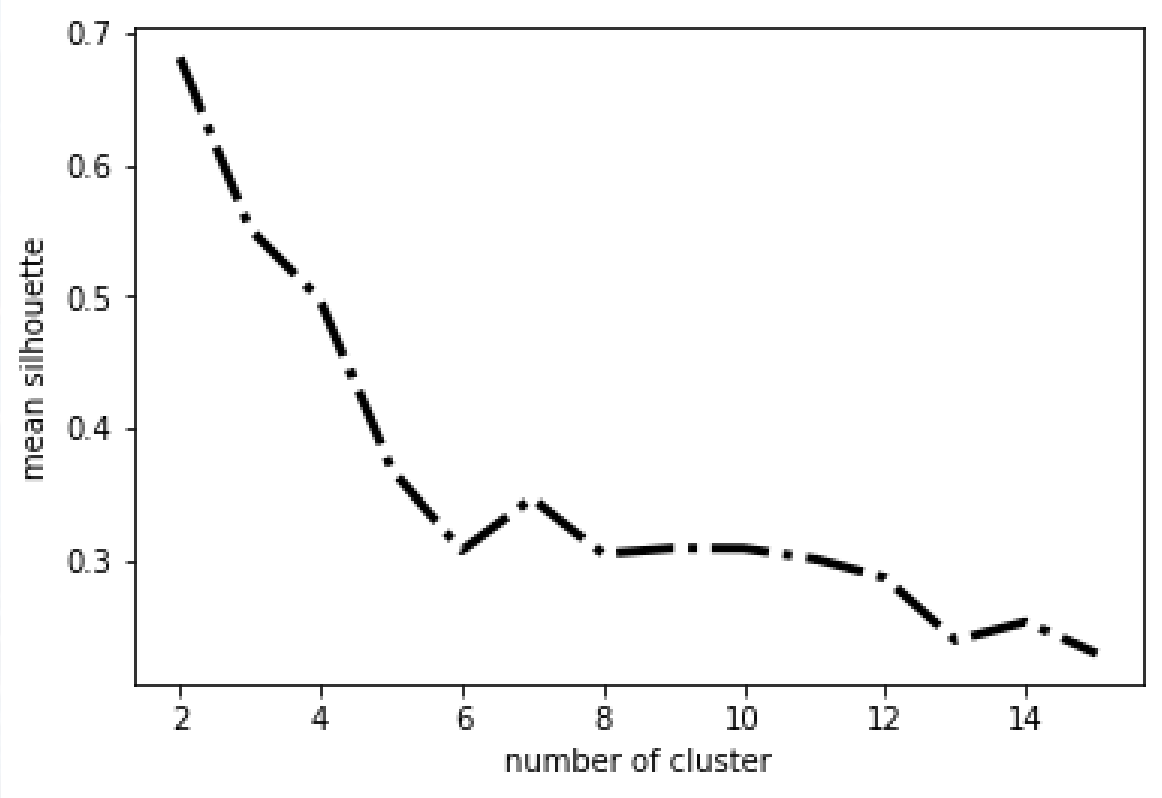


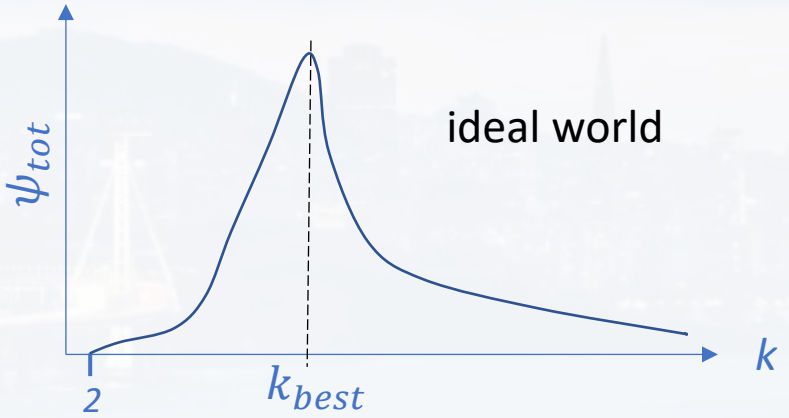


- plotting the data
- running k-means
- **evaluating the result**

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

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



