Mechatronic Engineering program:
Python for machine learning and data science

Regression, Classification & Clustering (revisited)

Ziemowit Dworakowski

AGH University of Krakow

What you should probably know already?

Outliers
(their sources, ways to deal with them)

Overlapping classes
(And how to deal with them)

Basic optimization methods
(Gradient, gradient with momentum, 1+1, grid search)

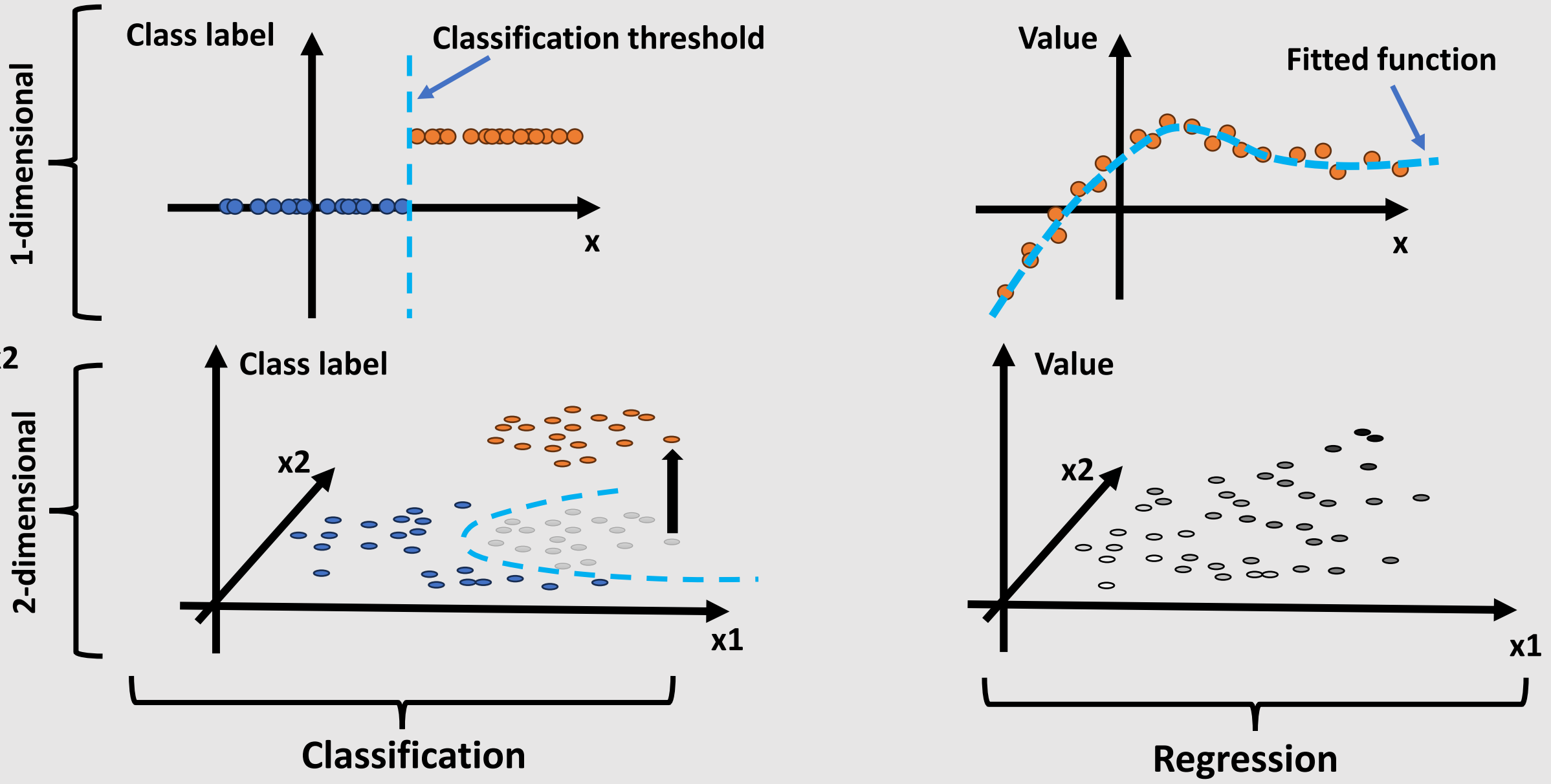
Overfitting
(And how to deal with it)

Basic classification algorithms
(logistic regression, kNN, manually configured decision tree, ANN)

Metaparameter configuration procedure
(2nd order optimization)

Basic regression algorithms
(linear and weighted linear regression, polynomial, ANN)

Classification / regression analogy



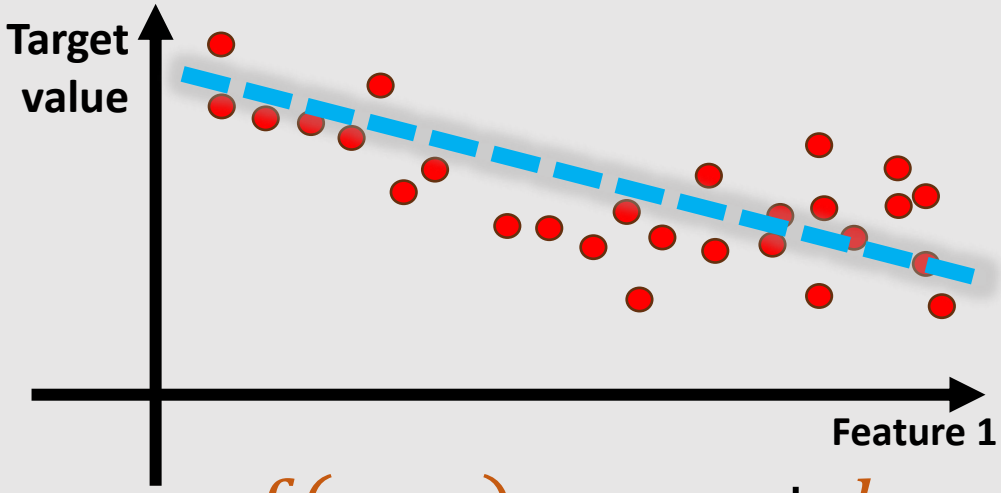
Regression recap:

We want to have a model that predict **Target value** based on features values and model parameters.

$$f(\boldsymbol{w}, \boldsymbol{x})$$

Linear model

3D+: using **hyperplane**



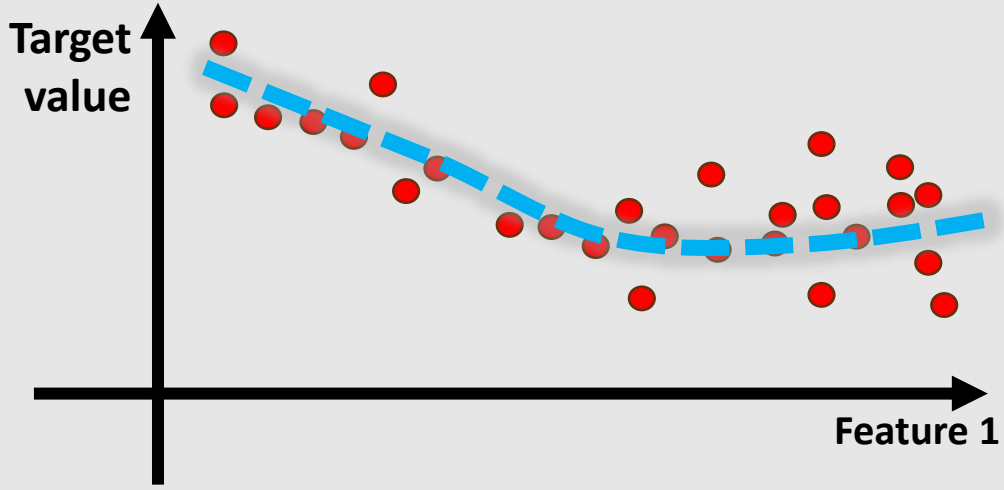
$$f(\boldsymbol{w}, \boldsymbol{x}) = \boldsymbol{w}\boldsymbol{x} + b$$

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

- Features
- Model parameters

Nonlinear model

3D+: using **surface**



We will still have a general $f(\boldsymbol{w}, \boldsymbol{x})$ form, but the relations will be nonlinear now and depending on the model.

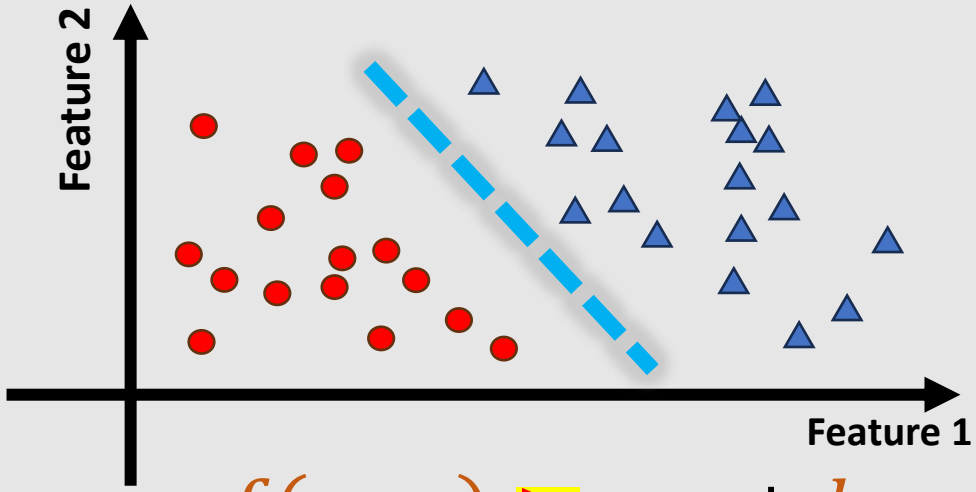
Classification recap:

We want to have a model that predict **Class label** based on features' values and model parameters.

$$f(\boldsymbol{w}, \boldsymbol{x})$$

Linear model

3D+: using **hyperplane**



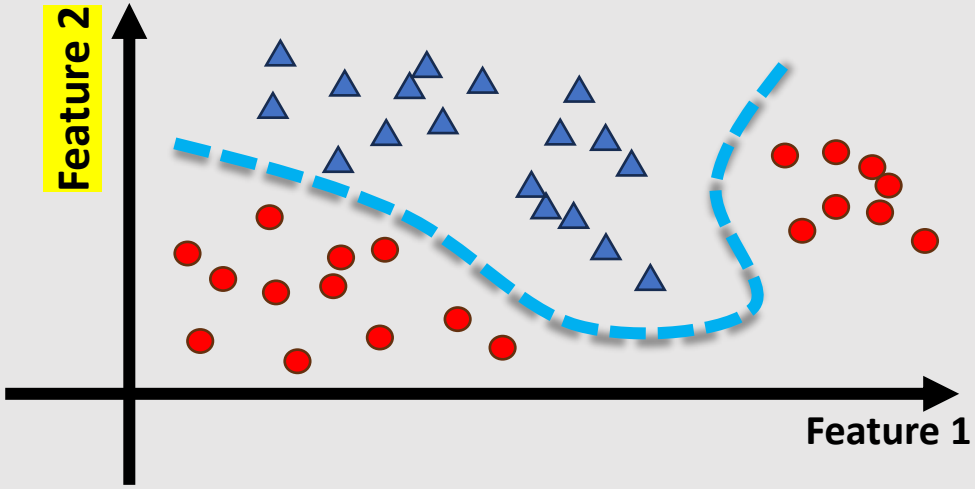
$$f(\boldsymbol{w}, \boldsymbol{x}) \geq \boldsymbol{w}\boldsymbol{x} + b$$

