

# Foundations of Machine Learning and AI

Theodoros Evgeniou - Nicolas Vayatis

Sessions 9-10: Ensemble methods

# Principle of machine learning

## Regularized optimization

- Objective: to find a function that fits the data and displays predictive power
- Until now: Learning amounts to the minimization of training error for some loss function over the hypothesis class of functions  $h \in \mathcal{H}$  plus some penalty

$$C_n(h) = \underbrace{\hat{L}_n(h)}_{\text{Training error}} + \lambda \underbrace{\text{pen}(h, n)}_{\text{Regularization}}$$

- Example : ridge regression where  $h(x) = \theta^T x$ :  
 $\hat{L}_n(h) = \frac{1}{n} \sum_{i=1}^n (Y_i - \theta^T X_i)^2$  and  $\text{pen}(h, n) = \frac{1}{n} \|\theta\|_2^2$
- The penalty grows with the complexity of  $h$  (or the size of  $\mathcal{H}$ ) and vanishes when  $n \rightarrow \infty$

# Machine Learning Methods

## Optimization is central

Examples seen so far:

- (Sparse) Linear models  $\longrightarrow$  gradient method (and extensions)
- Kernel ridge regression  $\longrightarrow$  quadratic optimization (with KKT conditions)
- Deep learning  $\longrightarrow$  nonconvex optimization (stochastic gradient descent) + implicit regularization (tricks)

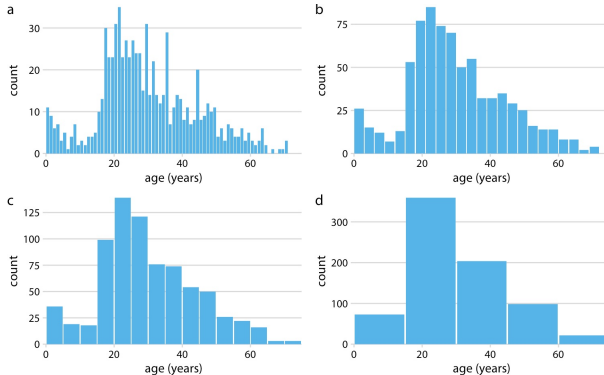
# This session

## Other forms of regularization

- General idea: Regularized function estimation without global optimization
- Two directions:
  - Local methods: nearest-neighbors and decision trees
  - Ensemble methods: bagging, boosting, random forests

# Regularization without optimization

## The case of histograms



Distribution of the age of the passengers of the Titanic with bins varying from 1 year to 15 years

# Ingredients for that type of regularization

- Histograms use two general ideas of locality (bins) and averaging (piecewise constant function)
  - define local: which training data can be considered to be close to the point where a prediction has to be made?
  - averaging (or voting if discrete outcome) : take the average of the values over each bin
- Regularization through hyperparameter selection: find the optimal bin size amounts to finding the right hypothesis class

# From histograms to Machine Learning

- In the previous example, the objective was to estimate a density function from a sample drawn from this distribution (problem known in the literature as *nonparametric density estimation* or *kernel density estimation*)
- Density estimation can be seen as an *unsupervised learning problem*
- In the supervised setting, we establish the values of the function on every bin either by averaging (regression setup) or by voting (classification setup). The general terminology for averaging/voting is aggregating/combining.

## A. Older Machine Learning approaches: Local methods

1. Nearest neighbors
2. Decision trees



## Two popular types of local methods

- Nearest neighbors: local are the closest points
- Partition-based rules (also called *decision trees*): local are the points within a cell from a partition of the input space only

Works for classification, regression and other problems... but here we will focus on classification

# Problem considered (Multiclass) Classification

- Given:

Consider a sample of classification data

$$(X_1, Y_1) \dots (X_n, Y_n)$$

where  $X_i \in \mathbb{R}^d$  vector of independent variables,  $Y_i \in \{1, \dots, C\}$   
the label

- Want:

to predict the label  $y$  at any position  $x$

## A. Local methods

### 1. $k$ -Nearest neighbors ( $k$ -NN)

# $k$ -Nearest Neighbor (1/4)

## Principle of the $k$ -NN algorithm

### 1 Compute distances

Compute pairwise distances  $d(x, X_i)$  for all  $i = 1, \dots, n$

### 2 Sort training data

Sort the data points from the closest  $X_{(1)}$  to the farthest  $X_{(n)}$   
(i.e.  $d(x, X_{(1)}) \leq \dots \leq d(x, X_{(n)})$ )