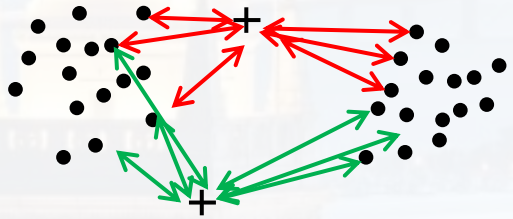


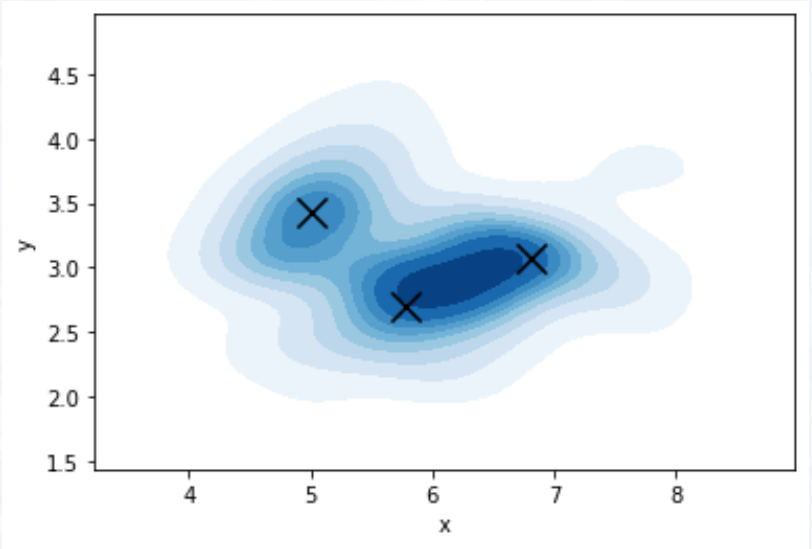
accuracy for 4D k = 3: 90%

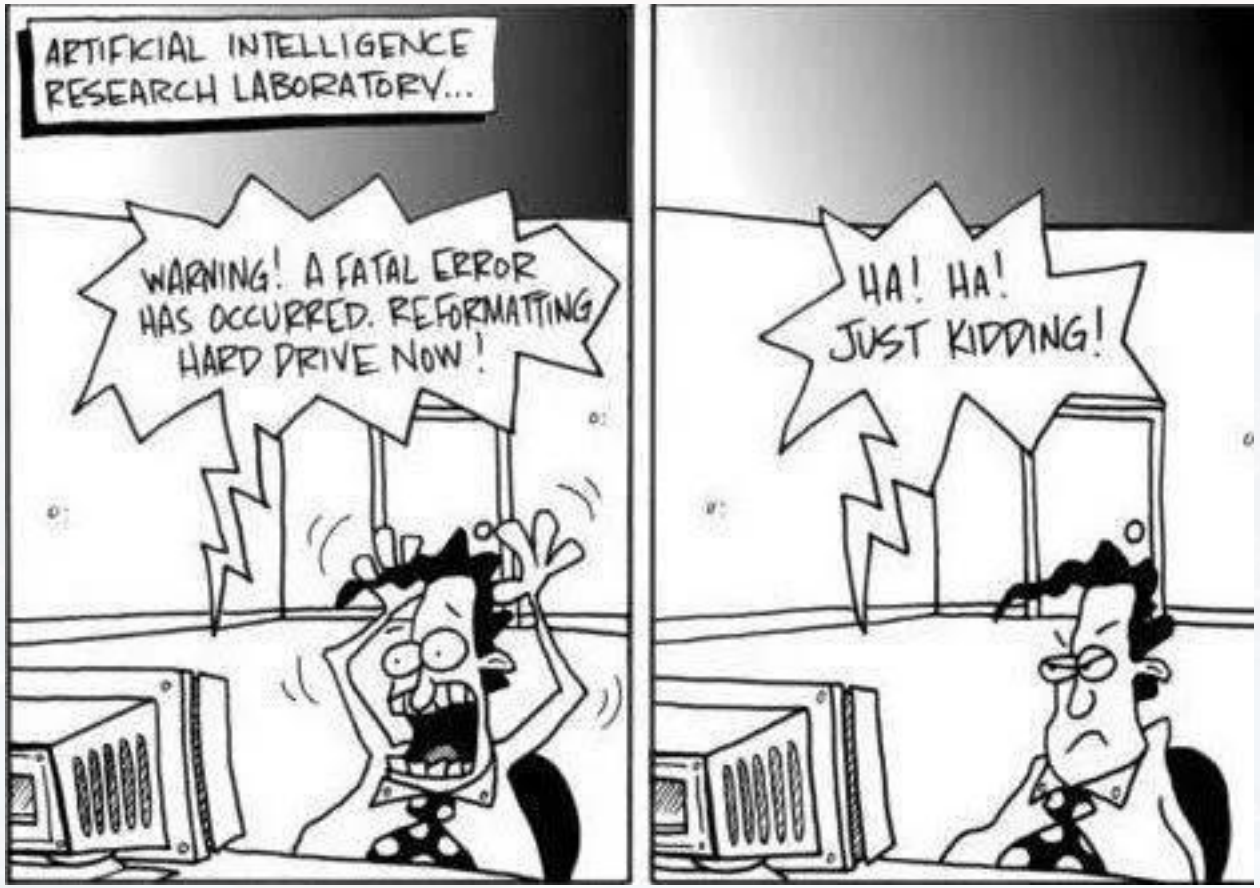
summary:

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

(elongated, hollow inside, scattered)