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b$$

- Features
- Model parameters

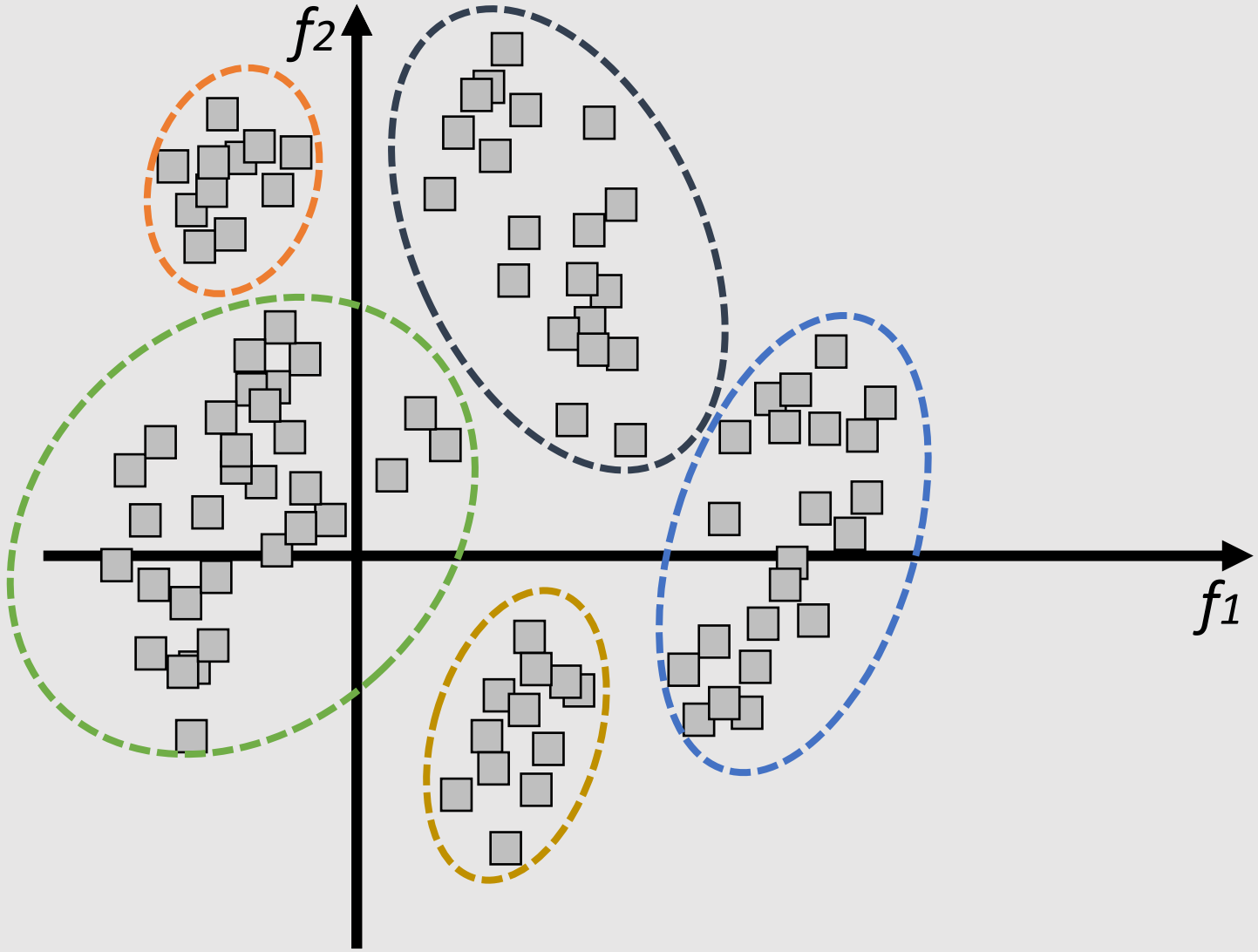
Nonlinear model

3D+: using **surface**

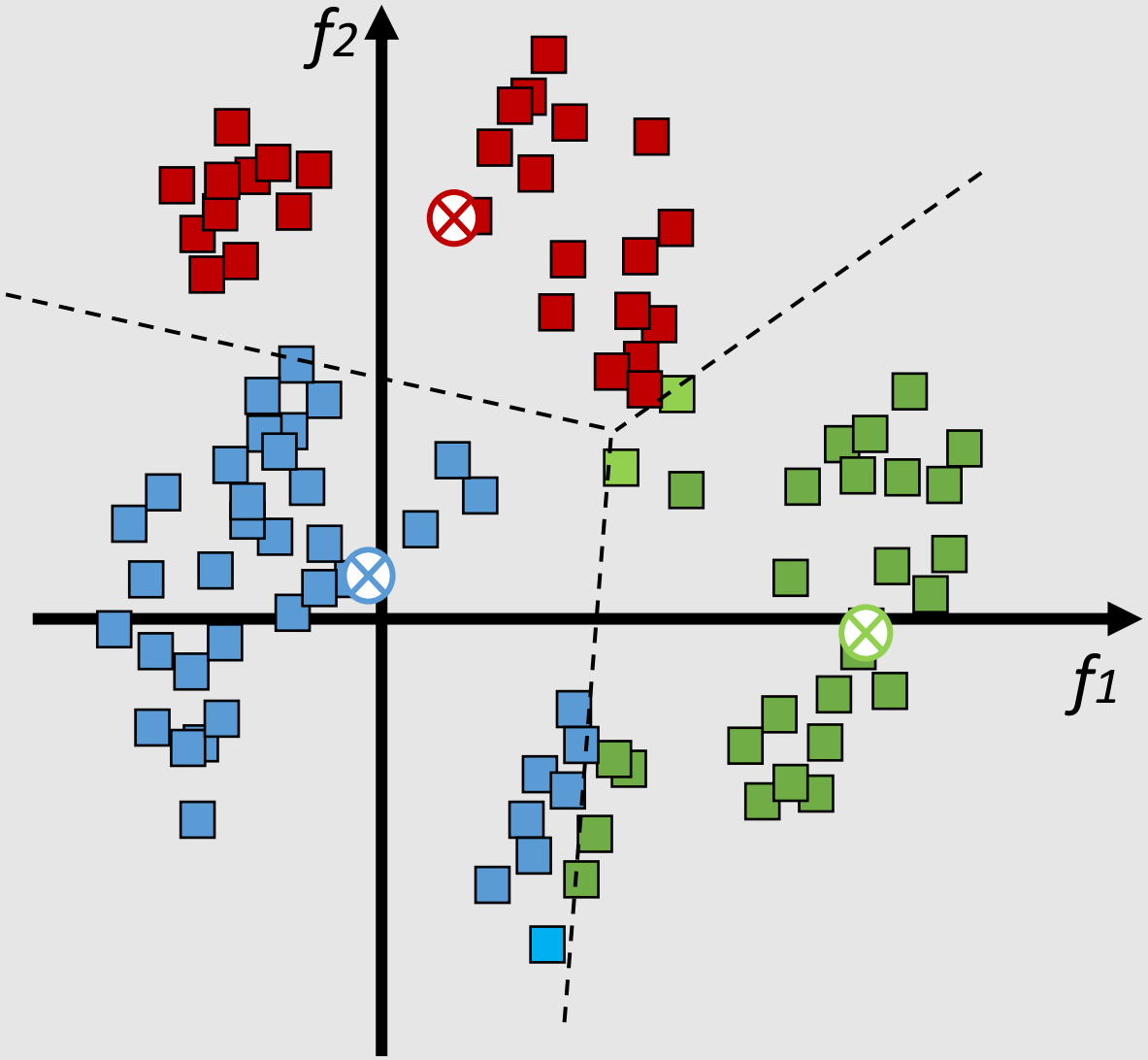


We will still have a general $f(\boldsymbol{w}, \boldsymbol{x})$ form, but the relations will be nonlinear now and depending on the model.

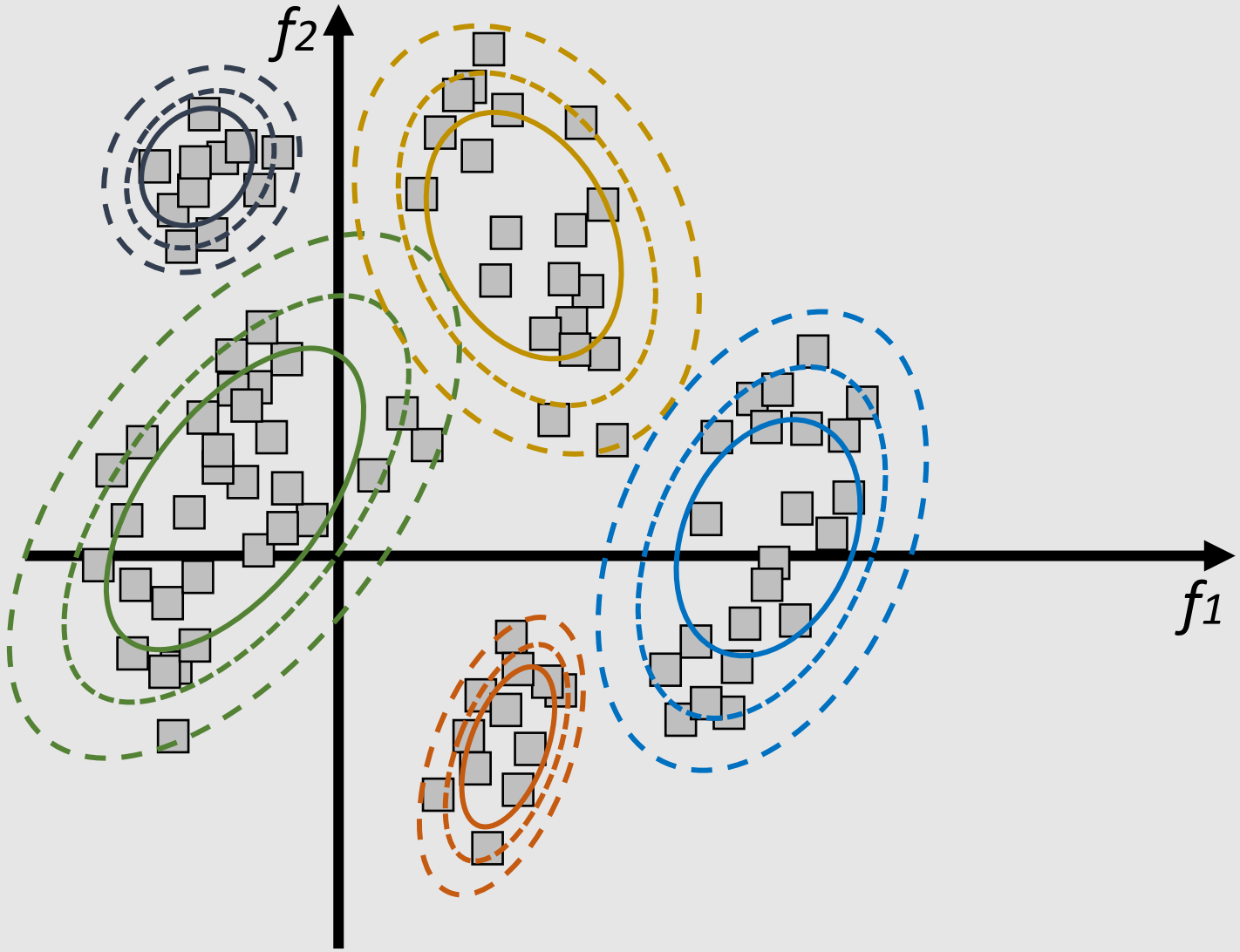
Clustering recap



K-means clustering



Gaussian mixture model



Model summary

Linear regression
Simple, straightforward, quick

Weighted linear regression
Able to simplify nonlinear regression to linear one, slow, memory-consuming

Polynomial fit
Simple, able to tackle simple nonlinearities in data relations

Logistic regression
Simple, straightforward, quick

kNN
Easy to setup, slow, memory-consuming, scales poorly

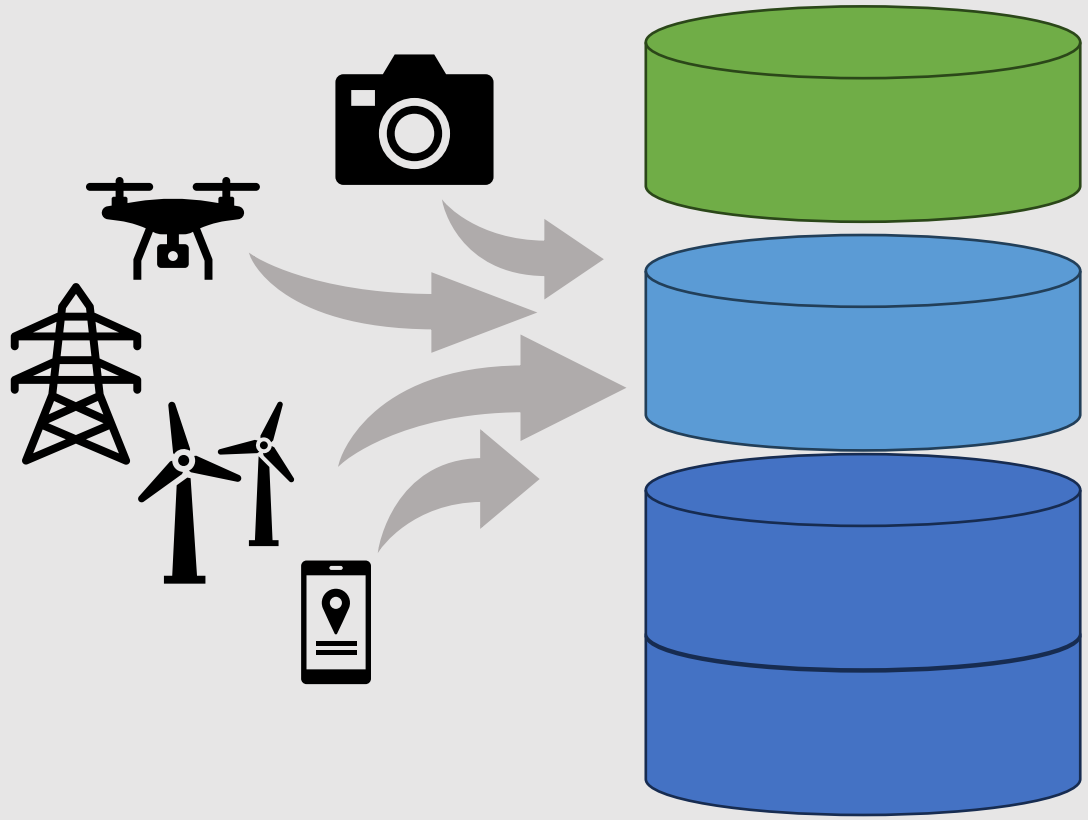
Decision tree
Easy to setup, fast, scales poorly (hard to train)

k-means clustering
Easy to setup, intuitive, requires determination of cluster number

Gaussian mixture model
Provides probability of sample presence, models the data

Artificial neural network
Adjustable for any problem (universal), complex configuration, slow training
(Its a family of different methods)

Data organization



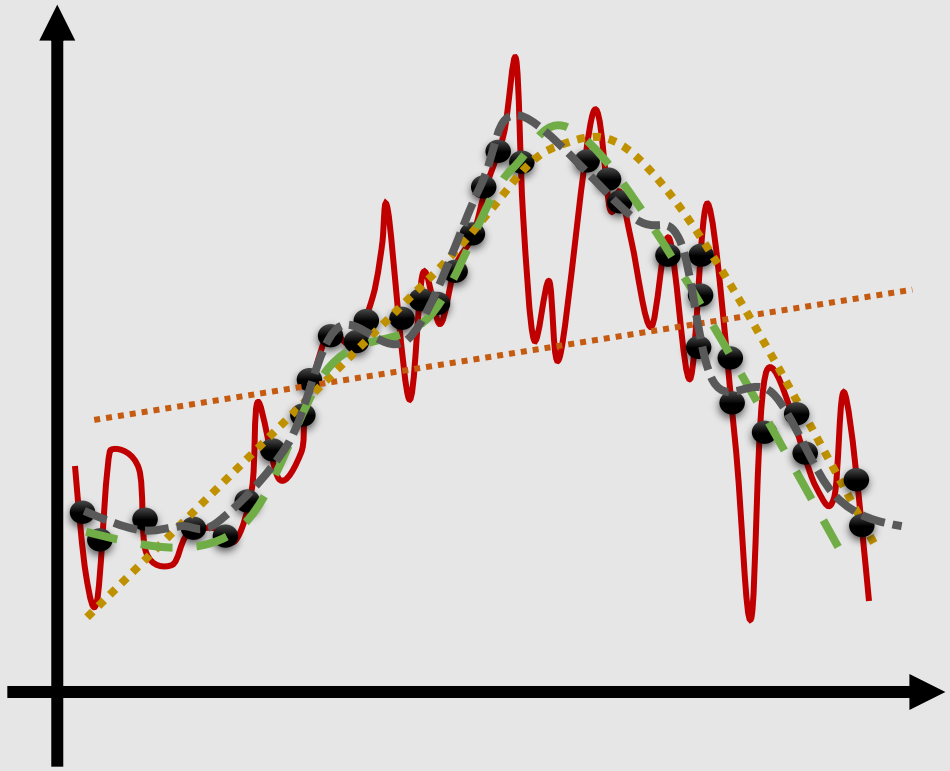
These subsets have to be separated!

