MLRF Lecture 06

J. Chazalon, LRDE/EPITA, 2019

🎉 with **Nicolas Boutry** 🎊
as guest lecturer 🎤

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 subspace

Non-linear classification
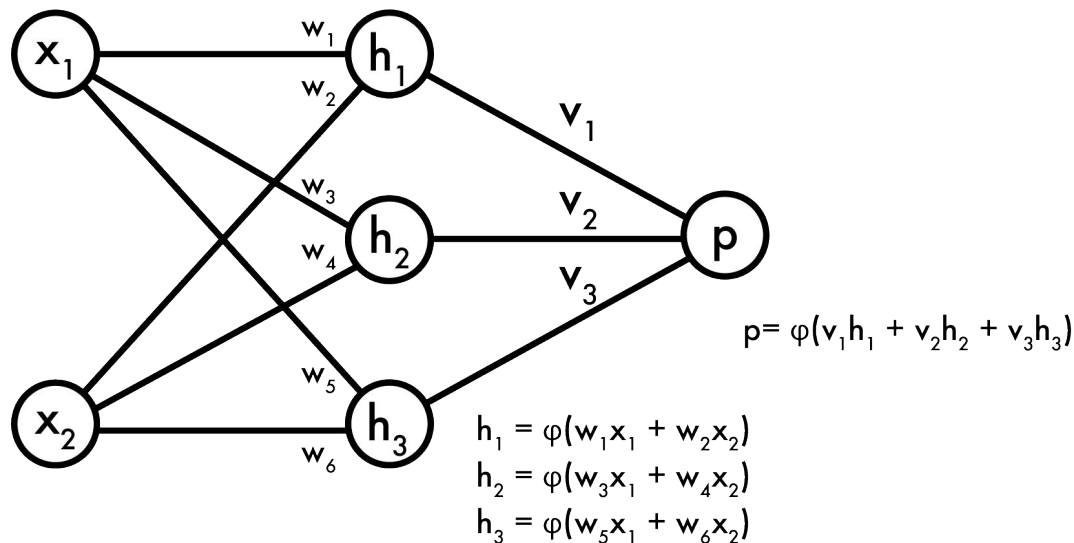# using combinations of linear classifiers

Multi-layer Perceptron

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

Perceptron
very simple linear classifier

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

$p = \varphi(v_1 h_1 + v_2 h_2 + v_3 h_3)$

$h_1 = \varphi(w_1 x_1 + w_2 x_2)$
$h_2 = \varphi(w_3 x_1 + w_4 x_2)$
$h_3 = \varphi(w_5 x_1 + w_6 x_2)$

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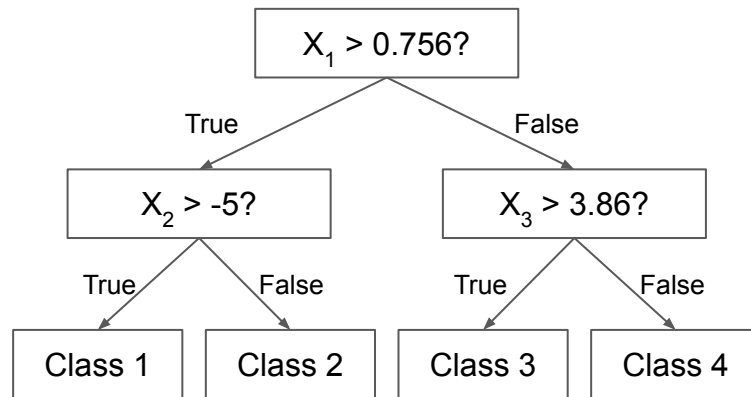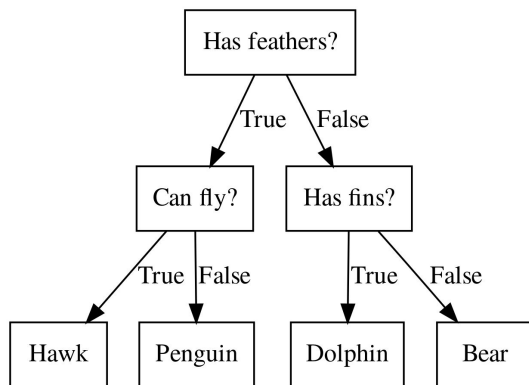
can model any continuous function f: $I_m \rightarrow R$

(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.