### 3 Prediction $\hat{h}(x, k)$ = Majority vote of the $k$ -NN

Consider the labels  $Y_{(1)}, \dots, Y_{(k)}$  of the  $k$  closest points to  $x$   
and take the majority vote

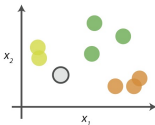
$$\hat{h}(x, k) = \arg \max_c \left\{ \sum_{l=1}^k \mathbb{I}\{Y_{(l)} = c\} \right\}$$

# $k$ -Nearest Neighbor (2/4)

## Principle of the $k$ -NN algorithm

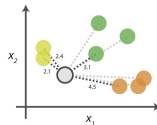
### kNN Algorithm

#### 0. Look at the data







Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

#### 1. Calculate distances









Start by calculating the distances between the grey point and all other points.

#### 2. Find neighbours

Point	Distance	
	2.1	→ 1st NN
	2.4	→ 2nd NN
	3.1	→ 3rd NN
	4.5	→ 4th NN

Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

#### 3. Vote on labels

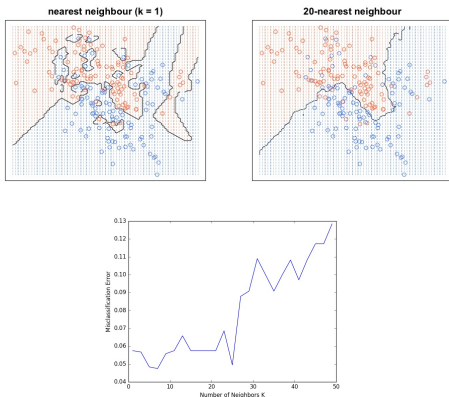
Class	# of votes	
	2	→ Class  wins the vote! Point  is therefore predicted to be of class  .
	1	
	1	

Vote on the predicted class labels based on the classes of the  $k$  nearest neighbours. Here, the labels were predicted based on the  $k=3$  nearest neighbours.

# Nearest Neighbors (3/4)

## Hyperparameters

- Choice of a distance  $d$  between points of  $\mathbb{R}^d$
- Number  $k$  of Nearest Neighbors, estimated by cross-validation:



# $k$ -Nearest Neighbor (4/4) Theory

- Recall: classification error  $L(h) = \mathbb{P}(Y \neq h(X))$  and  $L^* = \inf L$
- Consistency result:

$$\mathbb{E}L(\hat{h}(\cdot, k_n)) \rightarrow L^*$$

under the condition:  $k_n \rightarrow \infty$  and  $k_n/n \rightarrow 0$  when  $n \rightarrow \infty$

- No closed-form solution for optimal  $k_n$  (in practice, we use cross-validation)
- No theoretical clue on the choice of the distance (related to data representation and the physics of the problem)

## A. Local methods

### 2. Partition-based (decision trees)

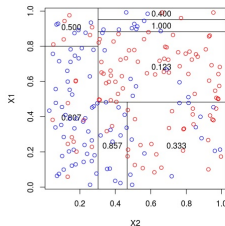
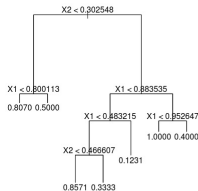


# Partition-based classifier (1/4)

## Computing the prediction for fixed partition

Denote the partition by  $c = \bigcup_j \gamma_j$  with cells  $\gamma_j$

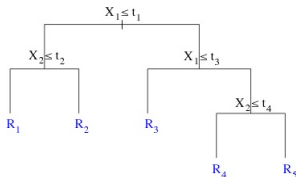
- 1 Find the cell  $\gamma(x)$  where  $x$  falls
- 2 Consider the training data in the cell  $\gamma(x)$
- 3 Prediction  $\hat{h}(x, c) = \text{Majority vote over the training data in cell } \gamma(x)$



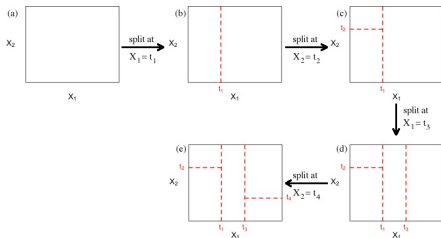
# Partition-based classifier (2/4)