Based on a **testing** subset of entire dataset we evaluate expected final result

Based on a **validation** subset of entire dataset we configure pattern recognition models and check if patterns are general

Based on a **training** subset of entire dataset we learn patterns present in data

How to set model complexity?



Model complexity (relative to the task)

Model overfits unless training is stopped early

Model performs well but contains unnecessary complexity

„Sweet spot" – model complexity allows to model all relevant data characteristics and generalizes well

Model is general enough (but not optimal) – General characteristics are modeled, nuances are lost

Model is too simple

How to set model complexity?

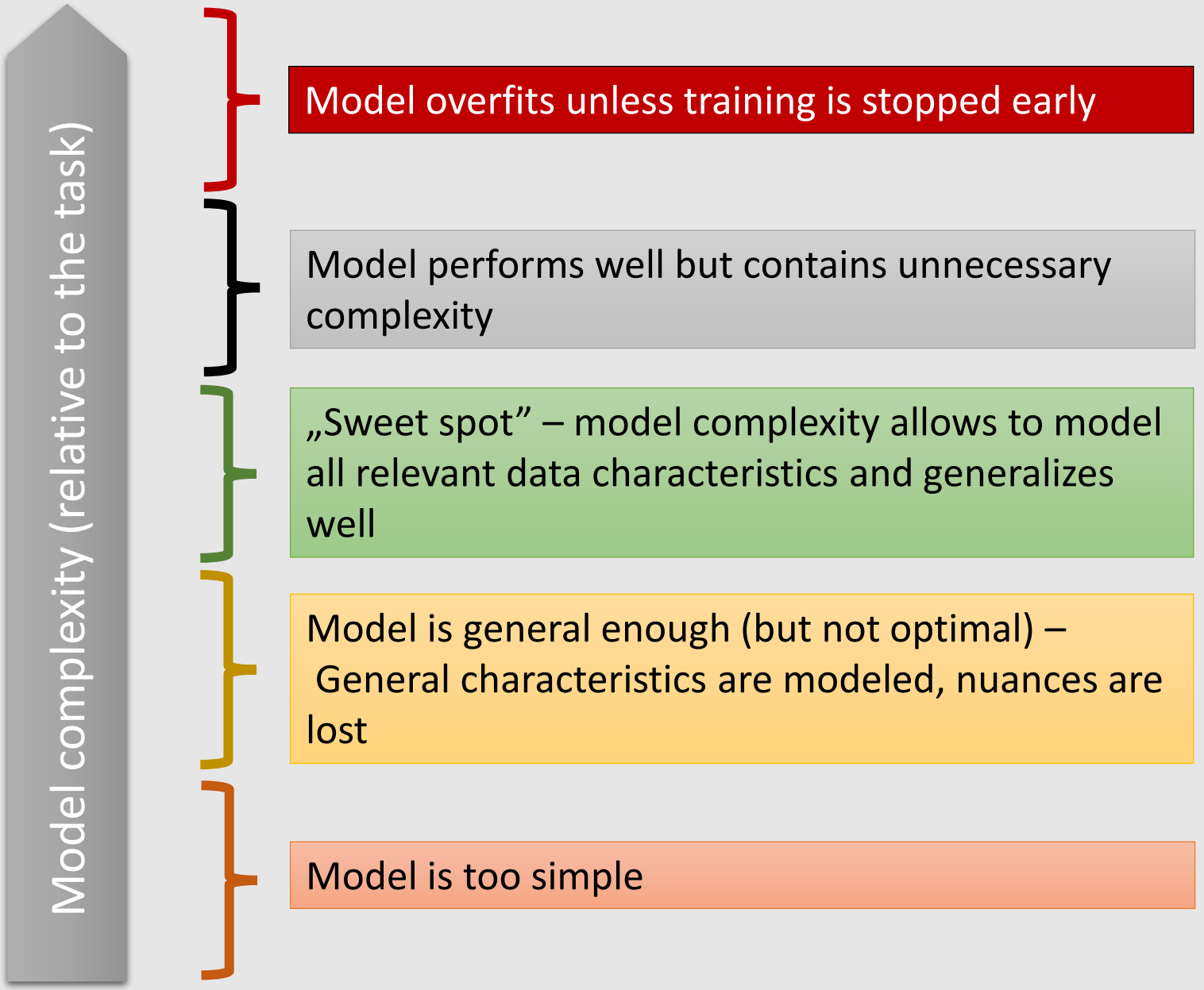
We can start from the top down, reducing model complexity until overfitting stops

OR

We can pick complexity based on data: 10 independent samples per adjustable parameter

OR

We can start from the bottom up, increasing model complexity until validation error stops decreasing



General remarks from before:

1. **Classifier should be chosen on a basis of our knowledge of a feature space**
   (How many dimensions? How many features? Are the samples clustered?)
   The „task type”, (e.g. do we classify cookies or vegetables) is much less relevant.

2. **Training, testing (and validation) datasets should be separate**
   Either we begin with random division of a data into training and testing subsets, or **(better!)** we gather new portion of data for testing purposes in another experiment.

3. **Number of degrees of freedom of a classifier (e.g. net weights) should depend on number of data samples**
   A good „rule of thumb” is that for each DOF of a classifier at least 10 data samples are required. If we can’t do that, we make sure that overfitting is accounted for!

4. **Feature quality > Classifier**
   Good features allow for easy classification even with a simple classifier. Advanced classifier won’t overcome weak features. It is better to spend more time on feature extraction than on classifier configuration.

Metaparameter configuration – in practice

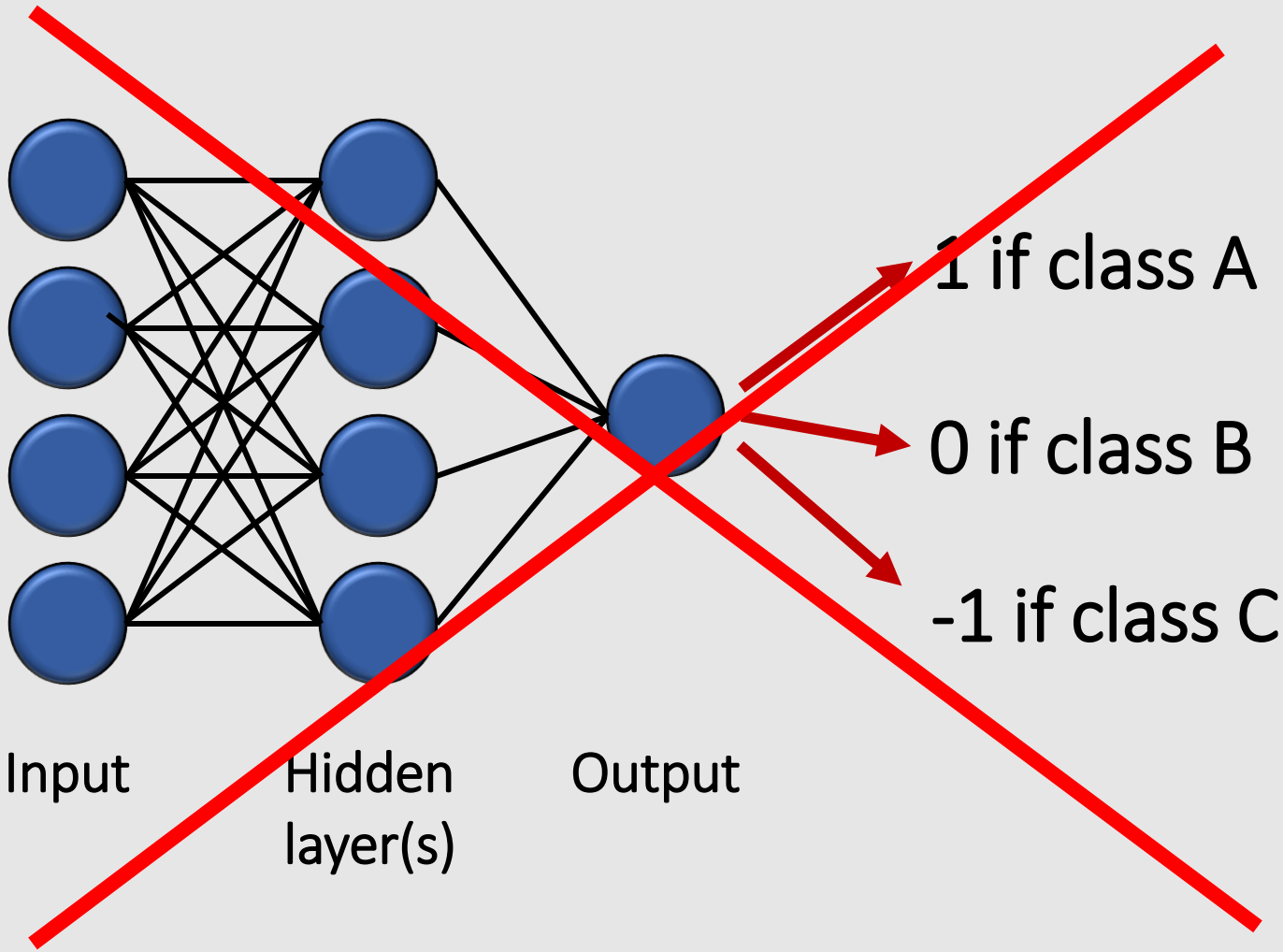
Approach 1: Make sure only that the model is complex enough to grasp data and go with „defaults" – or copy hyperparameter values from a similar setup
(Saves time and data)

Approach 2: Pick a reasonable starting point (default?) and then slightly change one hyperparameter at a time checking if it helps in a statistically significant way – until we run out of time (a 1+1 approach)
(We are allocating time and data resources as they are needed -
but we can end up in a local minimum or stagnate)

Approach 3: Decide which hyperparameters will influence the result the most – and do full optimization routine only for those (using grid search)
(*We risk doing lots of computations that will end in conclusion that it does not matter much…)*

Approach 4: Do a one-dimensional optimization for all hyperparameters separately
(Easy optimization but high risk of local minima here.
Hyperparameters often influence each other to a large extent)

Classifier outputs

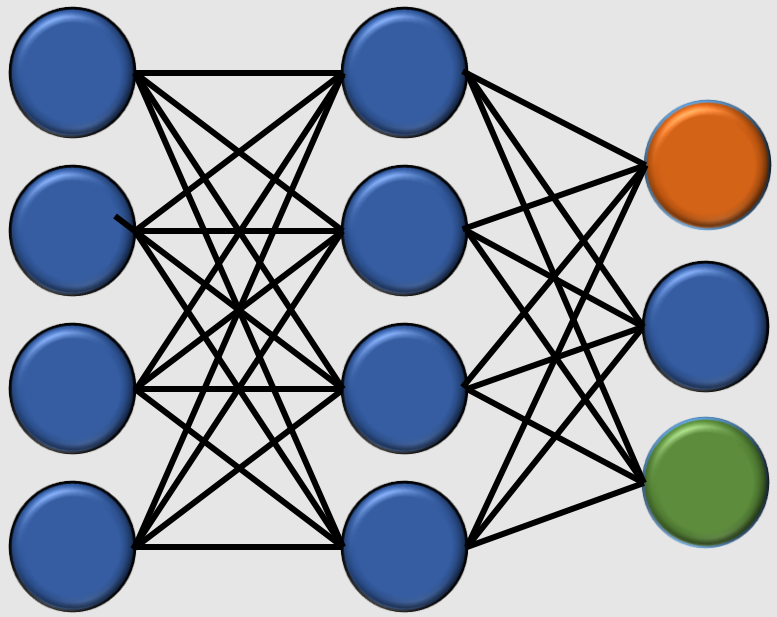


This tells the classifier, that class A is in relation to class C such that A is reversed C

Causes problems in training

Note 1: this example uses a neural network, but the reasoning is valid for all typical classifiers
Note 2: The scheme here is simplified. No biases and no activation functions.