Concept

Supervised Learning

- **S**upport **V**ector **M**achine
- K-nearest

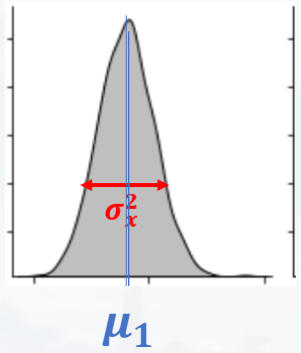
Unsupervised Learning

- K – means
- **G**aussian **M**ixture **M**odels

Gaussian Mixture Models

one feature

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



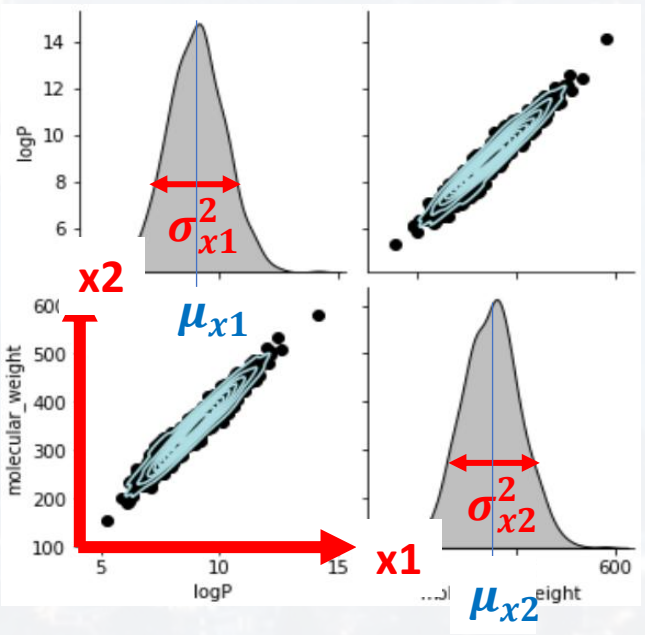
two features

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

covariance matrix

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

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

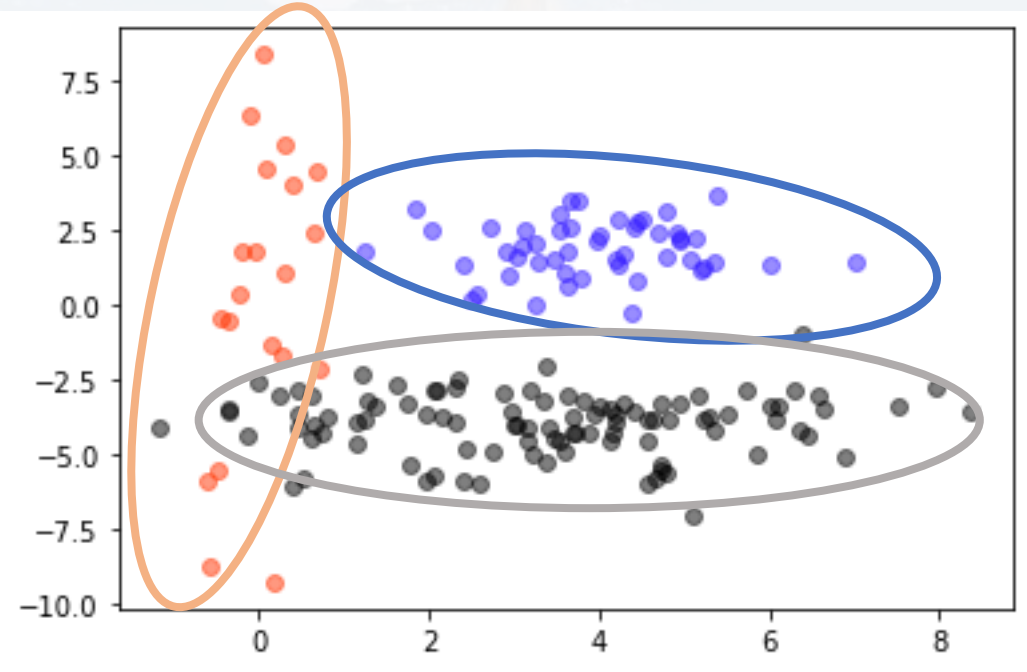


