MLRF Lecture 06

J. Chazalon, LRDE/EPITA, 2021

Some classifiers – part 2

Lecture 06 part 02

How to build non-linear classifiers?

2 solutions:

1. **Preprocess** the data – *seen last time*
   Ex.: explicit embedding, kernel trick…
   ⇒ change the input to make it linearly separable
2. **Combine** multiple **linear classifiers** into nonlinear classifier – *current topic*
   Ex.: boosting, neural networks…
   ⇒ split the input space into linear subspaces

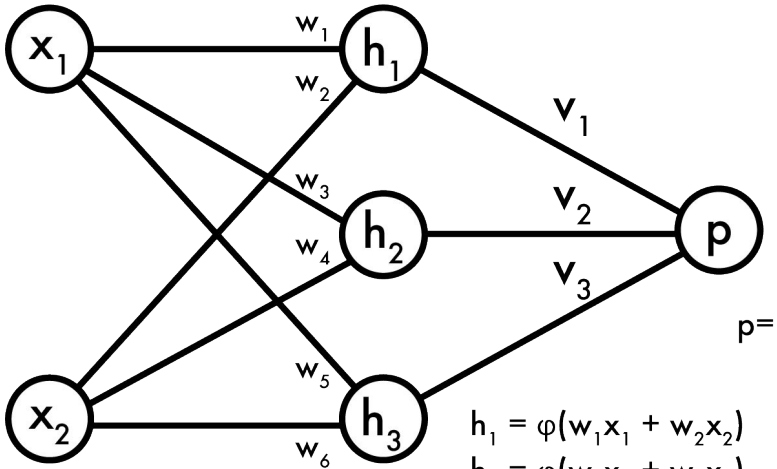
Non-linear classification using combinations of linear classifiers

Multi-layer Perceptron

Combine features linearly, apply a linear activation function φ, repeat.

Perceptron
very simple linear classifier

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$



from JP Reddie

Universal approximation theorem

What if φ not linear?

Universal approximation theorem (Cybenko 89, Hornik 91)

φ: any nonconstant, bounded, monotonically increasing function

I_m: m-dimensional unit hypercube (interval [0-1] in m dimensions)

Then 1-layer neural network with φ as activation

can model any continuous function f: $I_m \rightarrow R$

(no bound on size of hidden layer)

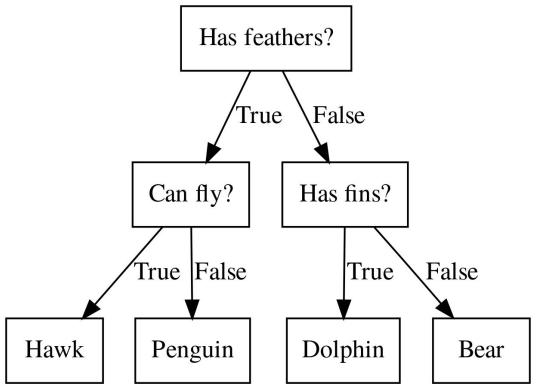
By extension, works on f: bounded $R^m \rightarrow R$

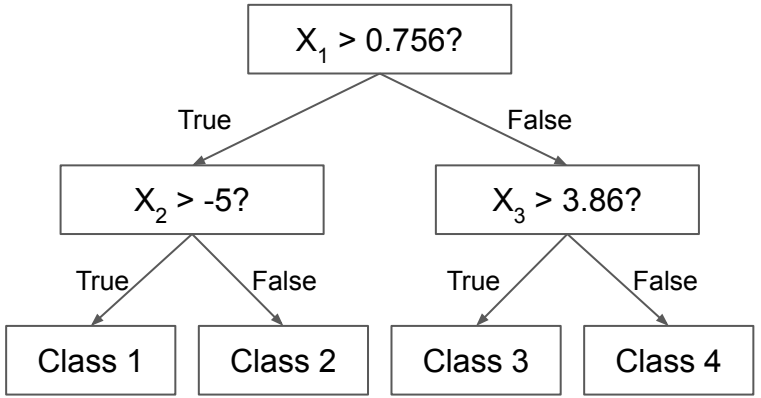
What can we learn? What can't we? UAT just says it's possible to model, not how.