Classifier outputs



Input

Hidden layer(s)

Output

1 if class A
0 otherwise

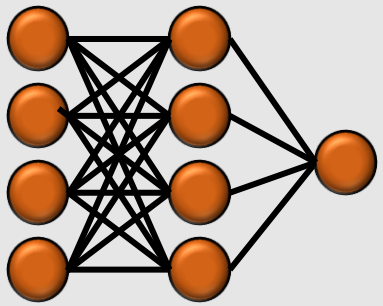
1 if class B
0 otherwise

1 if class C
0 otherwise

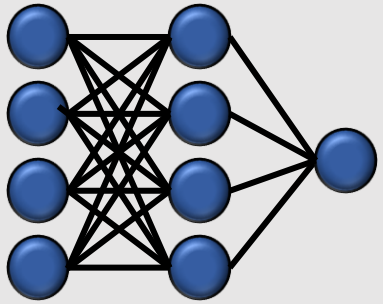
In multiclass classification we can either use multiple outputs from a classifier (each focusing on one class)

Note 1: this example uses a neural network, but the reasoning is valid for all typical classifiers
Note 2: The scheme here is simplified. No biases and no activation functions.

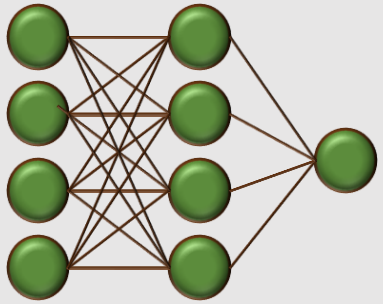
Classifier outputs



1 if class A
0 otherwise



1 if class B
0 otherwise



1 if class C
0 otherwise

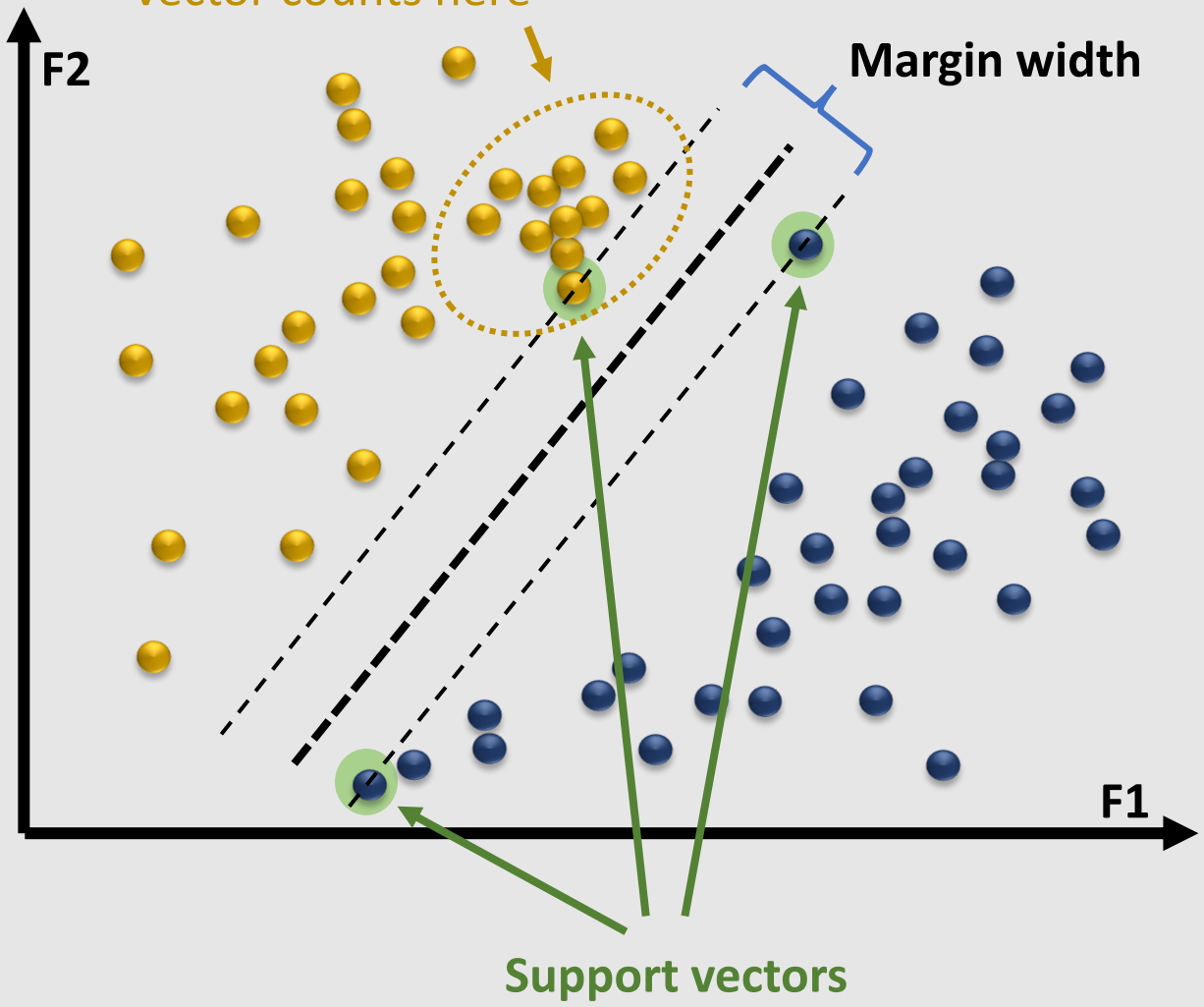
Or we can use multiple classifiers
(each focusing on one class)

Note 1: this example uses a neural network, but the reasoning is valid for all typical classifiers
Note 2: The scheme here is simplified. No biases and no activation functions.

What if we aim to fit the classifier PERFECTLY?

(Support Vector Machine)

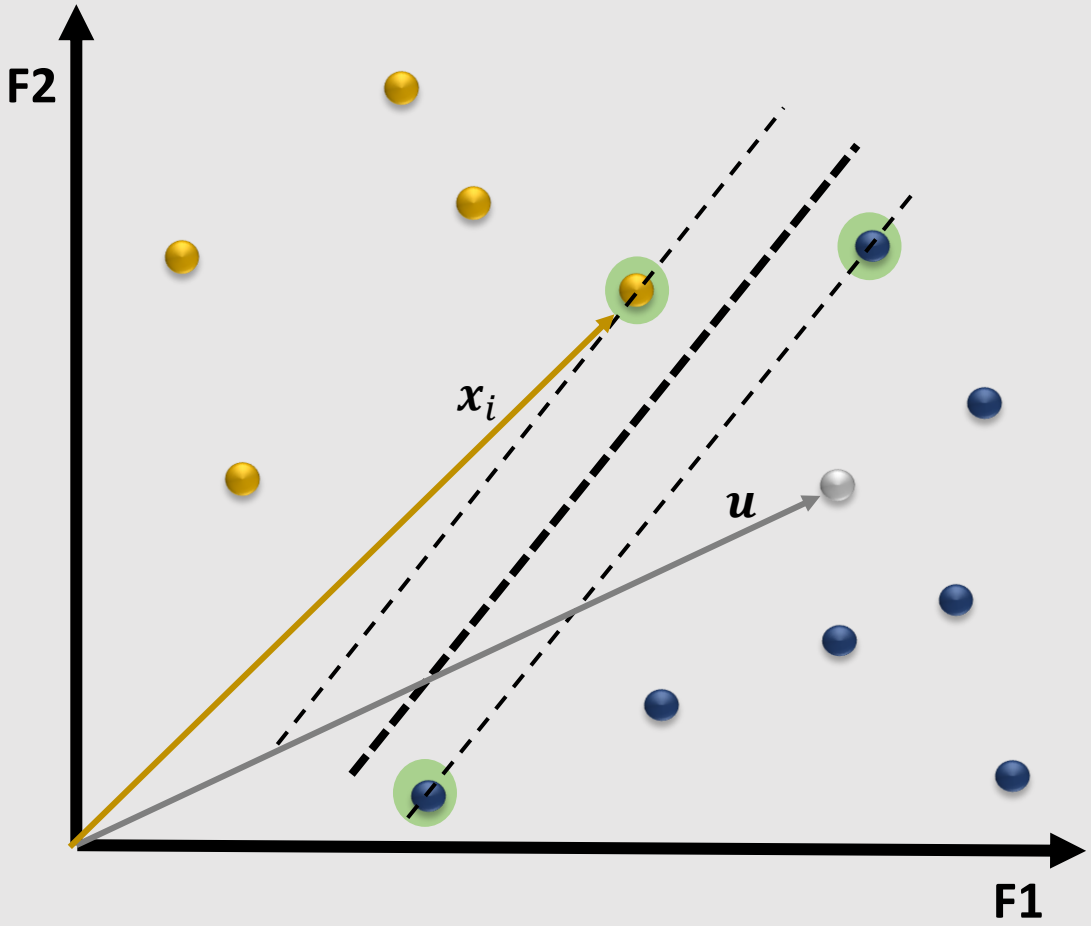
Support Vector Machine (SVM) principle



In regression SVM aims to make marigin as small as possible and fit all the data **inside** the margin.

In classification it does the opposite: it tries to make margin as wide as possible and keep all data **outside** of it

SVM revisited: How does it do that?



Let:

$\boldsymbol{\omega}$ - be a vector perpendicular to the margin
$\boldsymbol{x_i}$ - be the i-th sample
$\boldsymbol{u}$ - be an unknown sample
y_i - be a class label (equal to either 1 or -1)
α_i - be a Lagrangian multiplier denoting whether a point should be treated as a support vector

In a nutshell, If we want to classify a new sample, we are using the following equation:

$$\sum_i \alpha_i y_i \, \vec{x}_i \circ \vec{u} + b \geq 0 \Rightarrow THEN \oplus$$

Note, that to classify a new point we calculate its dot product with all the support vectors – that will come in handy in a second…

If you want to know what is the source for this equation, here is a brief explanation. You don't need to learn that for the test though!

Decision rule:

$$\vec{\omega}\circ\vec{u}+b\geq 0\Rightarrow THEN\oplus$$

For the training samples:

$$\begin{cases}\vec{\omega}\circ\vec{x}_{+}+b\geq 1\\ \vec{\omega}\circ\vec{x}_{-}+b\leq -1\end{cases}\Rightarrow y_i\left(\vec{\omega}\circ\vec{x}_i+b\right)-1\geq 0$$

For samples „on the margin" (support vectors):

1 $$y_i\left(\vec{\omega}\circ\vec{x}_i+b\right)-1=0$$

Margin width:

$$\left(\vec{x}_{+}-\vec{x}_{-}\right)\circ\frac{\vec{\omega}}{\|\omega\|}=\frac{2}{\|\omega\|}$$

We minimize $\frac{1}{2}\|\omega\|^2$

With respect to constraint 1

$$L=\frac{1}{2}\|\omega\|^2-\sum_i\alpha_i\left[y_i\left(\vec{\omega}\circ\vec{x}_i+b\right)-1\right]$$

$$\frac{\partial L}{\partial\omega}=\vec{\omega}-\sum_i\alpha_i y_i\vec{x}_i=0\Rightarrow\vec{\omega}=\sum_i\alpha_i y_i\vec{x}_i$$

$$\frac{\partial L}{\partial b}=-\sum_i\alpha_i y_i=0$$

After combination of the above equations, we have:

$$L=\sum_i\alpha_i-\frac{1}{2}\sum_i\sum_j\alpha_i\alpha_j y_i y_j\vec{x}_i\circ\vec{x}_j$$

New decision rule:

$$\sum_i\alpha_i y_i\vec{x}_i\circ\vec{u}+b\geq 0\Rightarrow THEN\oplus$$

SVM for nonlinearly separable data

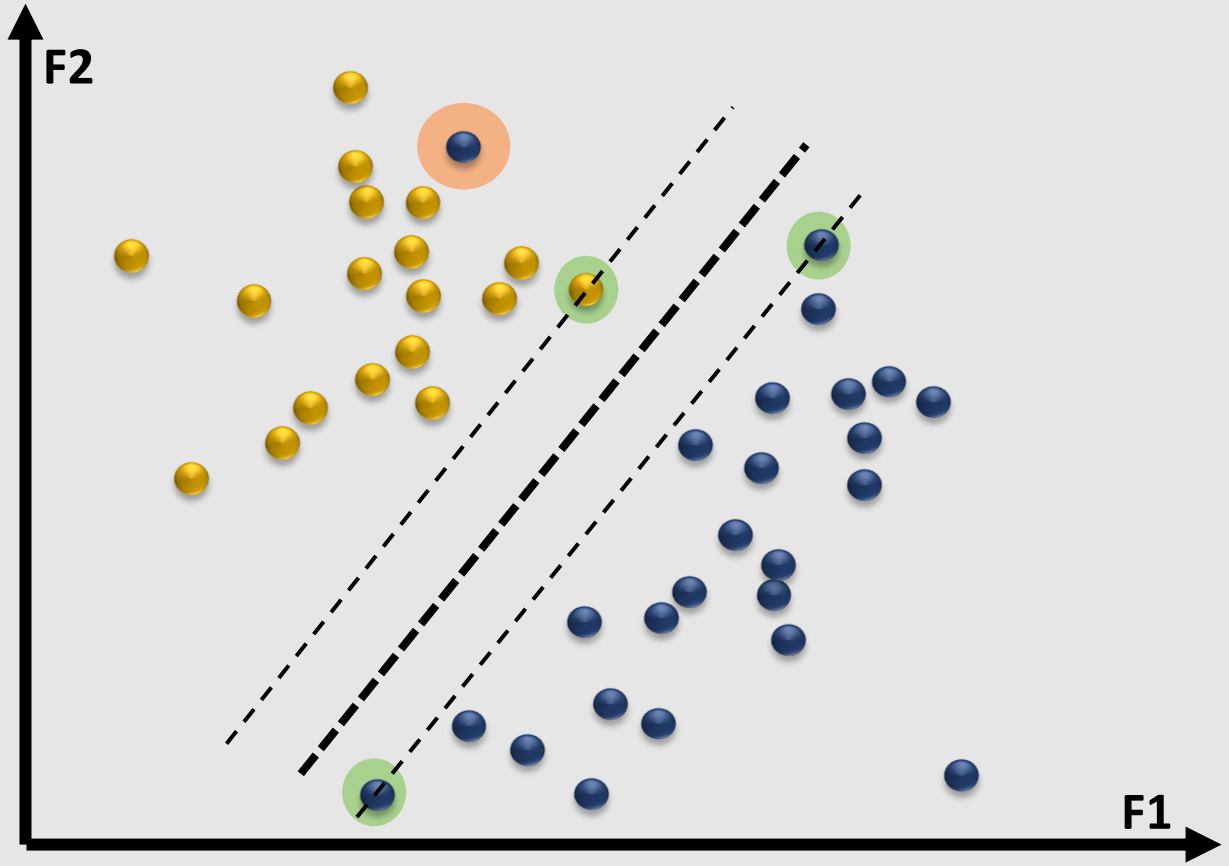
Nonlinearly separable data

We can allow the algorithm to ignore a few points

(Sometimes ignoring just a few outliers makes the problem linearly separable)

