Ensemble Methods and Aggregation of Experts

Yannig Goude [1]

[1] EDF R&D, EDF Lab Saclay
LMO

15 février 2023

Sommaire

Introduction

- The complex structure of real life data makes it complicated to design a single model which performs well in all the situations. A good idea and a research hot spot is to create a set of models, each with different properties, different information and aggregate them in a single forecast to obtain better performances
- Ensemble models tend to outperform single models in a lot of applications :
  - Kaggle competition, M competition (focus on time series), KDD cups : ensemble methods, in particular gradient boosting trees and its powerful/clean implementation xgboost are very popular ([BM21]).
  - They have been applied in all the field of application of machine learning : emotion recognition, medical diagnosis, weather forecasting, electtricity load forecasting, solar/wind power forecasting, financial forecasting...
  - We use ensemble learning in day life e.g. ratings of products.
- Ensemble learning is more of a framework, it can be combined with any machine learning method.
- Tukey introduced this concept in 1977 (2 stages linear regression), Schapire introduced boosting in 1990, Breiman introduced Baggind in 1996 the first step to random forest in 2001.

The Bagging

Introduce by Breiman in 1996 [Bre96].

- two major ingredients : **B**ootstrap and **agg**regation
- **aggregation** of **independent** forecasts (base or week learners) induce an important error reduction
- the data scientist needs to aggregate learner as independent as possible
- naïve idea : train our base learners (ex : decision trees as CART) on different splits of the data
- pb : we live in a finite word, splitting the data will entail very low performance of the base learners
- we need to respect a trade off between quality of the learner-independence of the learners

Bagging

Introduce by Breiman in 1996 [Bre96] and contains two major ingredients : **B**ootstrap and **agg**regation.
The first idea is that **aggregation** of **independent** forecasts (base or week learners) induce an important error reduction
Suppose we model $\mu(x) = \mathbb{E}[Y/X = x]$ with a family of estimates $(\widehat{\mu_1}, ..., \widehat{\mu_B})$ and the uniform aggregation of it : $\widehat{\mu} = \frac{1}{B}\sum_{i=1}^{B} \widehat{\mu}_i$.

The bias-variance decomposition of the error of $\widehat{\mu}$ is :

$$\mathbb{E}\left[(\widehat{\mu}(X) - \mu(X))^2\right] = (\mathbb{E}[(\widehat{\mu}(X)] - \mu(X))^2 + \mathrm{Var}(\widehat{\mu}(X))$$

supposing that the $\widehat{\mu}_i$ are iid gives that :

- the aggregation has the same bias than each individual $\mathbb{E}[\widehat{\mu}(X)] = \mathbb{E}[\widehat{\mu_1}(X)]$
- but the variance of the aggregation is lower :

$$\mathrm{Var}(\widehat{\mu}(X)) = \frac{1}{B}\mathrm{Var}(\widehat{\mu_1}(X))$$

Bagging

- So, the data scientist needs to aggregate learners as independent as possible, naïve idea : train our base learners (ex : decision trees as CART) on different splits of the data
- pb : we live in a finite word, splitting the data will entail very low performance of the base learners
- we need to respect a trade off between quality of the learner-independence of the learners

Bagging algorithm

Algorithm: Bagging

Data: a sample of observations $D_n = (x_i, y_i)_{i=1,\dots,n}$, $x_i \in \mathbb{R}^p$, $y_i \in \mathbb{R}$
Parameter: B number of learners (=nb of boostrap samples), a base learner (e.g. CART)
for $k = 1, ..., B$ **: do**

1. sampling with replacement a boostrap sample $\tilde{D}_n$
2. estimate $\widehat{\mu_k}$ on $\tilde{D}_n$ with our learner