From JP Reddie
https://en.wikipedia.org/wiki/Universal_approximation_theorem

Decision trees

Works on categorical (like “red”, “black”) and numerical (both discrete and continuous) random variables.





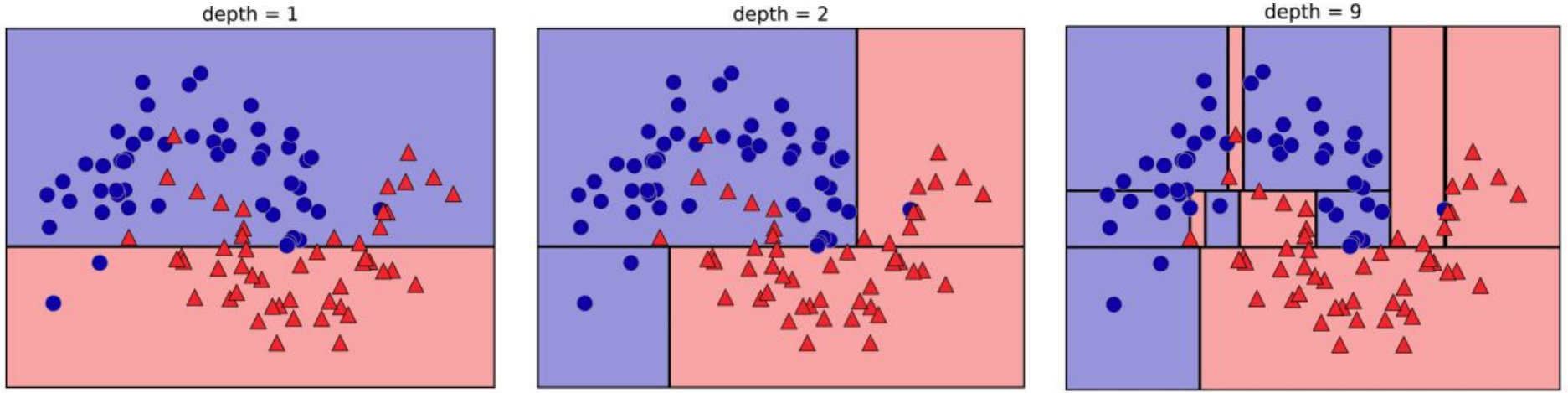
Train by optimizing classification “purity” at each decision (threshold on a particular dimension in numerical case).

Decision trees

Very fast training and testing. Non parametric.

No need to preprocess the features.

BUT: Very prone to **overfitting** without strong limits on depth.