We classify data accepting small error rate

„Soft margin" idea



SVM for nonlinearly separable data

Nonlinearly separable data

We can allow the algorithm to ignore a few points

(Sometimes ignoring just a few outliers makes the problem linearly separable)

We classify data accepting small error rate

„Soft margin" idea

We minimize:

$$\left[\frac{1}{2}\sum_{i=1}^{n}\max\left(0,1-y_i(\vec{\omega}\cdot\vec{x}_i-b)\right)\right]+\lambda\|\vec{\omega}\|^2$$

Previous **constraint** now is just a weighted part of the objective function

λ allows to balance importance of the margin width and influence of outliers

SVM for nonlinearly separable data

Nonlinearly separable data

„kernel trick”

Kernel $K(x_1, x_2)$ Defines „similarity measure” between two vectors of features. Examples of kernels include:

We can map the data nonlinearly into space with higher number of dimensions by using **Kernel functions** instead of dot products

(There is a chance that data that were not separable in euclidean space will be separable in our new kernel-based space)

We classify data using linear SVM

- Polynomial functions:

$$K(x_1, x_2) = (1 + {x_1}^T x_2)^p$$

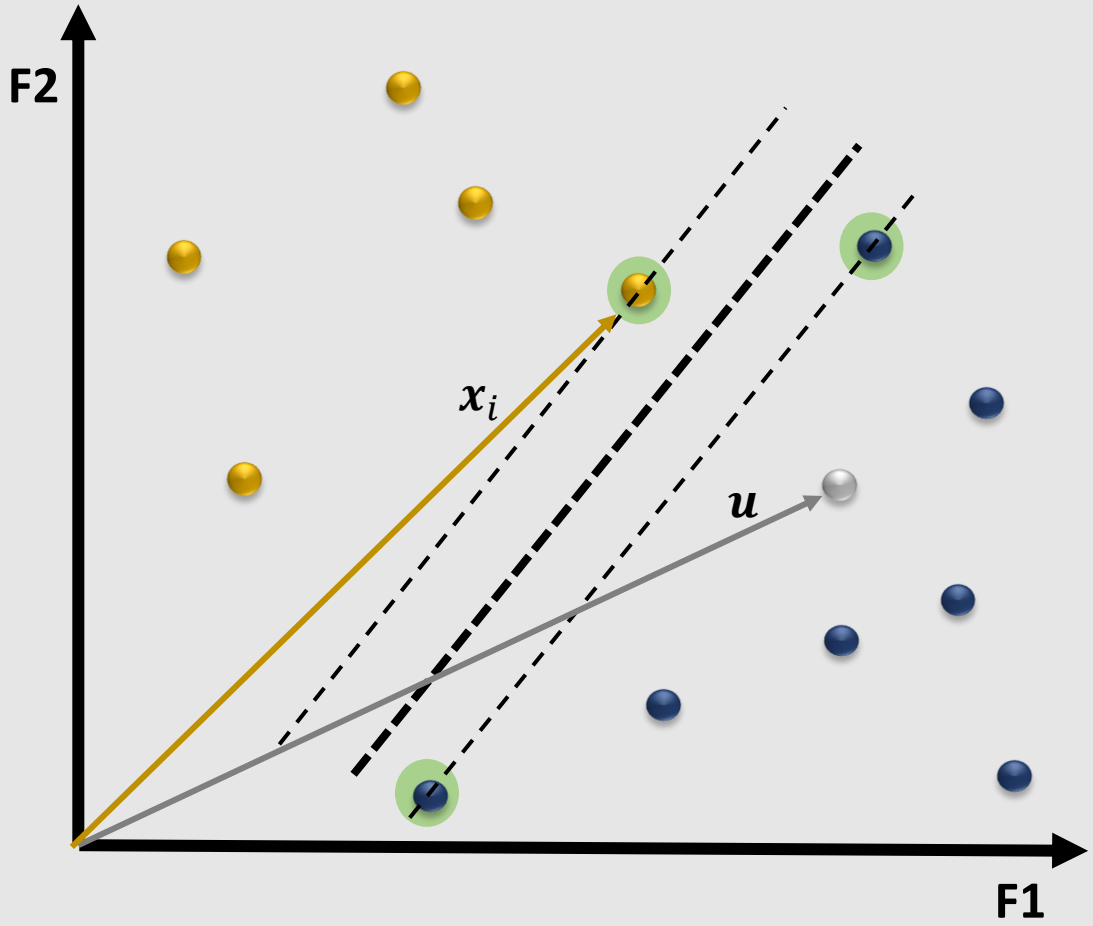
- Radial-basis functions (RBF)
(equivalent to weighted kNN model):

$$K(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$$

- Sigmoids:
(equivalent to single-layer MLP):

$$K(x_1, x_2) = \tanh(\beta_0 {x_1}^T x_2 + \beta_1)$$

SVM for nonlinearly separable data



Let:
- ω - be a vector perpendicular to the margin
- x_i - be the i-th sample
- u - be an unknown sample
- y_i - be a class label (equal to either 1 or -1)
- α_i - be a Lagrangian multiplier denoting whether a point should be treated as a support vector
- b – is a marigin width parameter

In a nutshell, If we want to classify a new sample, we are using the following equation:

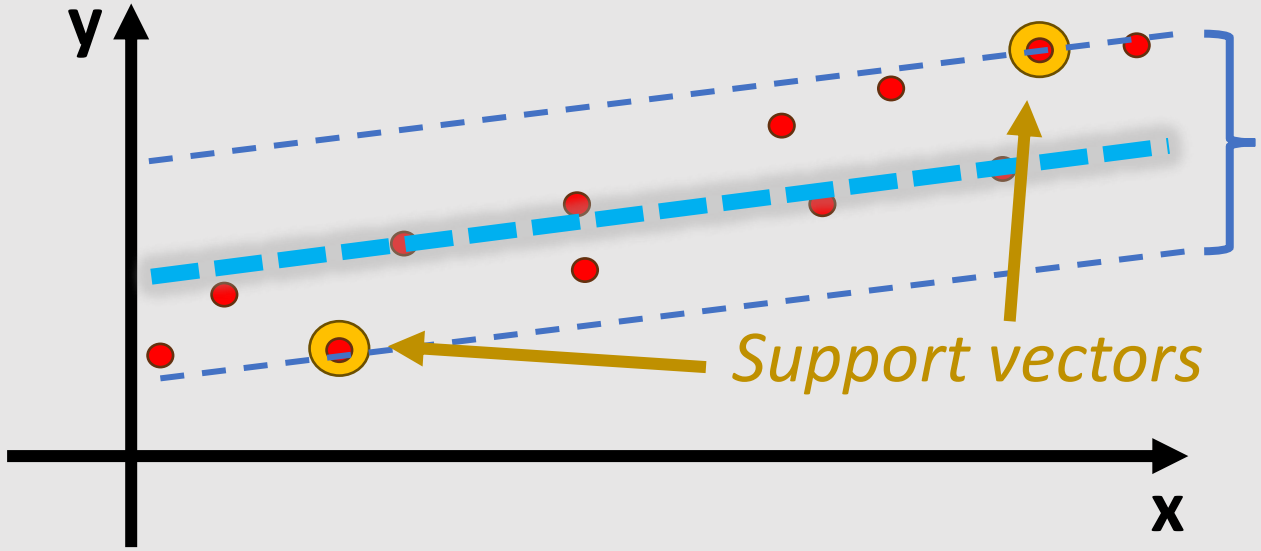
$$\sum_i \alpha_i y_i \, \vec{x}_i \circ \vec{u} + b \geq 0 \Rightarrow THEN \oplus$$

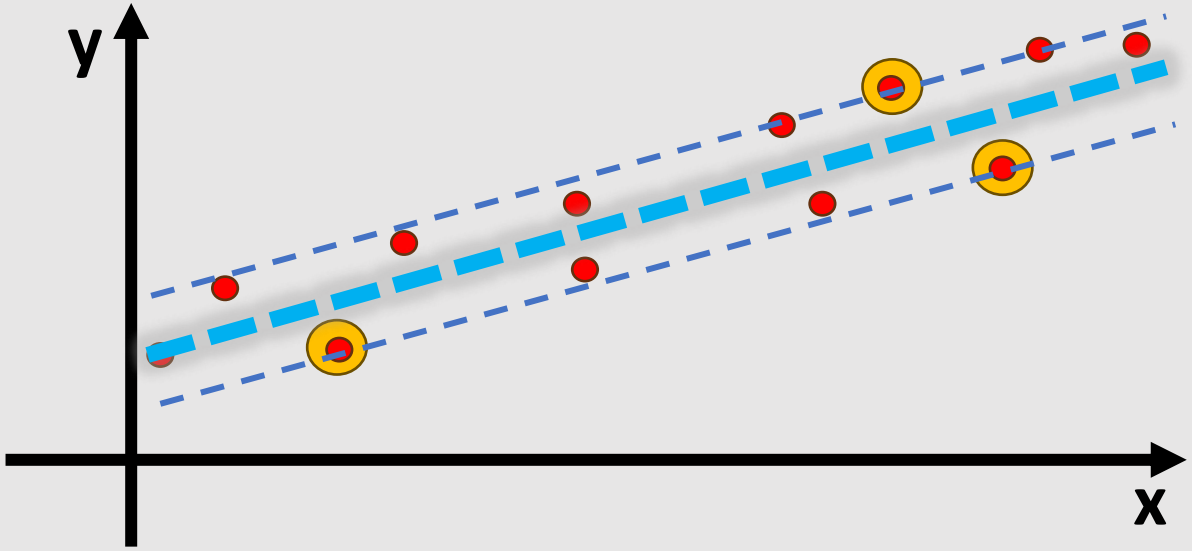
If we want to use nonlinear SVM, instead of dot product, we are using kernel function:

$$\sum_i \alpha_i y_i \, K(\vec{x}_i, \vec{u}) + b \geq 0 \Rightarrow THEN \oplus$$

A good visualization on how kernels affect data classification:
https://www.youtube.com/watch?v=Q7vT0--5VII