Gaussian **M**ixture **M**odels

n features

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

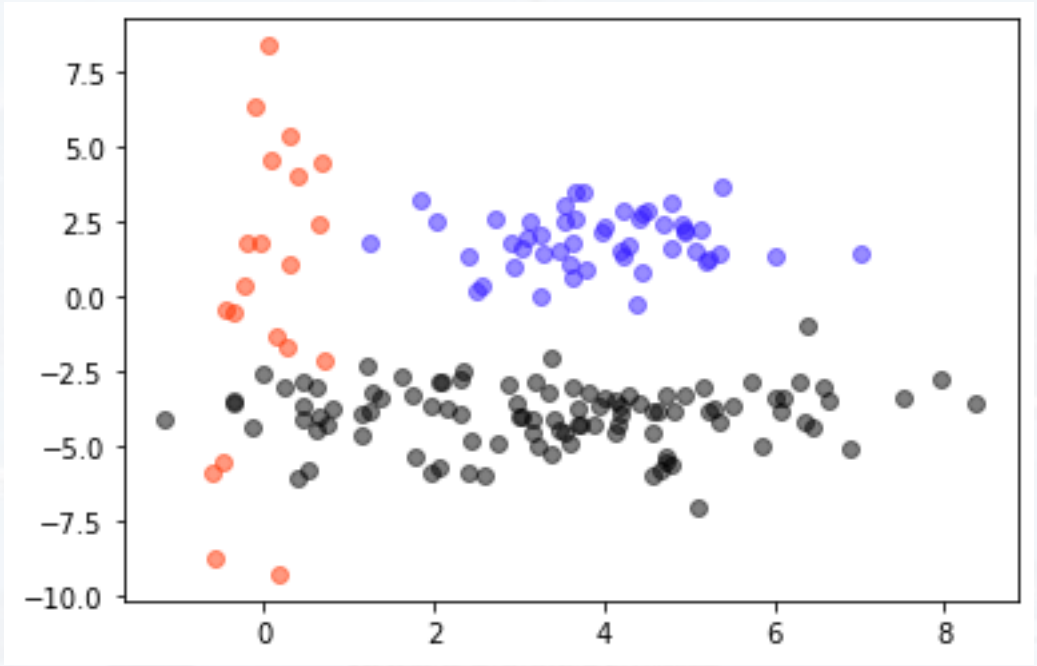
vectors *x* and μ



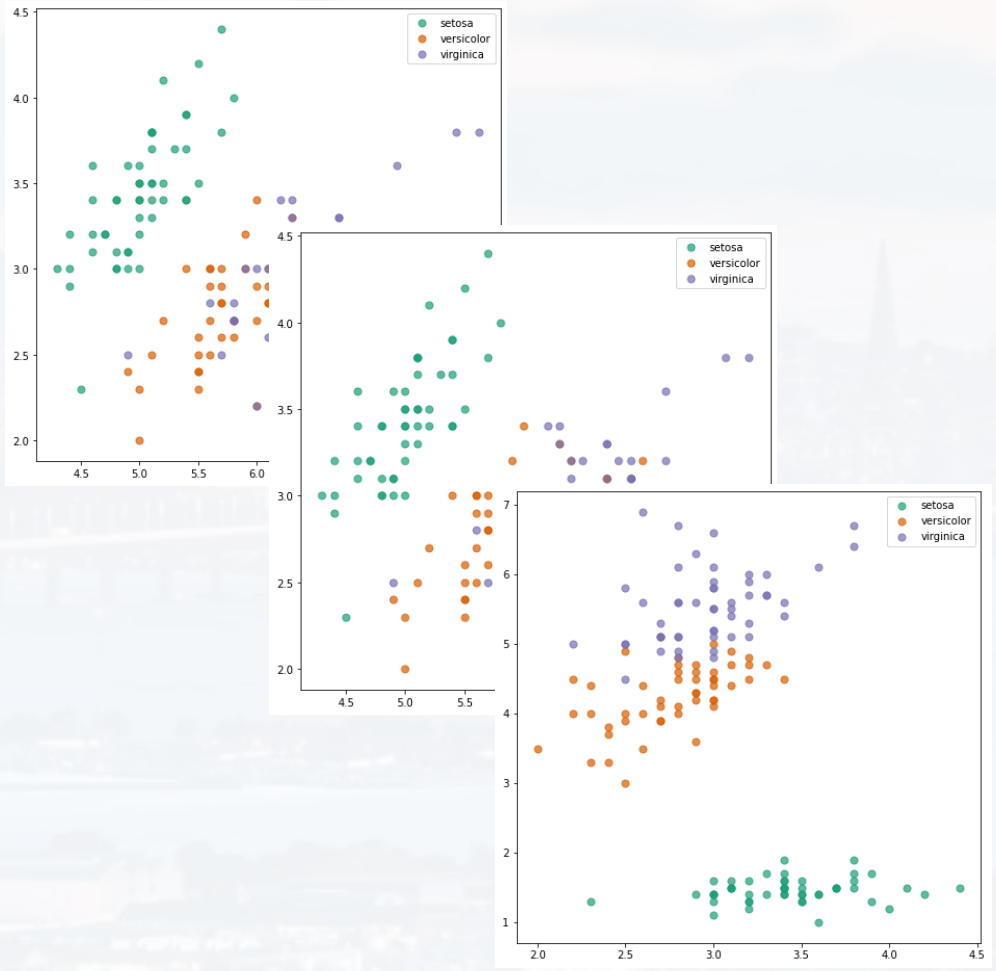
two features, *k=3* components

Gaussian Mixture Models

two features, *k=3* components



four features, *k=3* components



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

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

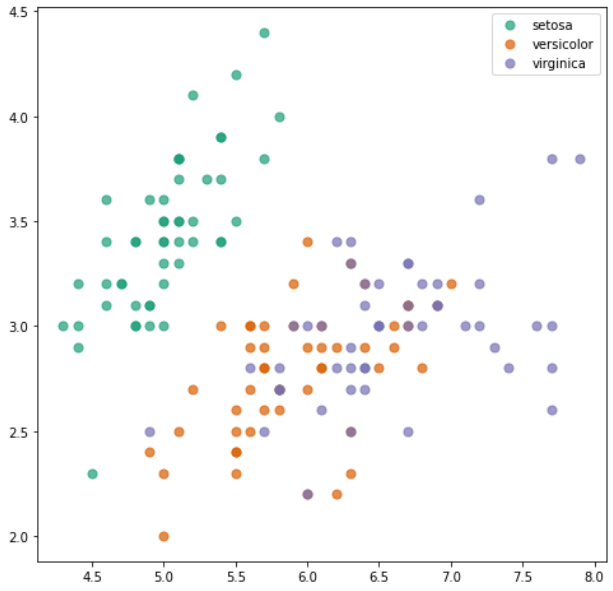
different algorithms:
- Bayesian
- **E**xpectation **M**aximization
- …