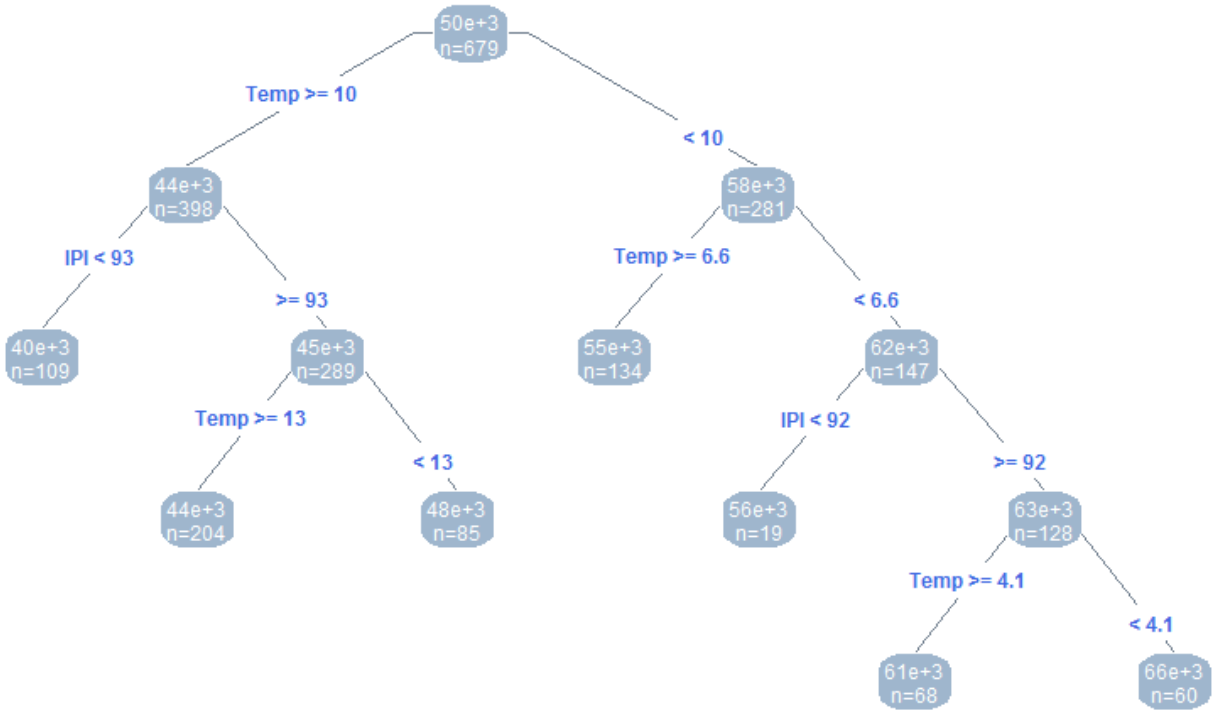
end
Result: $\frac{1}{B}\sum_{k=1}^{B} \widehat{\mu}_k$

- each **base learner** is learned on a subsample of the original data
- in the complementary data we can compute the **out-of-bag error**
- base learners can be computed in parallel making bagging computationaly efficient
- base learners need to be simple (fast computation), data adaptive, making CART a good candidate

CART : Classification and Regression Tree

- CART is an algorithm to build binary trees
- it is based on a recursive partition of the data
- it can deal we different types of covariates : continuous, discrete, ordered...
- it's a local model (contrary to linear regression), making it more data adaptive

```
rpart(Load ~ NumWeek + Temp + IPI, data = data0)
```



CART : Classification and Regression Tree

The objective of CART is to find a partition of the data in regions $R_1, ..., R_J$ (leaf of the tree) minimising

$$RSS = \sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \widehat{y}_j)^2$$

where $\widehat{y}_j = \frac{1}{n_j} \sum_{i \in R_j} y_i$, n_j the number of observations in leaf R_j. This problem is computationally intensive and CART is a greedy recursive approximation of this optimal partition.

CART : Classification and Regression Tree

For a covariate $X_j, j = 1, ..., p$, denote $R_-(j, s) = \{X_j < s\}$ et $R_+(j, s) = \{X_j \geq s\}$

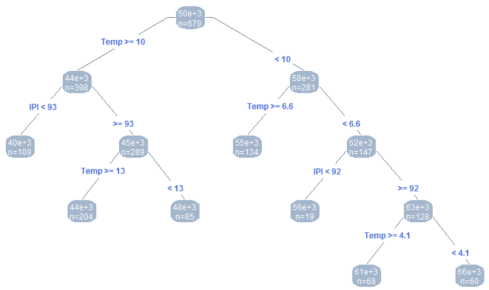
Algorithm: CART

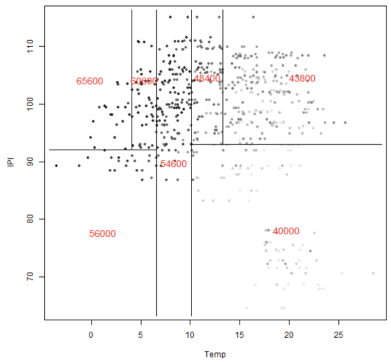
Data: a sample of observations $D_n = (x_i, y_i)_{i=1,...,n}$, $x_i \in \mathbb{R}^p$, $y_i \in \mathbb{R}$
Parameter: stopping criteria : stop
while **stop=FALSE** **do**

- In each leaf of the tree, find the covariate X_j and the treshold s minimising :

$$\text{RSS} = \sum_{i \in R_-(j,s)} (y_i - \widehat{y}_{R_-})^2 + \sum_{i \in R_+(j,s)} (y_i - \widehat{y}_{R_+})^2$$

end
Result: a binary tree partitioning the data, a piecewize constant approximation of f,
$y = f(x) + \varepsilon$





CART : Classification and Regression Tree

Example of stopping criteria are :

- a number of observation in the leafs
- a reduction the error criteria under a treshold
- depth of the tree...

Another alternative is **prunning**.

- the idea is to build maximal trees (no stopping criteria)
- then prune the tree (regrouping leafs) afterward using a selection criteria (AIC, CV, test set...)

A maximal tree has a large variance but a low bias. CART can be easily computed but very unstable (large variance) making it a very good candidate for ensemble methods.