Support Vector Machine (SVM) for regression

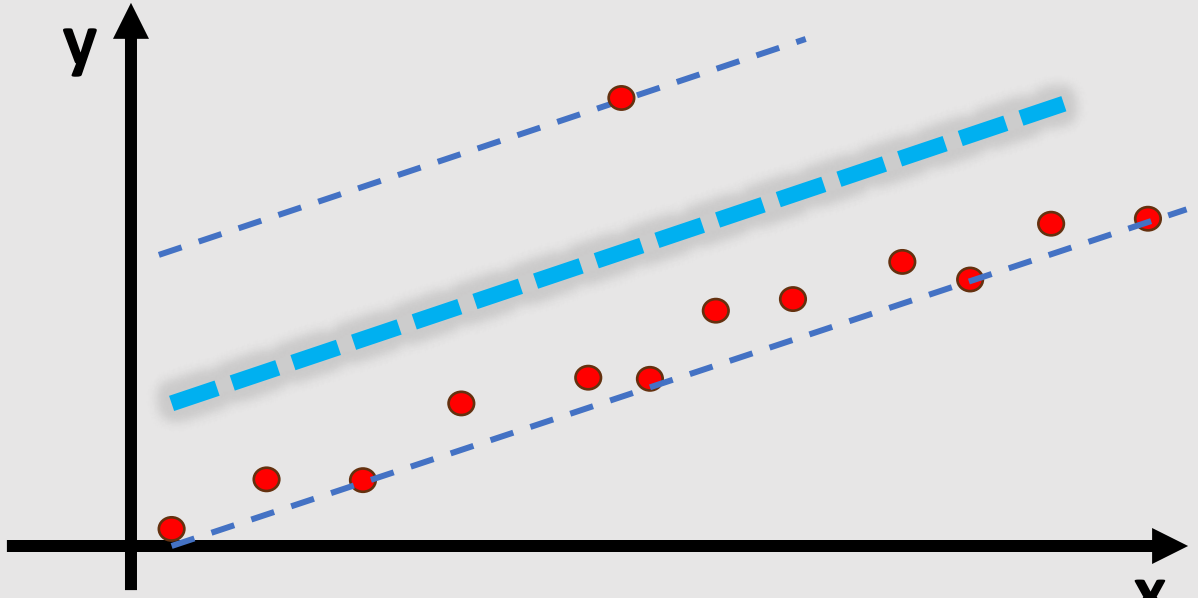


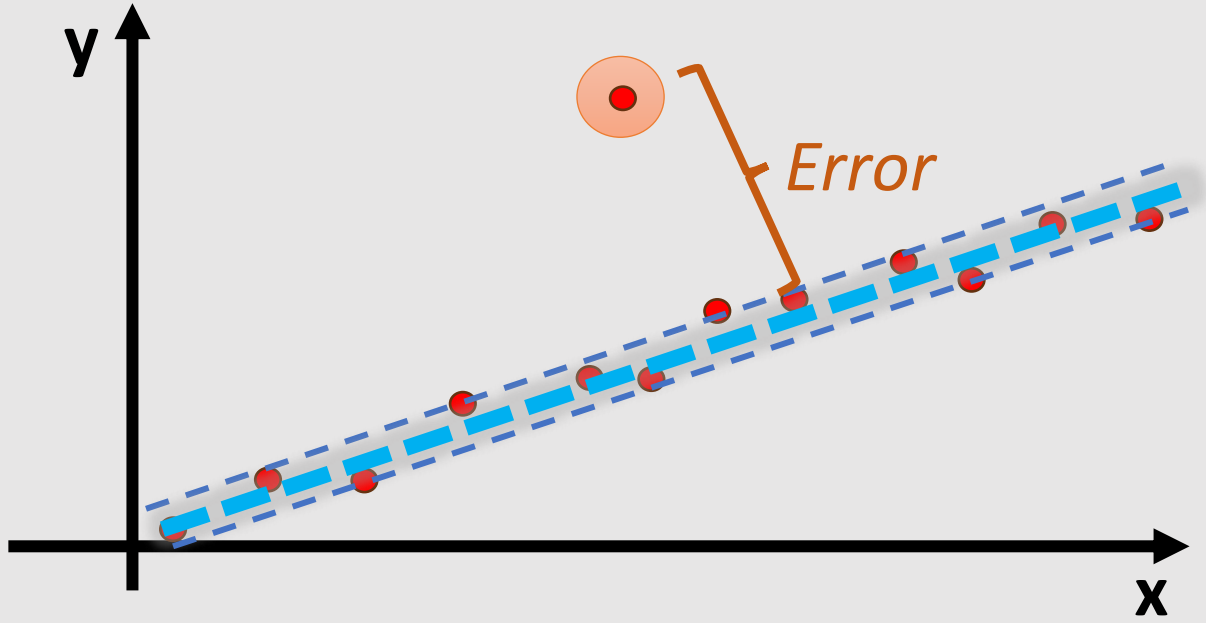


We fit the model so the width of the margin is minimal

(We are no longer interested in point-by-point errors)

Support Vector Machine (SVM) for regression

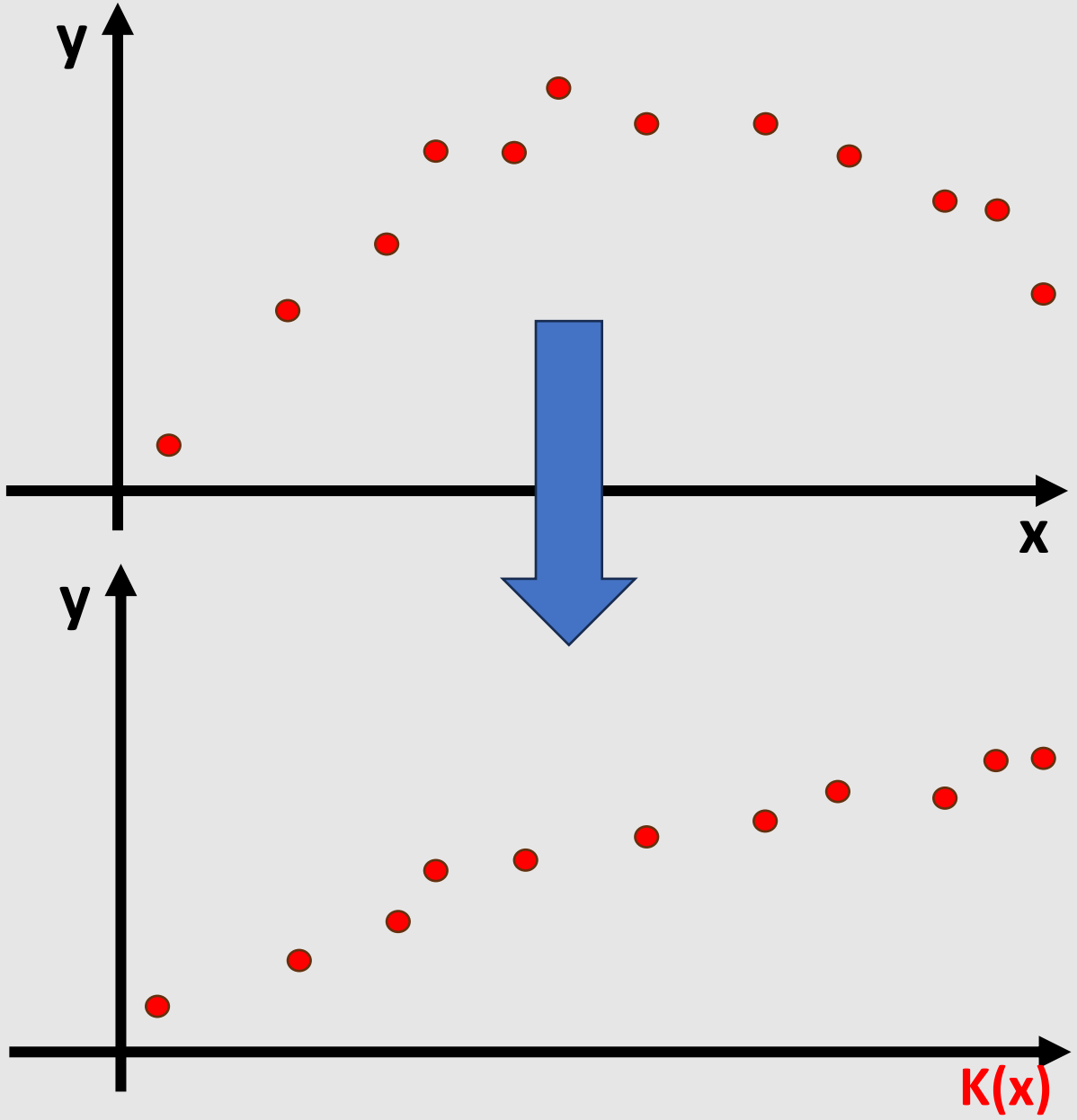




We fit the model so the width of the margin is minimal

We allow a few points to fall outside of margin (we add their errors to the cost function)

Support Vector Machine (SVM): kernel trick



$$f(\boldsymbol{w}, \boldsymbol{x})$$

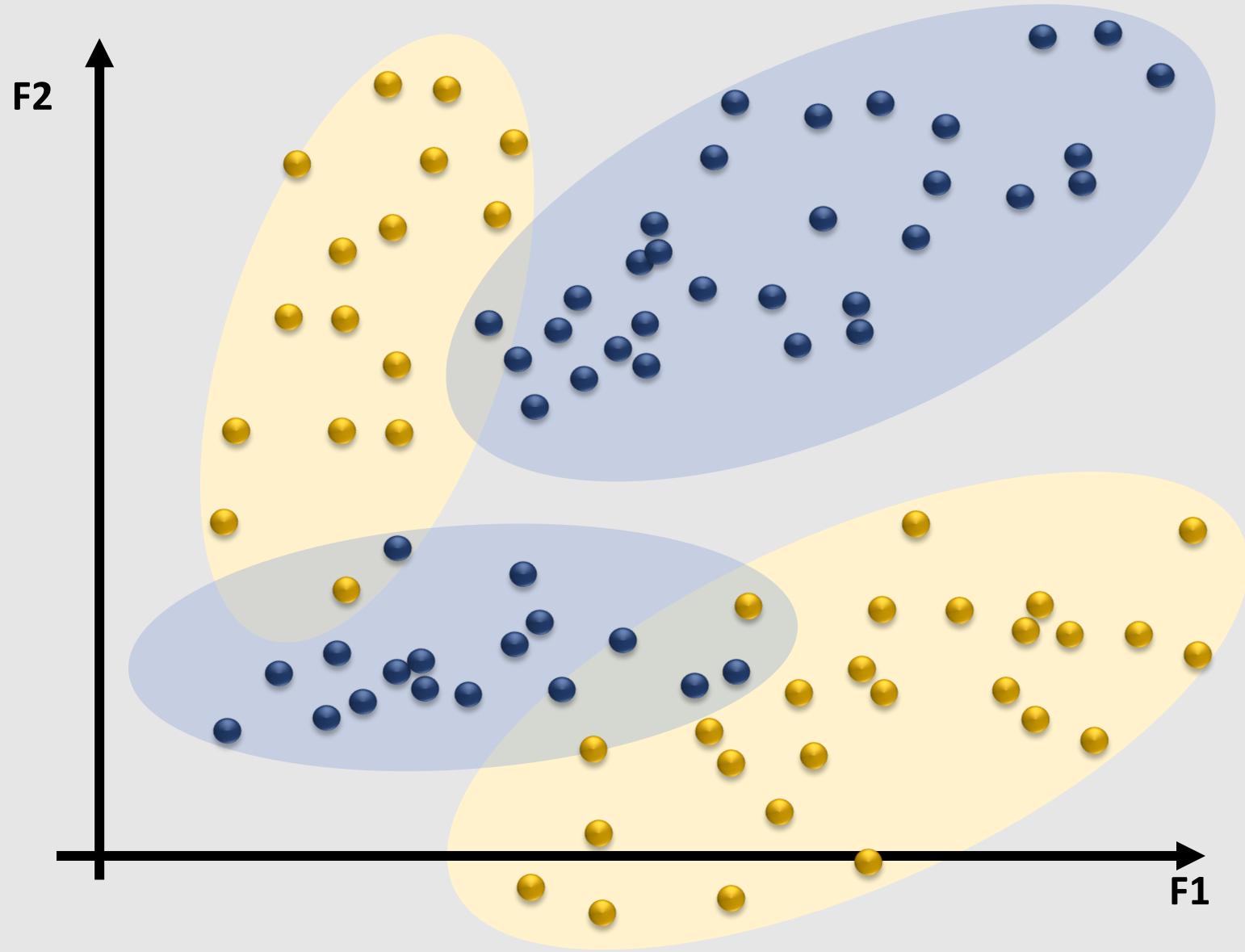
$$f(\boldsymbol{w}, K(\boldsymbol{x}))$$