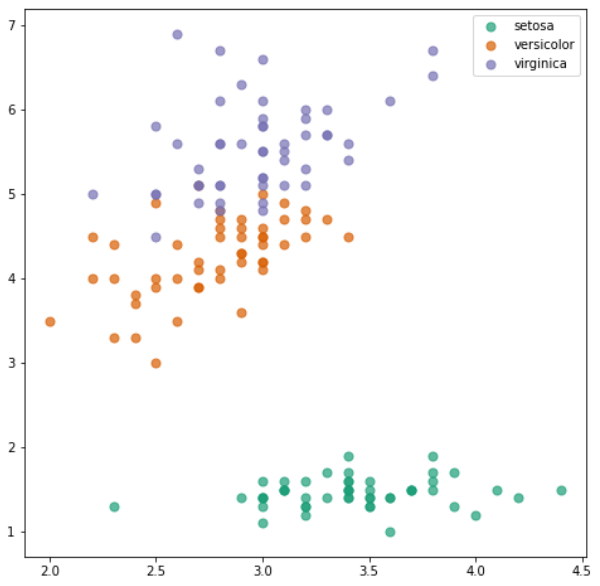
```
my_model   = GaussianMixture(n_components = k, random_state = 0).fit(X)

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

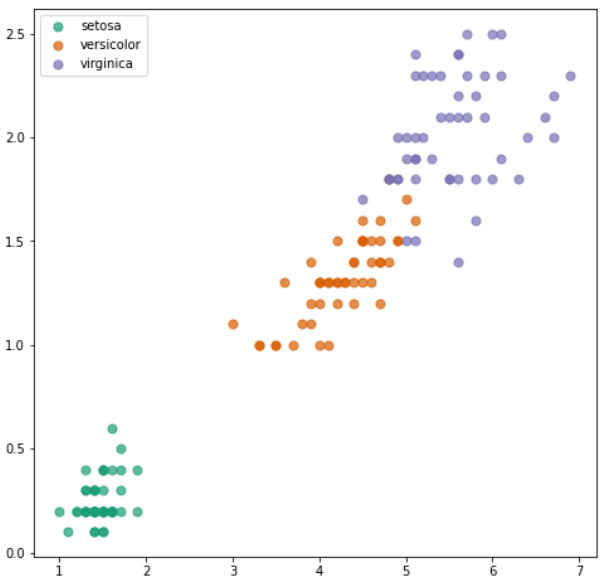
see **Walk_Through_GMM.ipynb**

setting initial labels randomly

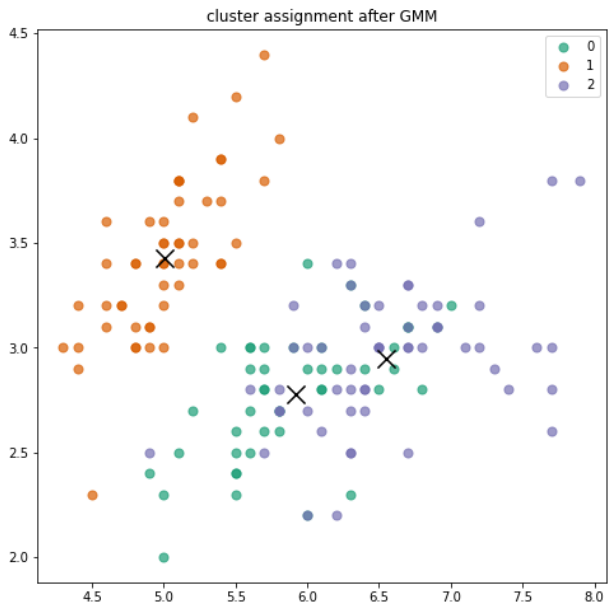
true classes