## Building data-driven partitions

- Start with all the training data and find a (simple) classifier which minimizes some cost function
- Repeat the process with the subset of training data on each side of the frontier of the classifier → this is called *recursive partitioning*



tree representation



recursive partitioning of the  $X$ -domain

## Partition-based classifier (3/4)

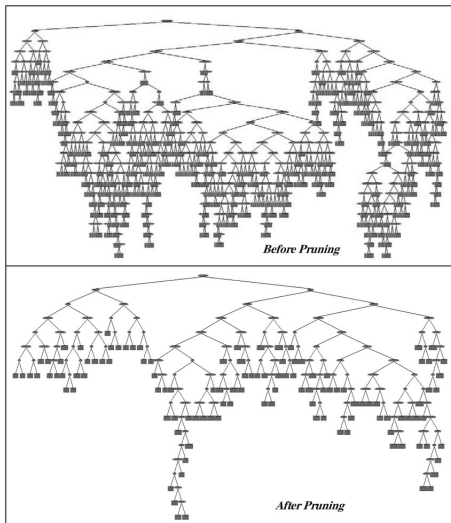
### Hyperparameters

- Cost function optimized locally (at the cell level for the data within the cell)
- Number of minimal points in a cell
- Maximal depth of the tree or total number of cells estimated by pruning the tree - pruning amounts to explore the class of all subpartitions (subtrees) and optimize a penalized criterion of the form

$$\arg \min_c \hat{L}_n(h_c) + \lambda |c|$$

where  $c \subset \hat{c}$  is the collection of subpartitions obtained from the learned partition by pruning from bottom to top

# Pruning example



# Partition-based classifier (4/4)

## Theory

- Case of regular partitions with cells which are hypercubes of  $\mathbb{R}^d$  with edges of length  $\delta_n$ :

$$\mathbb{E}L(\hat{h}(\cdot, \delta_n)) \rightarrow L^*$$

under the condition:  $n\delta_n^d \rightarrow \infty$  and  $\delta_n \rightarrow 0$  when  $n \rightarrow \infty$   
(need enough data points in every cell and cell diameter go to zero as sample size grows)

- Case of data-driven partitions: VC and Rademacher theory applies

## Take-home message on local methods

### Major limitations:

- The  $k$ -Nearest Neighbor method requires to store all the training data in order to predict the label of new entries.
- Decision trees are extremely unstable.
- Both display prediction performance below state-of-the-art methods

### Virtue of decision trees:

- Can handle missing/categorical data, scale change
- Can be expressed in terms of logical rule → explainable machine learning

What can be saved from decision trees?

## B. Shallow and efficient Machine Learning algorithms: Ensemble methods

1. Bagging and Random Forests
2. Boosting



# Motivation for ensembles

## Pointers to other fields

- Technology: the champions in data science competitions combine several methods to boost performance (e.g. BelKor team, winner of the Netflix challenge)
- Decision theory: Social choice theory
- Probability: Ergodic theorem
- Nonparametric statistics: aggregation estimators

# Ensemble methods

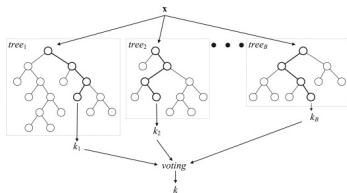
## Starting point

- Consider we already have a machine learning algorithm with reasonable performance that we want to improve, e.g. decision tree,  $k$ -NN, SVM, ...
- The idea of the ensemble is to generate different functions from the same training data and the same hypothesis space
- In the illustration coming next and most of the discussion, the basic hypothesis space is the one with decision trees obtained with orthogonal splits (such splits are called decision stumps).

# Ensembles of decision trees

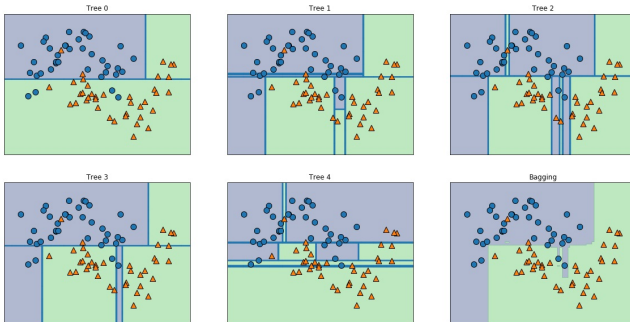
## General principle

- Generate a collection of *weak* predictors (ensemble) obtained with a basic Machine Learning algorithm (e.g. decision tree)
- For every point  $x$ , compute their individual predictions
- Take an average or a majority vote of the individual predictions to determine the prediction of the ensemble