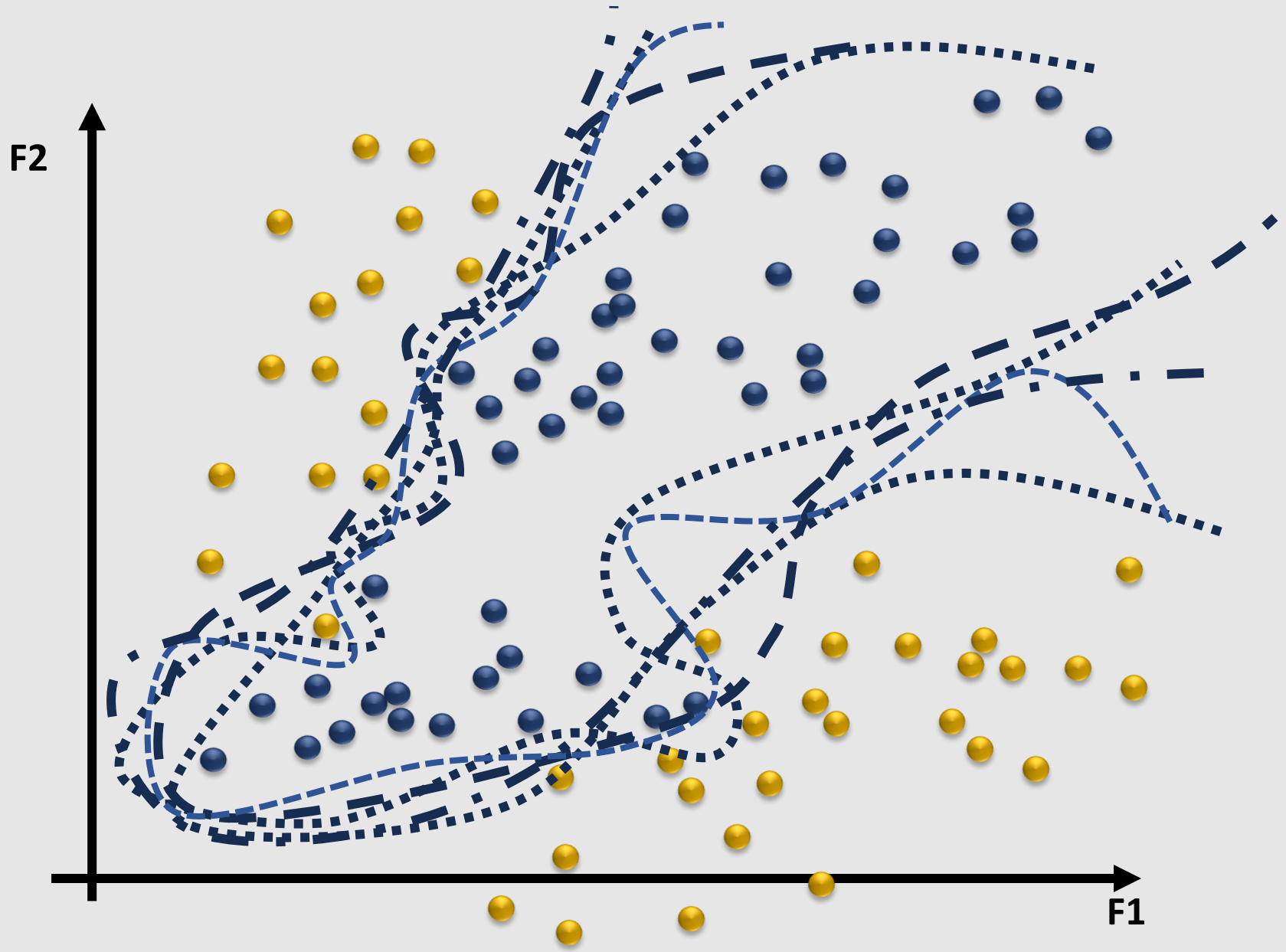
SVM materials

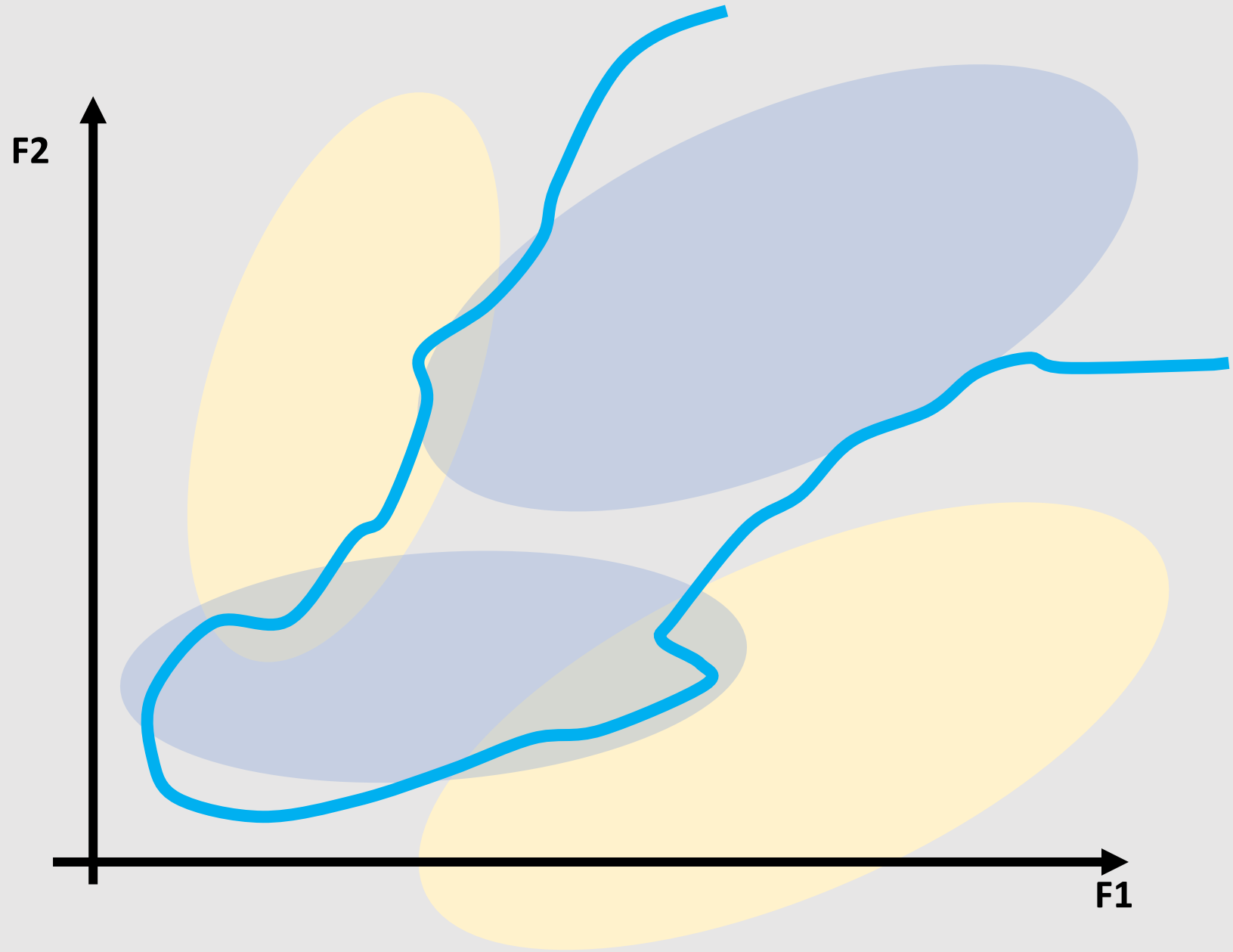
A good visualization on how kernels affect data classification:
https://www.youtube.com/watch?v=Q7vT0--5VII

Very good lecture introduction to SVMs can be found here:
https://www.youtube.com/watch?v=_PwhiWxHK8o

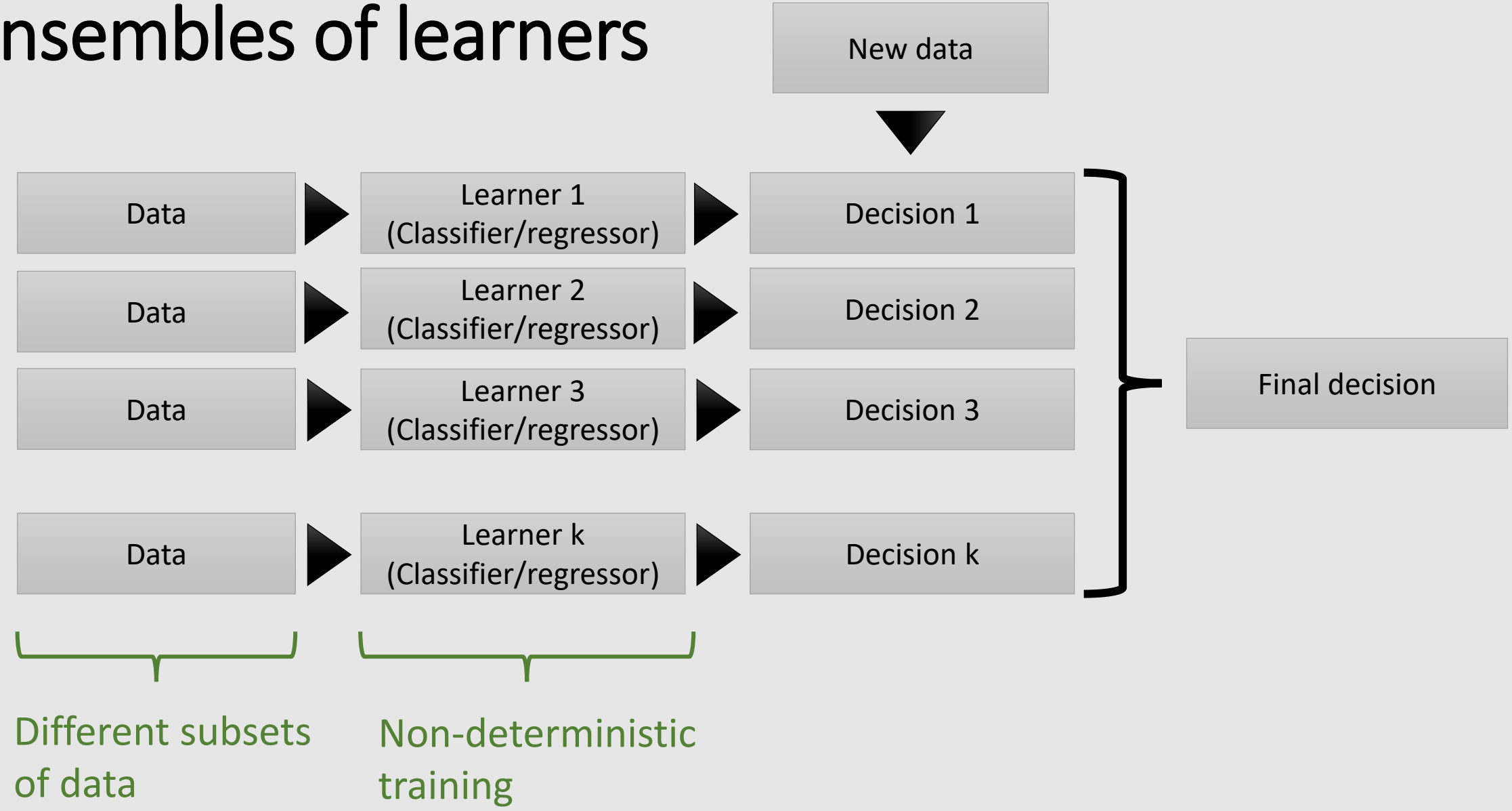
Paper on SVM implementation in scikit is here (beware of challenge!):
https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf


F2
F1


F2
F1

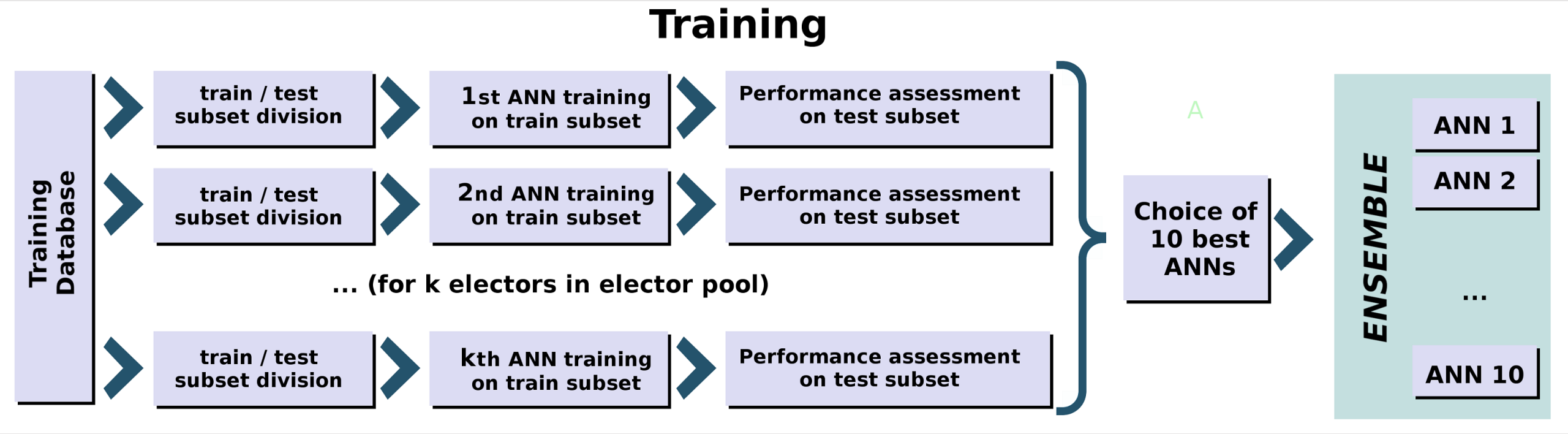

F2
F1

Ensembles of learners

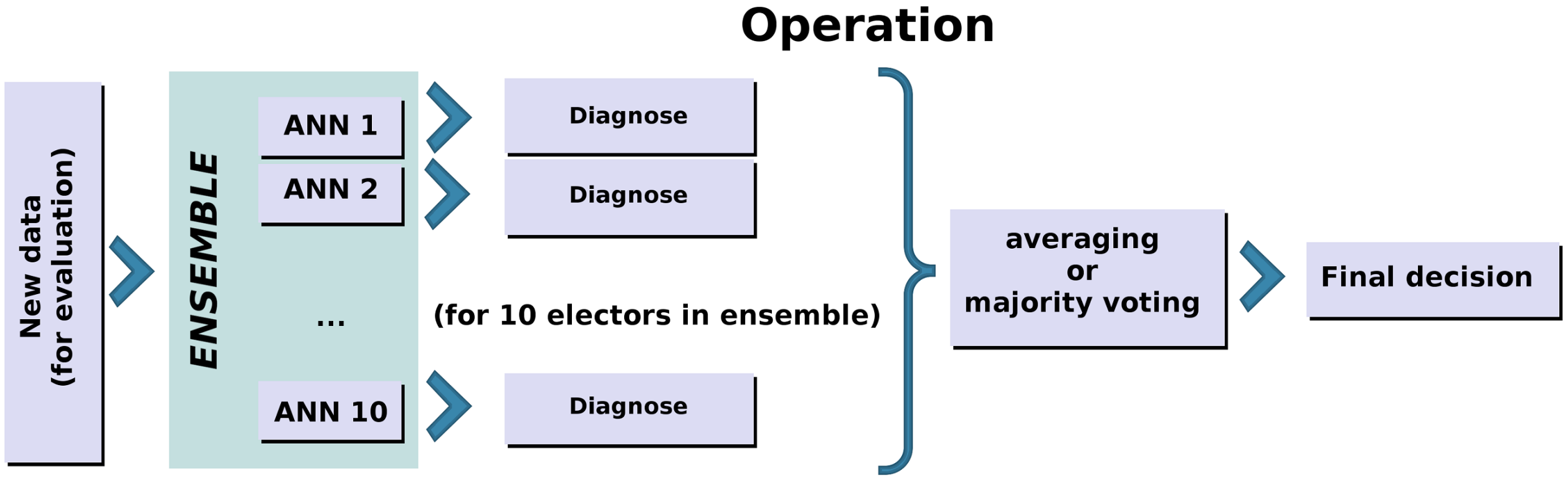


The more **diverse** and **accurate** the learners are, the better

Ensembles of learners: Example



Ensembles of learners: Example

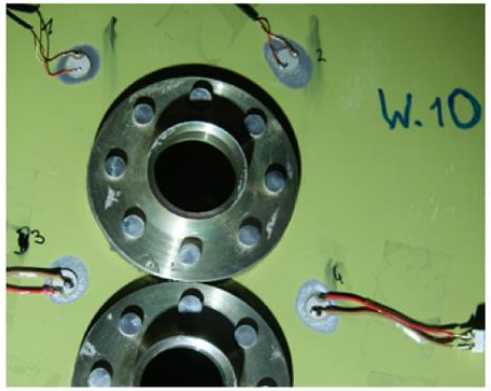


Ensembles of learners: Example

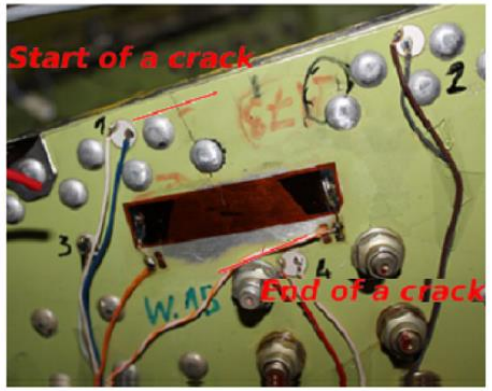
(a)