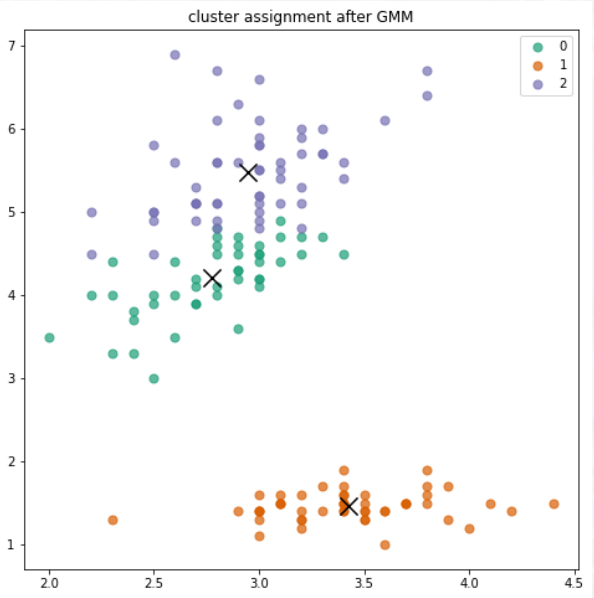




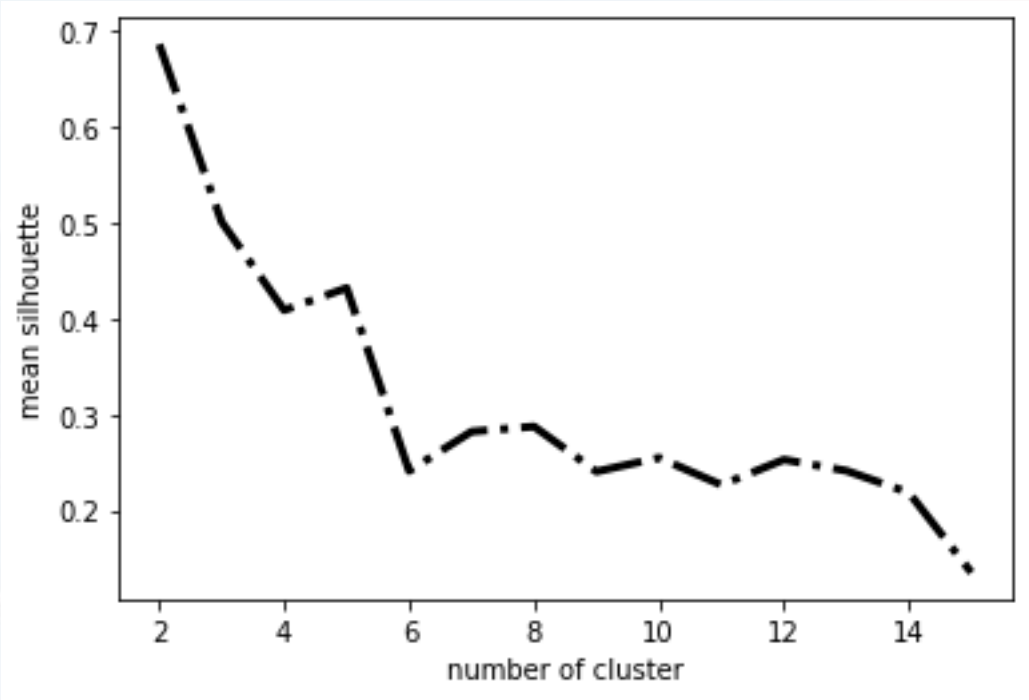


assigned classes





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



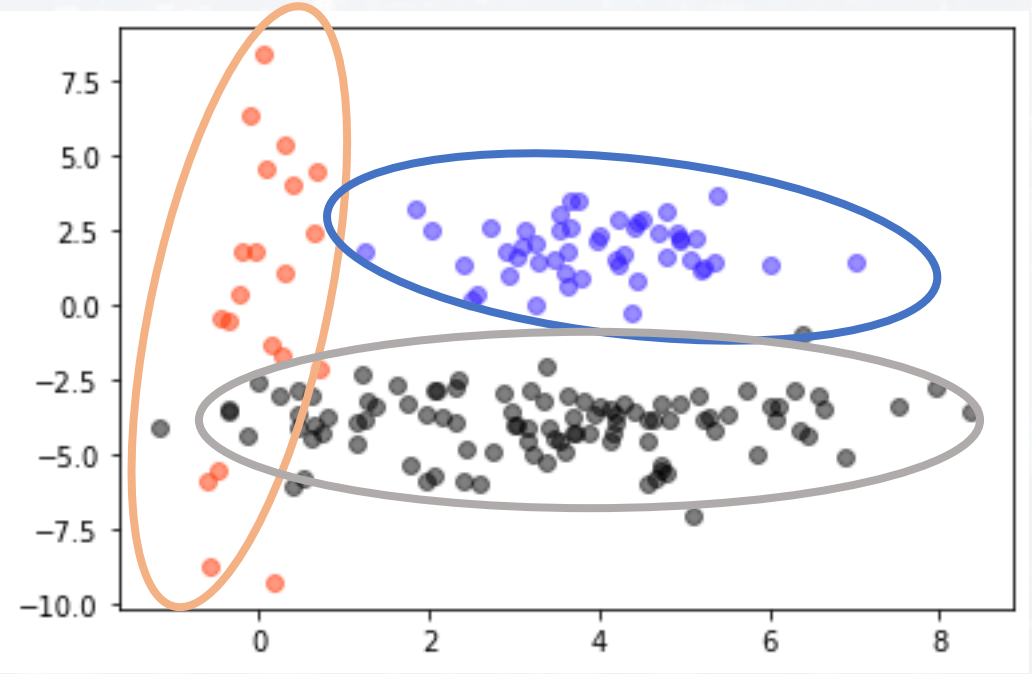
accuracy for 4D k = 3: 93%

summary:

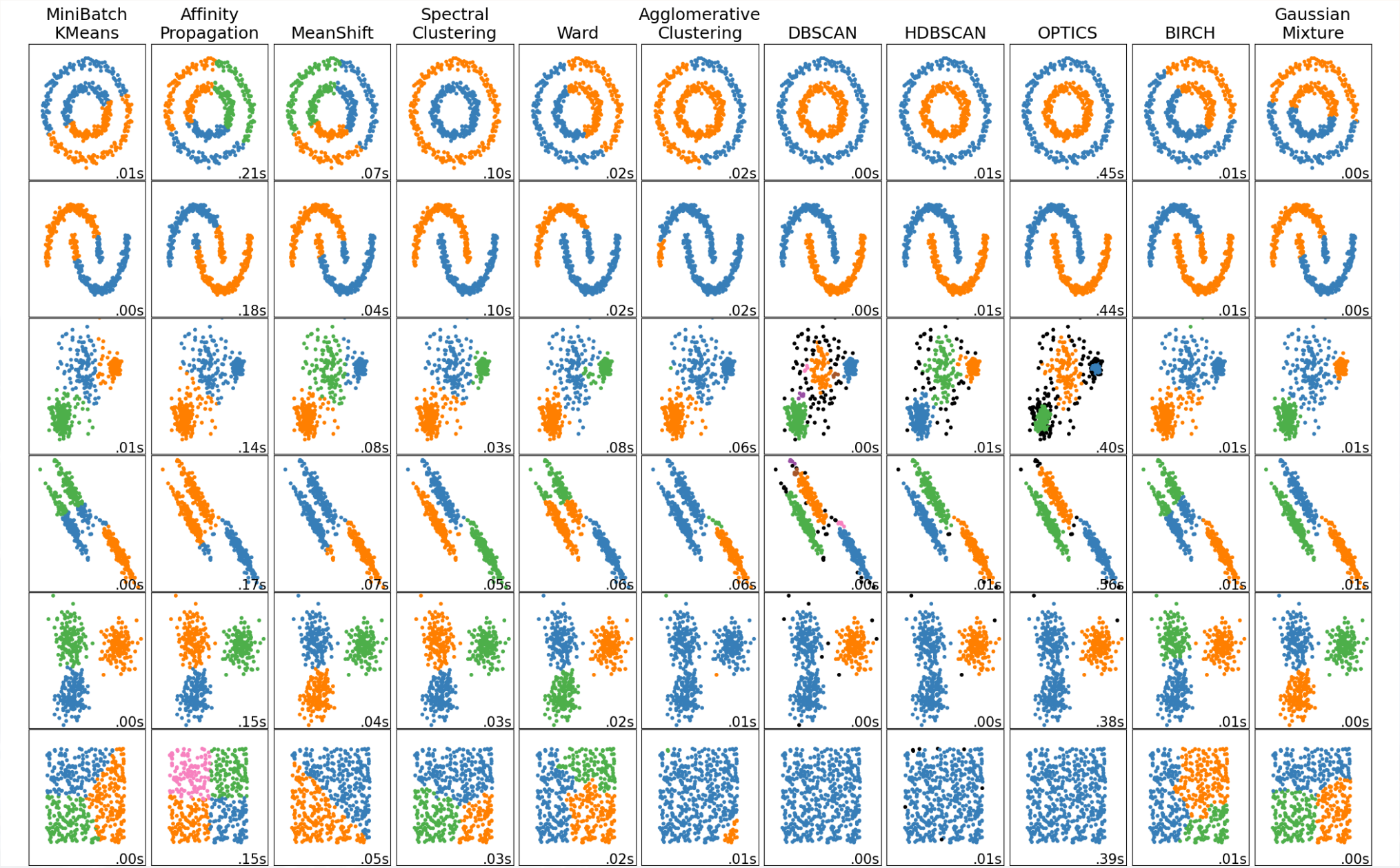
- simple and fast
- unsupervised