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).
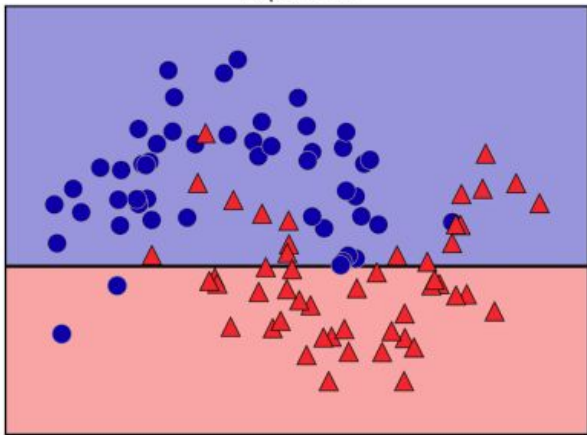
Decision trees
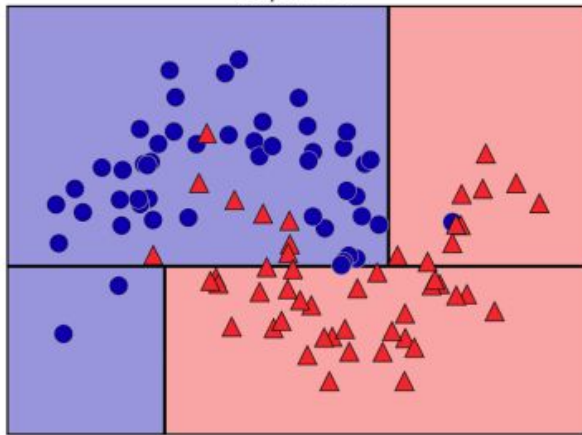
Very fast training and testing. Non parametric.

No need to preprocess the features.

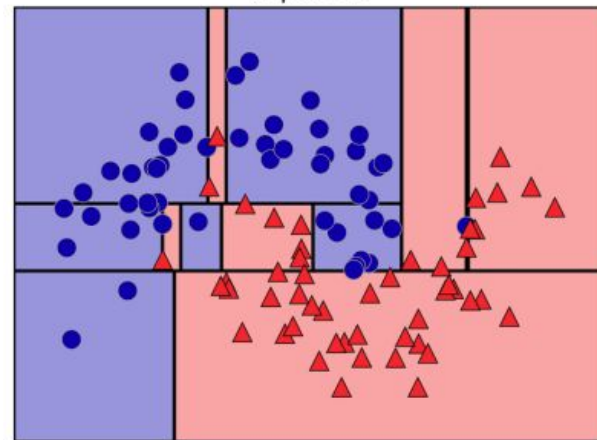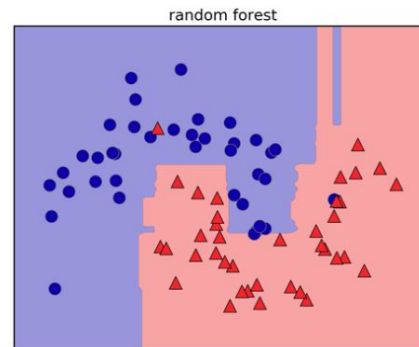BUT: Very prone to **overfitting** without strong limits on depth.