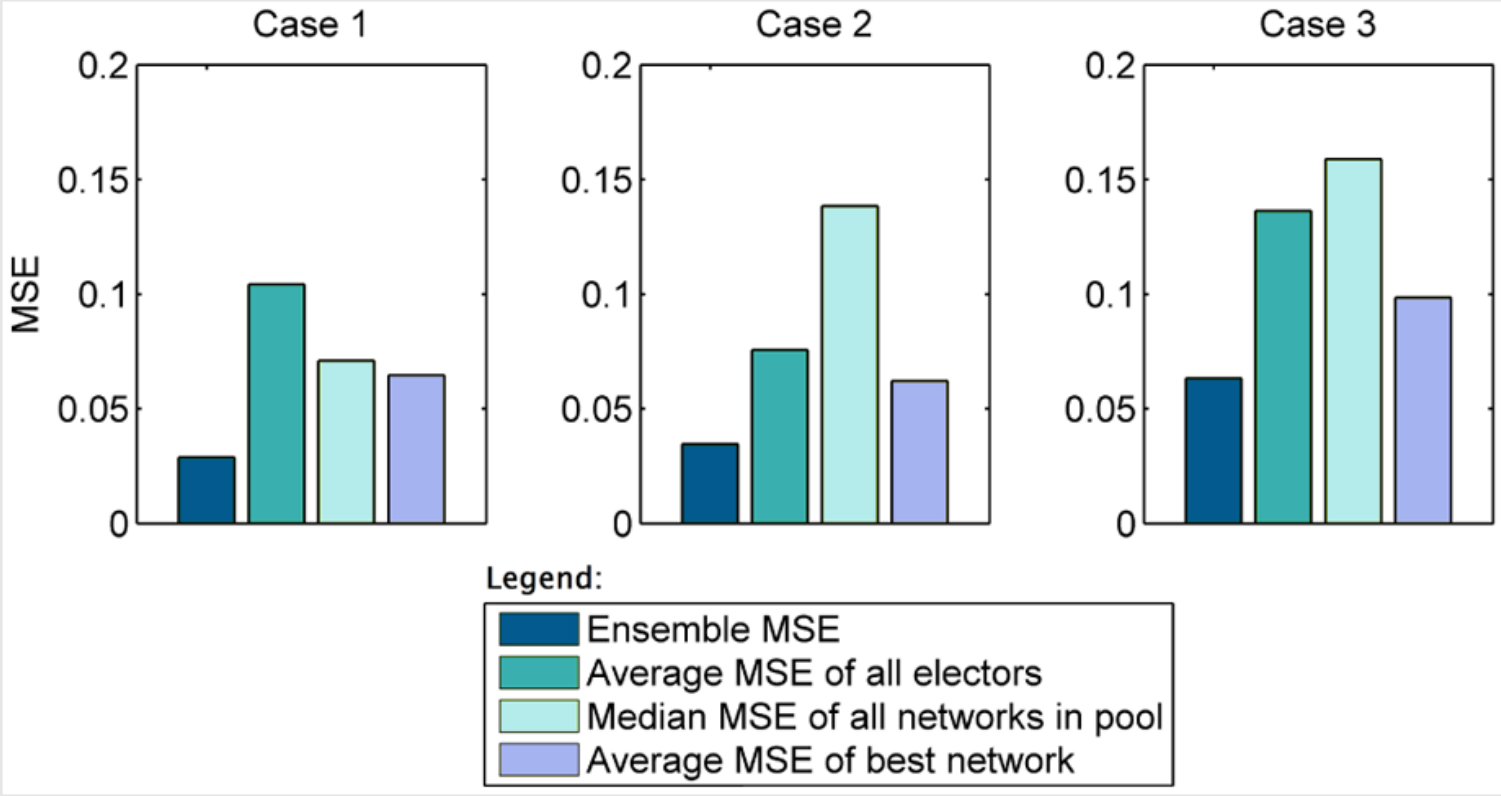


(b)



(c)





Random forest

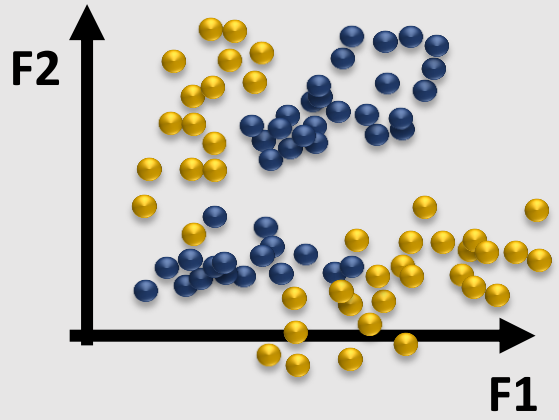


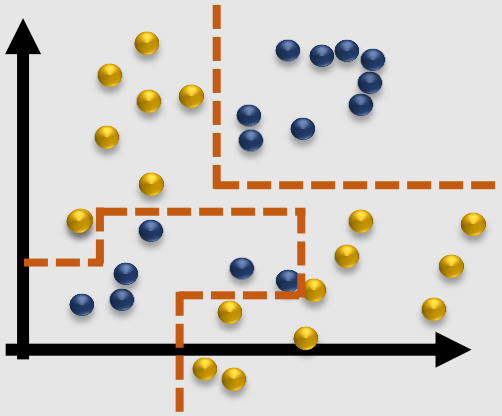
1. Do **Bagging** (Bootstrap Aggregating) by selecting repeatedly subsets of data using drawing with replacement

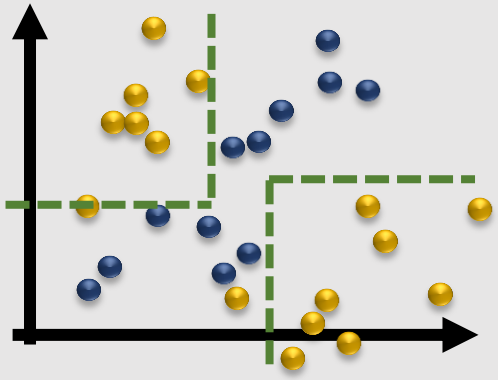
2. If dataset consists of many features, sample them also (let different subsets use also subsets of features!)

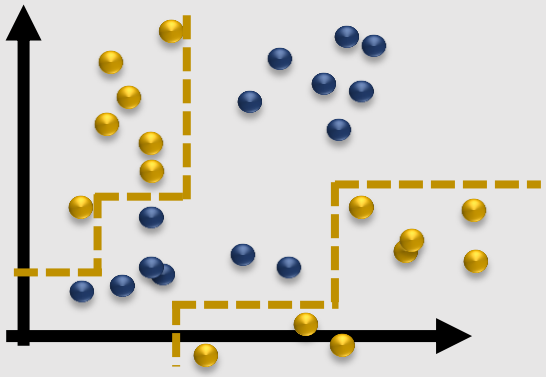
3. Train simple decision trees based on these subsets (each tree uses different subset)

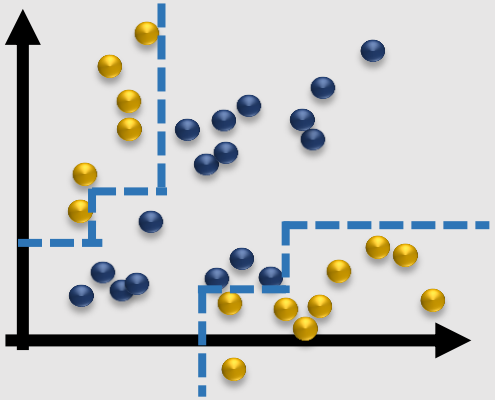
4. Average responses from many trees

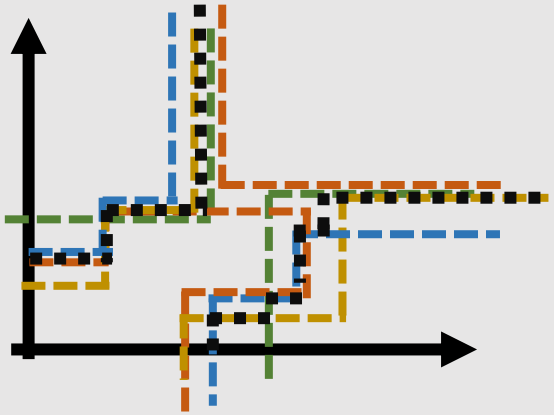

F2
F1

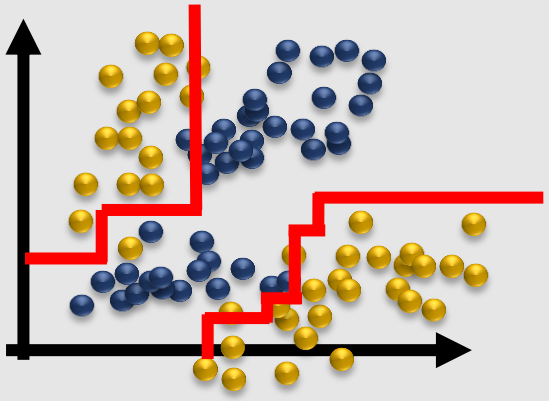










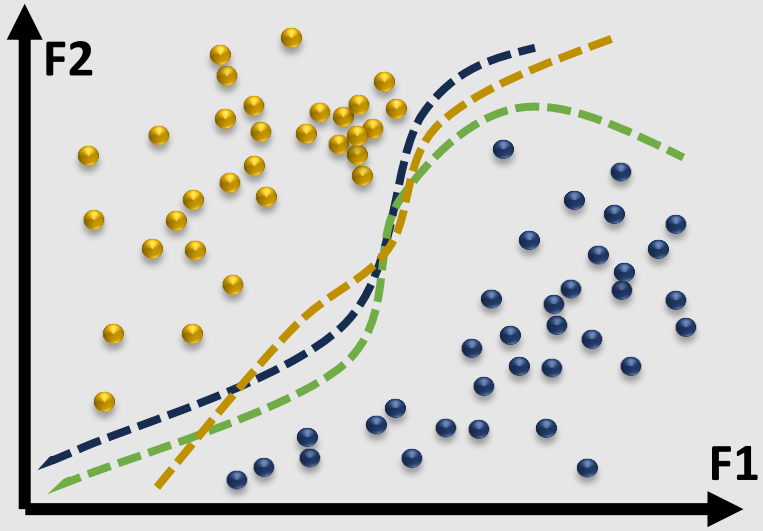


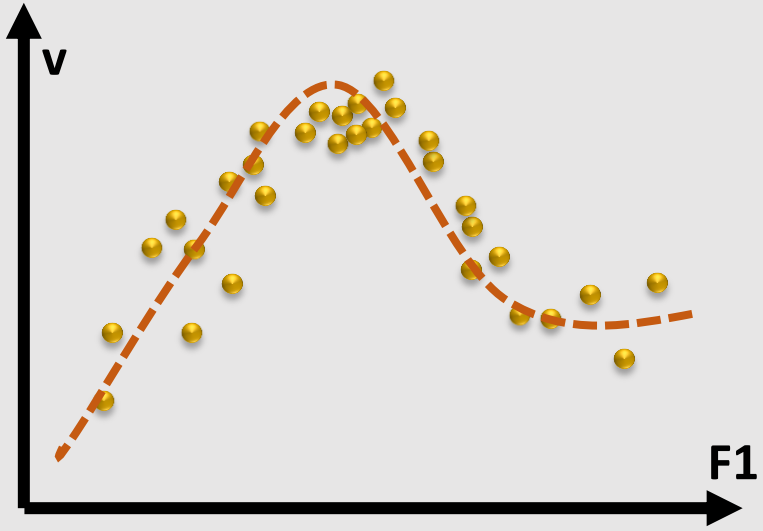
Ensembles of learners

In general: Ensembles provide **increased reliability** at the cost of

- **lack of direct control over classification proces**,
- **almost no structural optimization possibilities,** and
- **a lot of required memory and time**

So to sum up…





Layer 3 – Classifier/regressor structure introduces degrees of freedom and constraints

Ensembles prevent over-optimization (overfitting of metaparameters)

Layer 2 – training procedures introduce non-determinism

SVMs are deterministic

Ensembles provide robustness (average-out training differences)

Layer 1 – data space can be non-linear and is tested (probed) with uncertainty

SVMs provide „optimal separation”

Ensembles average-out noise in data

Things to remember:

1. Explain what classification and regression isa
2. Characterize basic clssification regression and clusterization algoritms (principle of operation and properties)
3. Explain approaches behind metaparameter optimization
4. Explain preferred output setup for multiclass classifiers
5. Explain principle of operation of SVMs for classification and regression (no equations needed)
6. Explain „soft margin” idea for SVMs
7. Explain kernel trick – with example for regression and classification
8. Explain what Ensembles are and what benefits they provide.
9. Show illustrative example of classification with random forest