- *k* has to be given → silhouette for determining best *k*
- problems if cluster have unusual shapes

(elongated, hollow inside, scattered)

- Gaussian: max entropy



there is a lot more...

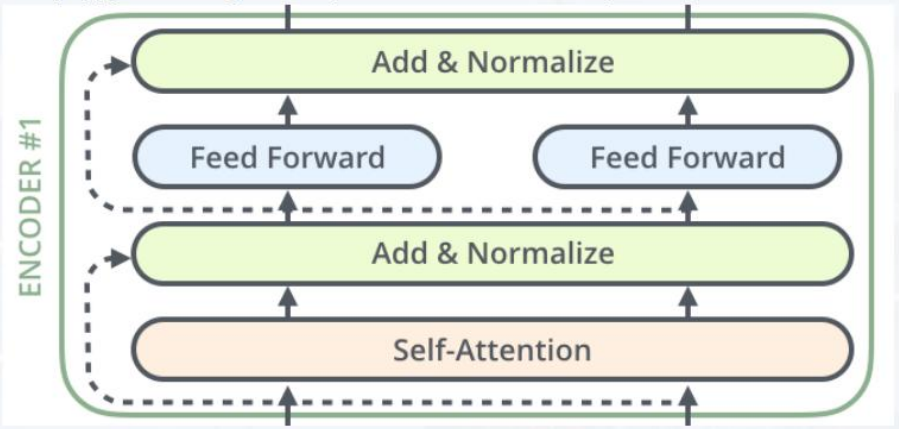


there is a lot more… Chem 277B **"Machine Learning Algorithms"**