# Ensembles of decision trees

## Resulting classifier



# Ensembles of decision trees

## Three popular methods

- Bagging (Breiman, 1996)
- Random forests (Amit-Geman, 1997; Breiman, 2000)
- Boosting (Freund-Schapire, 1996)

## B. Ensemble methods

### 1. Bagging and Random Forests

# Bagging and Random Forests

## What is their hypothesis space?

- Denote by  $\mathcal{H}$  the base hypothesis space (for the not so brilliant algorithm we already have, e.g. decision trees)
- Denote by  $D_n$  the training data and assume that we can sample functions  $\hat{h}_1, \dots, \hat{h}_t$  (the ensemble) from  $\mathcal{H}$  conditionally to  $D_n$
- With an ensemble of  $T$  functions, the output of bagging/random forests is the average of those "random" (generated based on the data) functions:

$$\hat{f}_T = \frac{1}{T} \sum_{t=1}^T \hat{h}_t$$

- The hypothesis space for those methods is the linear span of the base hypothesis space  $\mathcal{H}$ . This can be a huge space!

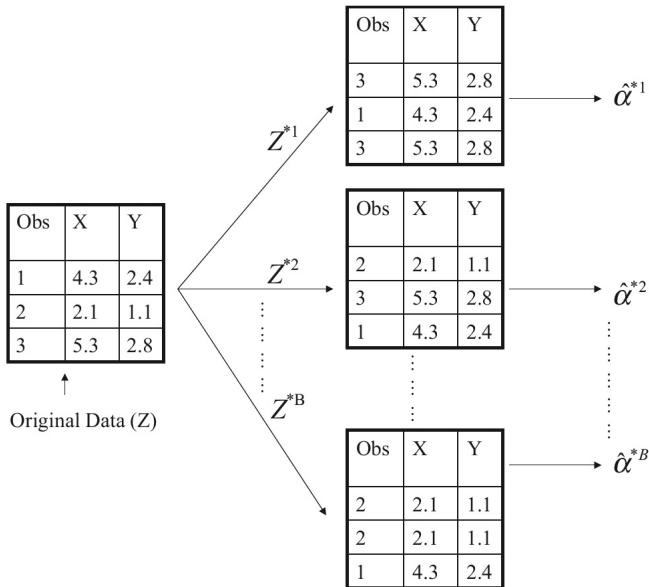
# How to generate the ensemble?

## Bootstrap and aggregation

- Bagging and random forests rely on bootstrap samples of the training data
- They differ by some different specifications of the recursive partitioning procedure to build each tree (no pruning involved)



# What is bootstrap in general?



# Bagging Theory

- Consistency result for some idealized version of bagging
- Most important! Bagging can render inconsistent rules consistent!
  - Biau, Devroye and Lugosi (2008) have considered bagging applied to 1-NN, given that 1-NN is inconsistent in general classification scenarios (except zero-noise or pure random labels)
  - Bagging applied to 1-NN classifier is consistent under some reasonable conditions on the sampling process

## B. Ensemble methods

### 2. Boosting

## Historical perspective on Boosting

- Original paper: Freund, Y. and Schapire, R. E. (ICML, 1996).
- Interpretation of the optimization problem solved as stochastic gradient descent: Friedman, J. H. (CSDA, 2002).
- Wald Memorial lecture (IMS, 2000): Leo Breiman declares that *"understanding Boosting is the most important problem in Machine Learning"*
- Proof of boosting consistency: Lugosi, G. and Vayatis, N. (Special issue with discussion of the Annals of Statistics, 2004).
- Xgboost, a scalable implementation: Chen, T. and Guestrin, C. (ACM SIGKDD, 2016).

# Boosting (1/7)

## Principle

- **Input**

- Data sample  $D_n = \{(X_i, Y_i) : i = 1, \dots, n\}$  with classification data  $\{-1, +1\}$
- Base hypothesis class  $\mathcal{H}$  of *weak* classifiers such as decision trees (assumed to be symmetric, i.e.  $h \in \mathcal{H}$  iff  $-h \in \mathcal{H}$ )

- **Iterations**  $t = 1, \dots, T$ .