from A. Müller

Random Forests

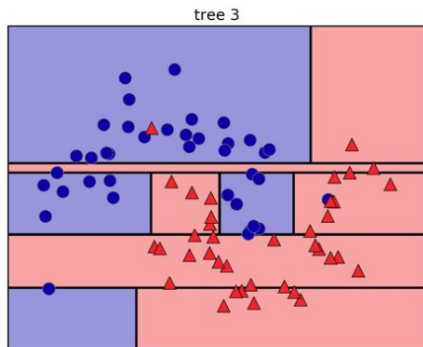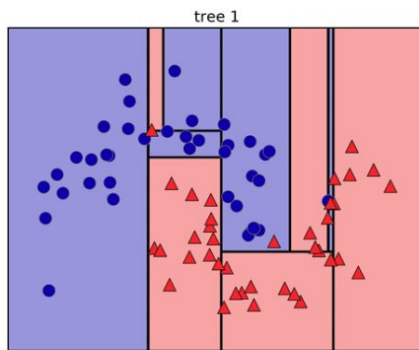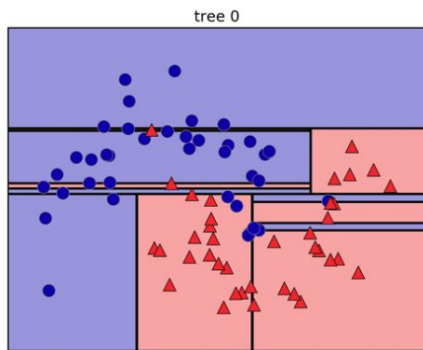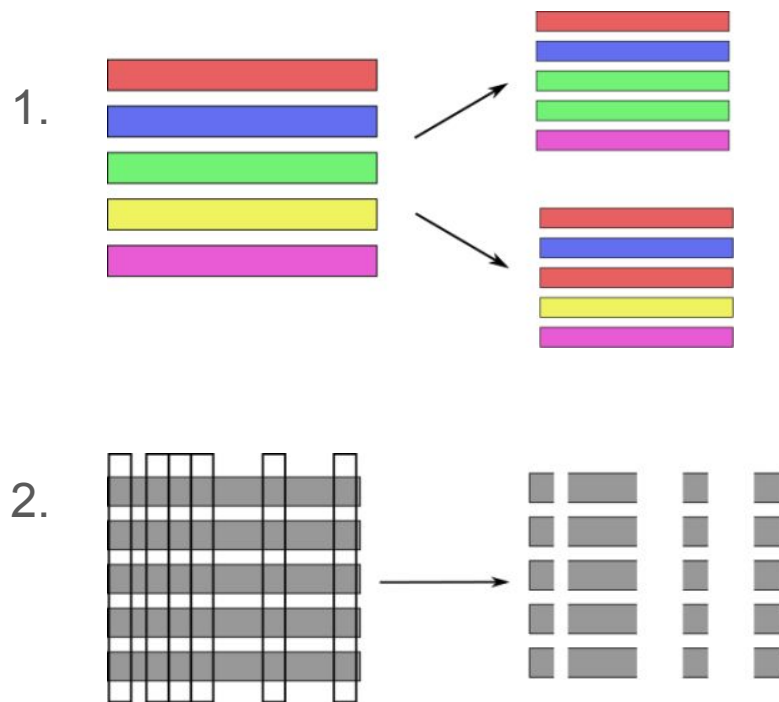**Average** the decision of multiple decision trees.

9
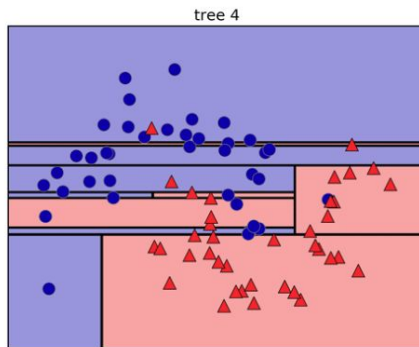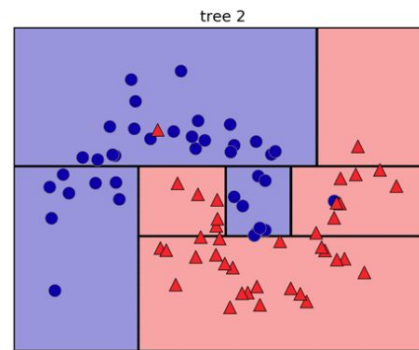
Random Forests

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.

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.

"Boosting", AdaBoost variant

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}.