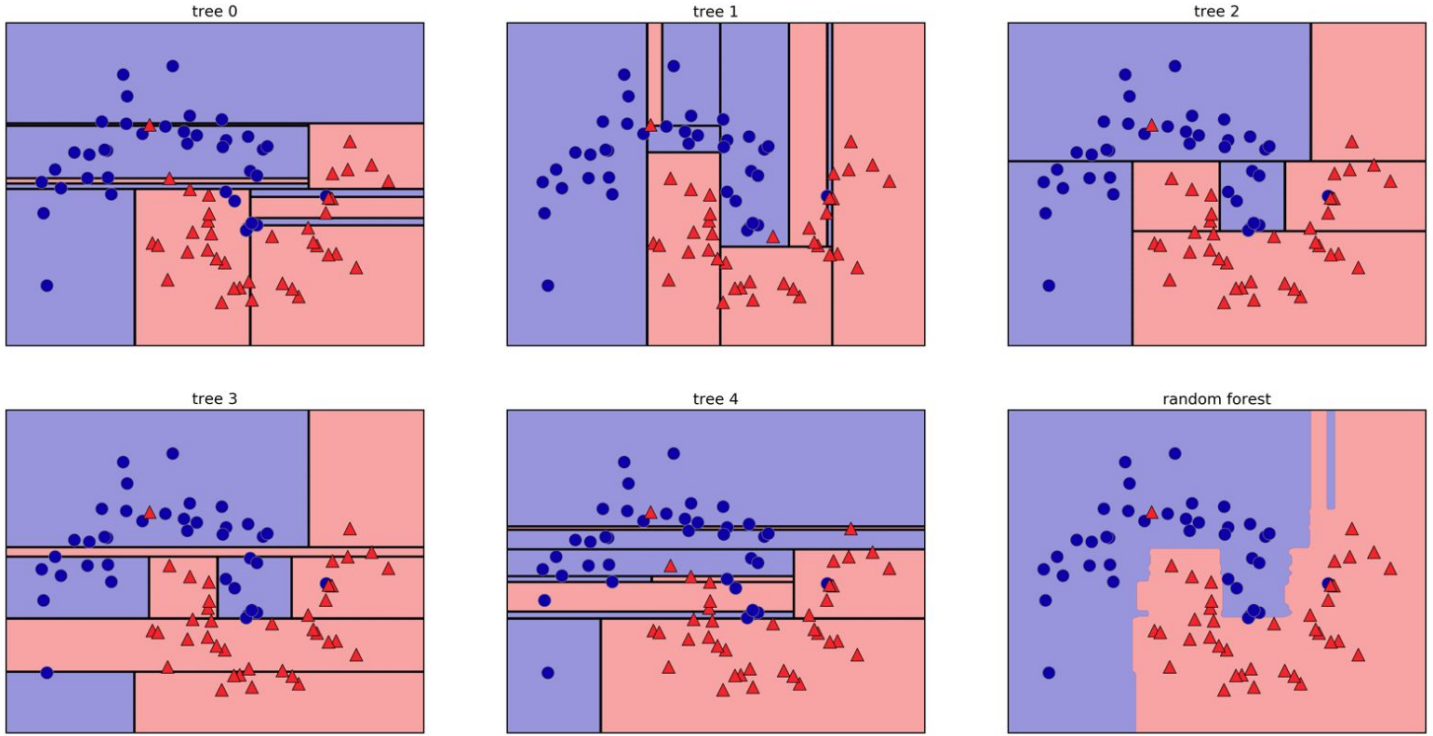


from A. Müller

Random Forests

[Breiman 2001]

Average the decision of multiple decision trees.



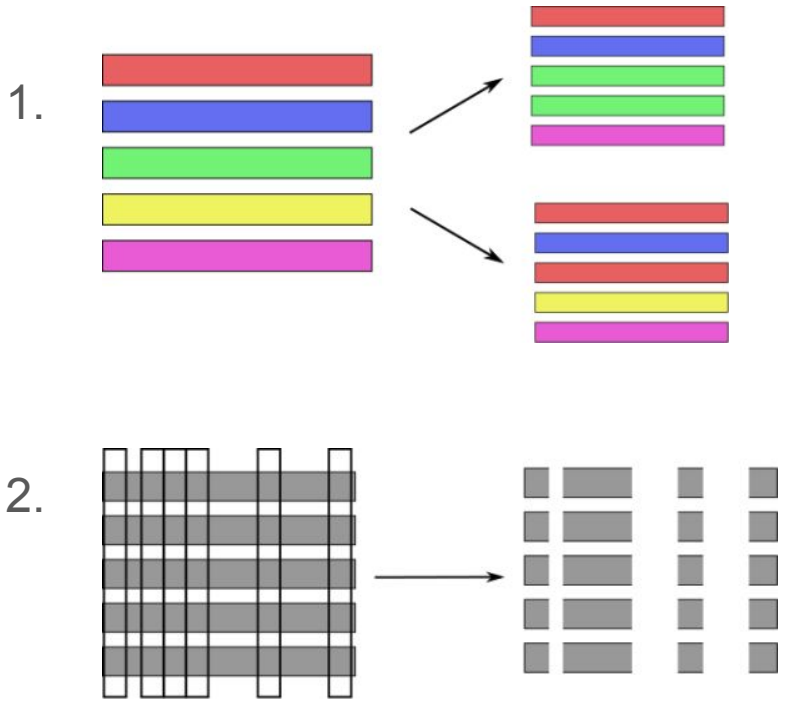
from A. Müller

Random Forests

Randomize in two ways:

1. For each tree,
   pick a **bootstrap** sample of data
2. For each split,
   pick random sample of features

More trees are always better.



from A. Müller

Ensemble methods

“Bagging” *or* “bootstrap aggregating”

[Breiman 96]

Underlying idea: part of the variance is due to the specific choice of the training data set.