- Compute weights  $w_t > 0$  and weak classifiers  $\hat{h}_t \in \mathcal{H}$

- **Output.**

- The Boosting classifier takes the sign of the following linear combination of weak classifiers:  $\hat{f}_n(x) = \sum_{t=1}^T w_t \hat{h}_t(x)$

## Boosting (2/7)

### Notations

- Boosting distributions on the data: sequence of discrete probability distributions over  $\{1, \dots, n\}$  denoted by  $\Pi_t$ ,  $t \geq 1$
- Weighted training error: for any weak classifier  $h \in \mathcal{H}$  and for  $t \geq 1$

$$\hat{\varepsilon}_t(h) = \sum_{i=1}^n \Pi_t(i) \mathbb{I}\{h(X_i) \neq Y_i\}$$

## Boosting (3/7)

### Original Algorithm: AdaBoost

- 1 **Initialization.**  $\Pi_1$  is the uniform distribution on  $\{1, \dots, n\}$
- 2 **Boosting iterations.** For  $t = 1, \dots, T$ , find the weak classifier such that :

$$\hat{h}_t = \arg \min_{h \in \mathcal{H}} \hat{\varepsilon}_t(h)$$

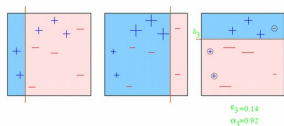
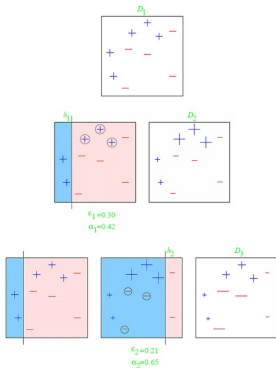
then set  $e_t = \hat{\varepsilon}_t(\hat{h}_t)$  and take the weight to be

$$w_t = \frac{1}{2} \log \left( \frac{1 - e_t}{e_t} \right)$$

- 3 **Boosting distribution update.** For any  $i = 1, \dots, n$ ,

$$\Pi_{t+1}(i) \propto \Pi_t(i) \exp \left( -w_t Y_i \cdot \hat{h}_t(X_i) \right)$$

# Boosting (4/7) Example



$$H_{\text{final}} = \text{sign} \left( 0.42 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array} \right)$$

$$= \begin{array}{|c|} \hline \text{blue} \\ \hline \end{array}$$



## Boosting (5/7)

- Boosting can be interpreted as a functional gradient descent on the following functional:

$$\hat{A}_n(f) = \frac{1}{n} \sum_{i=1}^n \exp(-Y_i f(X_i))$$

where  $f$  is taken in a hypothesis space which is the linear span of 'simple' set  $\mathcal{H}$  of classifiers.

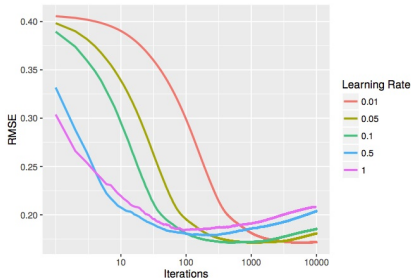
- Exercise: why?

Refer to: J. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine", The Annals of Statistics, Vol. 29, No. 5, 2001.

# Boosting (6/7)

## Hyperparameters for Gradient Boosting

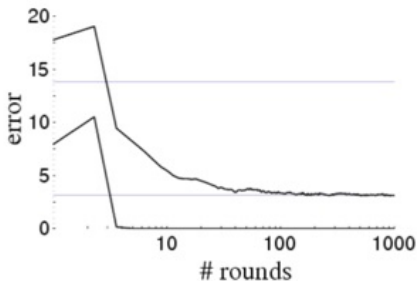
- The number  $T$  of iterations: the bigger, the higher the chance of overfitting.
- The stepsize  $\eta$  is fixed: decreasing learning rate tends to improve generalization performance.



## Boosting (7/7)

### A mystery not fully explained yet...

The test error continues to drop along the iterations even though the training error is zero  $\rightarrow$  Regularization effect thanks to averaging ??



# Packages

- Python: scikit-learn
- R:
  - rpart: recursive partitioning
  - caret: classification and regression training (SVM, random forest...)
  - xgboost: extreme gradient boosting

## Further topics

- Explainability
- Reinforcement Learning
- Adapting these concepts to other problems:
  - either in terms of objectives: such as preference learning, scoring, ranking, anomaly detection, novelty detection...
  - or in terms of learning setups: online learning, unsupervised learning, transfer learning, multitask learning, budgeted learning, active learning...