Berkeley UNIVERSITY OF CALIFORNIA Berkeley UNIVERSITY OF CALIFORNIA Berkeley UNIVERSITY OF CALIFORNIA **LLM & Transformer:** Transformer Architecture

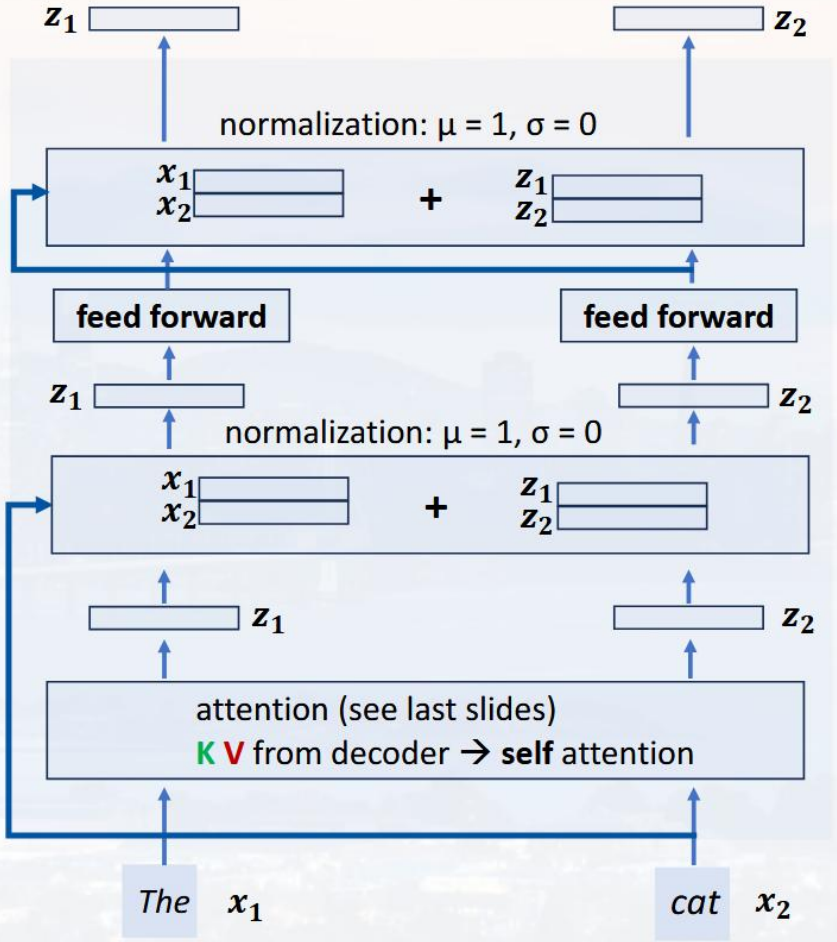
that was attention → now: encoder

https://jalammar.github.io/illustrated-transformer/



h

depend
the outp
may hav
lengths t
input fea



Thank you very much for your attention!