Weak Classifier 1

Weights Increased

Weak Classifier 2

Weak classifier 3

Final classifier is linear combination of weak classifiers

from D. Lowe

13

A quick comparison

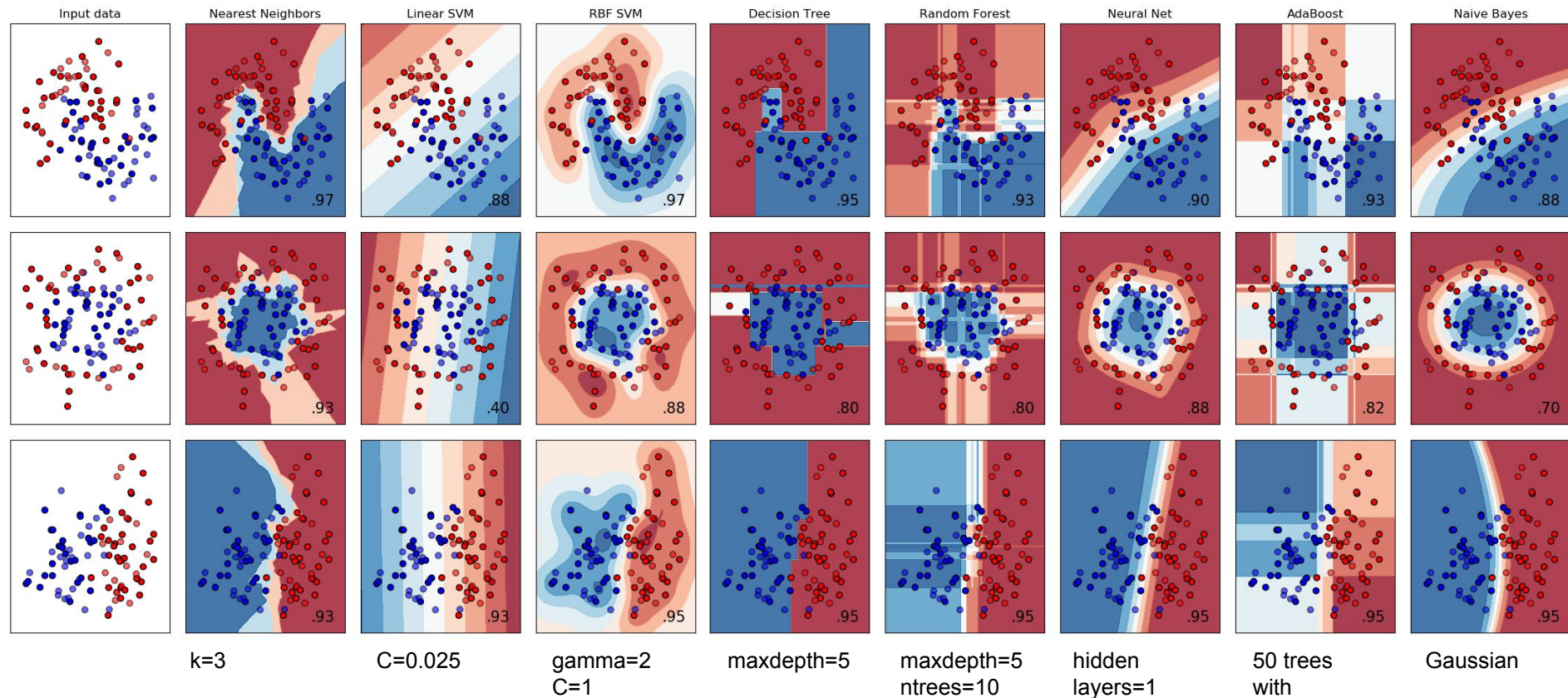|	Input data	Nearest Neighbors	Linear SVM	RBF SVM	Decision Tree	Random Forest	Neural Net	AdaBoost	Naive Bayes
Row 1		.97	.88	.97	.95	.93	.90	.93	.88
Row 2		.93	.40	.88	.80	.80	.88	.82	.70
Row 3		.93	.93	.95	.95	.95	.95	.95	.95

| 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

14

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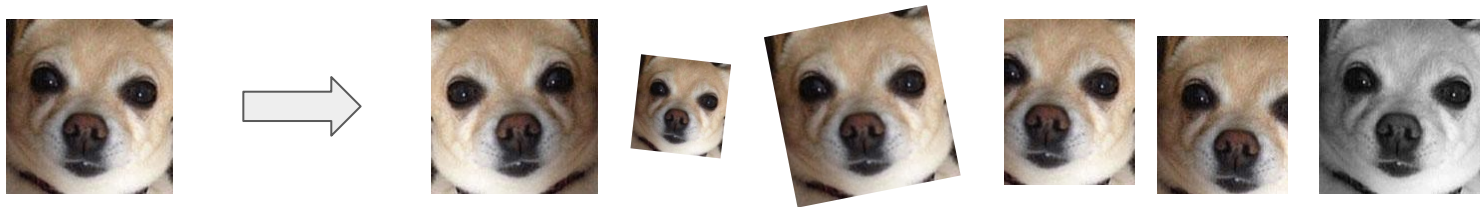
For **vector data**: interesting problem. Possible approach: train/fit PCA then add random noise in low-energy features.

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.

...