1. Let us create many similar training data sets using **bootstrap**.
2. For each of them, train a new classifier.
3. The final function will be the **average** of each function outputs.

⇒ If generalization error is decomposed into bias and variance terms then bagging reduces variance. (average of large number of random errors ≈ 0)

Random forest = a way of bagging trees.

from L. Miclet

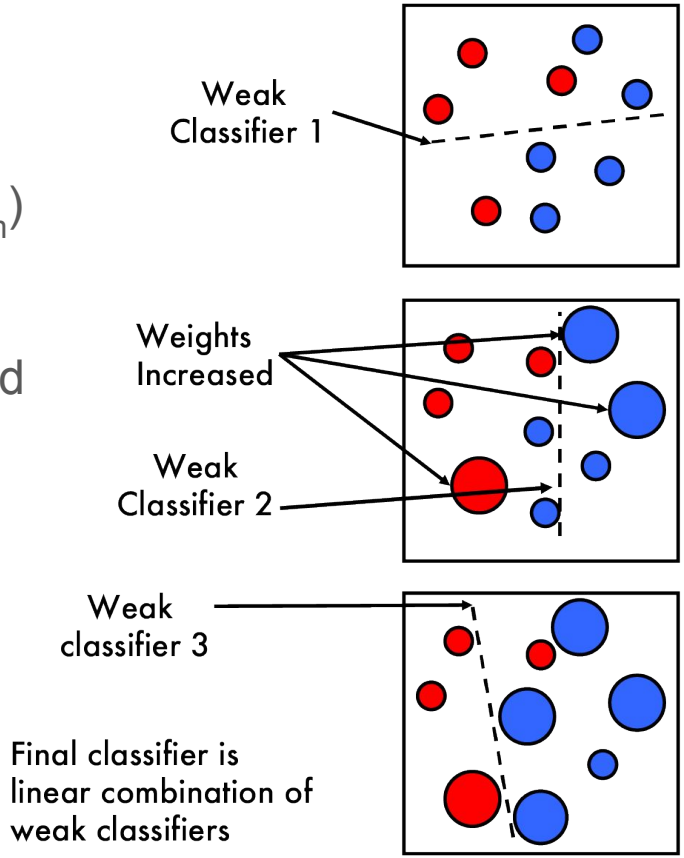
“Boosting”, AdaBoost variant

[Freund & Shapire 95]

Combinaison of weak classifiers $\sum_m \alpha_m G_m(x)$

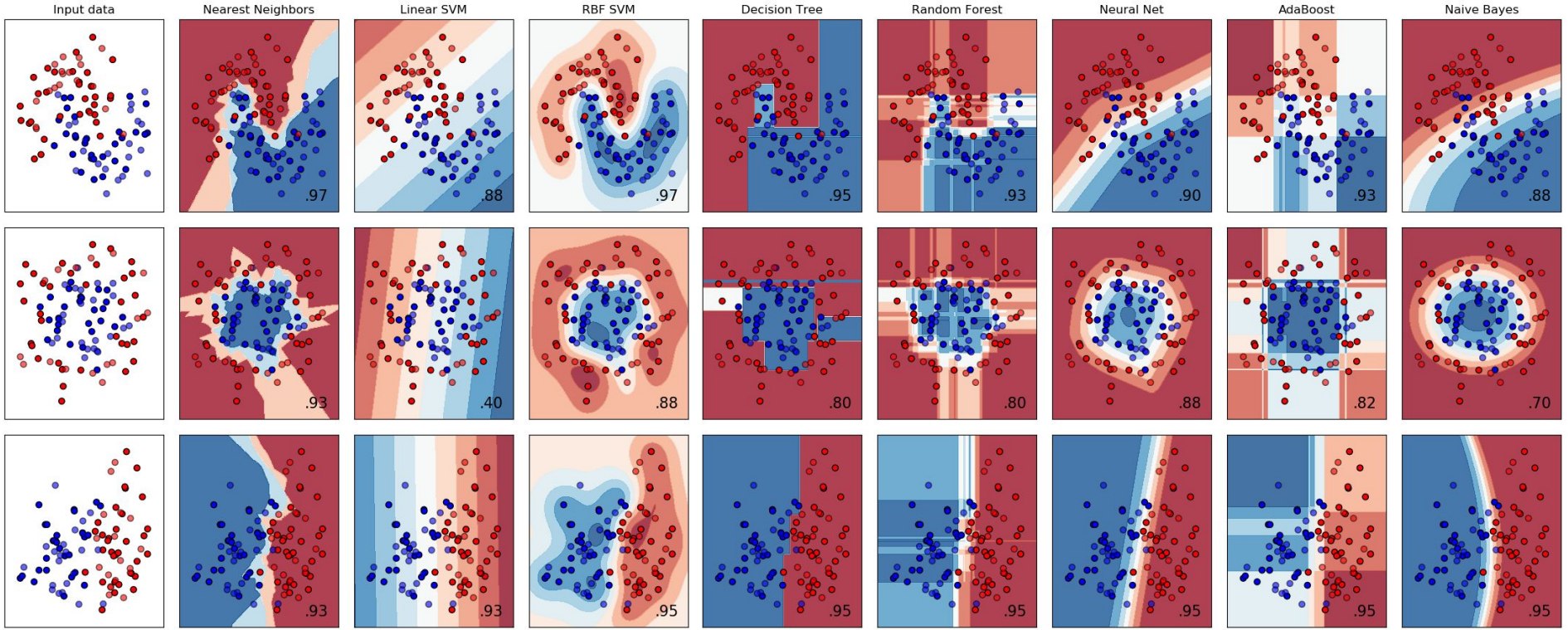
α_m increases with precision (less errors, bigger α_m)

The classifier G_m is trained with **increased error cost** for the observations which were misclassified by G_{m-1}.



from D. Lowe

A quick comparison



k=3 | C=0.025 | gamma=2 C=1 | maxdepth=5 | maxdepth=5 ntrees=10 | hidden layers=1 hidden units=100 | 50 trees with maxdepth=1 | Gaussian

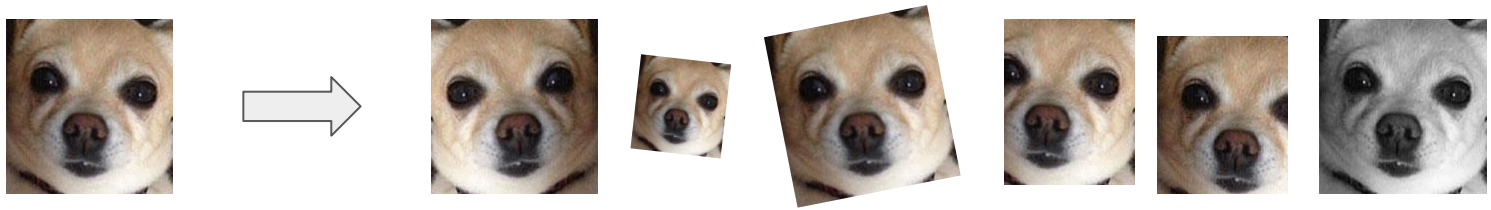
More here:
https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

More tricks

Data augmentation

Add realistic deformations to your input in order to improve domain coverage.

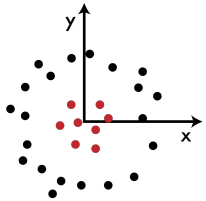
For **image data**, depending on what is possible in production: rotations, horizontal & vertical flips, scaling, translation, illumination change, warping, noise, etc.



For **vector data**: interesting problem. Possible approach: train/fit PCA then add random noise in low-energy features.

Reject

Several options:



1. *Improve the model of class boundary*
   In 1-vs-all training, add noise to the “others” samples.
2. *Adjust the decision function depending on your application*
   Look at the prediction probability of your classifier,
   and threshold it as per your need using a ROC curve.
3. *Model the noise*
   Add a “none” class to your classifier,
   with samples for real life cases of negative samples.

...