Random forest

Algorithm: Random forest

Data: a sample of observations $D_n = (x_i, y_i)_{i=1,\dots,n}$, $x_i \in \mathbb{R}^p$, $y_i \in \mathbb{R}$

Parameter: B : number of trees, α_n : size of bootstrap samples, m_{try} : nb of variables randomly sample

for i=1,...B do

- sample α_n points from D_n (with or without replacement)
- build a binary tree on this sample this way :
  - at each note select randomly m_{try} variables
  - select the best split by optimizing the CART-split criterion along the m_{try} candidate variables

end

Result: a forest averaging the B trees, the predicted value of the forest at x is the average of the forecast of each tree at x.

---

rks :

- the for loop could be seen as parallel estimation of the B independently generated trees.
- $m_{\text{try}} = p$ results in bagging CART, $m_{\text{try}} = 1$ purely random choice of variable to do a split. In practice defaut value of $m_{\text{try}}$is $p/3$ for regression and $\sqrt{p}$ for classification.

Variant of random forest

Many variants of random forest has been proposed :

- Pure Forest : variables and splits are randomly generated (independent of the observations), useful to study theretical properties as in [AG14].
- Extra-Trees (Extremely Randomized Trees) [GEW06] : for each split select randomly m_{try} variables and thresholds (uniform distribution on the support of the variables), chose the best according to CART criteria.
- Random subspace ([LTF05] : for each tree, select randomly m_{try} variables used top optimize all the splits.

OOB error, Variable importance

- Out Of Bag error : thanks to the use of bagging, each tree leaves apart a subset of the data. For each observation, we can compute an approximation of the forest forecast using only the trees built without this data.
- OOB error can be useful to compute a score of importance of the variables in the forest. For a variable X^j the permutation importance $VI(X^j)$ is :
  - compute the OOB error E
  - randomly permutes the observations x_i^j and the OOB error E^j
  - the variable importance is $E^j - E$

Another way to compute the importance is Gini importance : at each split we allow to the splitting variable an importance which is the improvement in the split criterion, then sum over all trees.

Implementation in R

- historical package `randomForest` of Breiman :

```
rf<- randomForest(Load~Temp+Temp1+Time+Load1+ToY, ntree=500,
  mtry=3, data=data0)
rf.forecast=predict(rf, data=data1)
```

- package `ranger` (parallel optimisation)

```
rf <- ranger(formula,  num.trees = 500, mtry=7, data=data0)
rf.forecast <- predict(rf, data=data1)$predictions
```

- package `xgboost`, RF can be seen as special case of boosting

```
dataxgb0<-as.matrix(data0[,-c(2,3,4,6)])
dataxgb1<-as.matrix(data1[,-c(2,3,4,6)])
xgb0 <- xgboost(params=list(subsample=0.9,  colsample_bytree=0.5, eta=1),
 objective = "reg:squarederror",  nround = 500, num_parallel_tree=500,
  num_boost_round=1, booster = "gbtree", data =dataxgb0 , label=data0$Load)
```

RF, pros and cons

Pros

- RF don't overfit (warning : variable importance)
- very efficient : bagging generate diversity but also OOB error
- ensemble of trees is generate in parallel (contrary to boosting where the trees are computed sequentially)
- easy to calibrate (very few parameters : weak learner parameter, B, m_{try})
- often use as benchmark in a lot a ML problem/challenge

Cons

- black box : difficult to interpretate, to improve it
- random forest are often "no too bad" but not excellent

Boosting

Introduce by [Sch90] in 1990 then [FS+96] proposed adaboost, the first boosting algorithm.
Boosting is an ensemble method based on learning a larger number of weak learner.
Whereas in the RF the learner are learnt idependently from each other, in boosting learning is done sequentially.
The base elements of boosting are :

- a base learner or weak learner
- gradient descent

If we have access to a weak learner, a little bit better than random guess, we can **boost** it to obtain a good model.

Boosting

Let play the following game :
We have a sample $D = (x_i, y_i)_{i=1,\dots,n}$ and we want to adjust a regression model $F(x)$ minimizing the quadratic error.
Suppose that we have a model $F(x)$ provided by a friend. This model is OK but not perfect. How can you improve it knowing that :

- you don't have the right to modify the parameters of F
- you can add to F a simple correction h based on a simple model to build a new prediction $F(x) + h(x)$

Boosting

Ideally, we could choose h such that :

$$\begin{aligned} F(x_1) + h(x_1) &= y_1 \\ F(x_2) + h(x_2) &= y_2 \\ &\dots \\ F(x_n) + h(x_n) &= y_n \end{aligned}$$

ie

$$\begin{aligned} h(x_1) &= y_1 - F(x_1) \\ h(x_2) &= y_2 - F(x_2) \\ &\dots \\ h(x_n) &= y_n - F(x_n) \end{aligned}$$

Boosting

We can do that approximatively with a regression tree base learner.

- We can fit a tree on the data set :

$$(x_1, y_1 - F(x_1)), ..., (x_n, y_n - F(x_n))$$

  where $y_i - F(x_i)$ are called the residuals.
- If $F + h$ is not good enough, we can add a new tree on the new residuals and etc.

This algorithm will, by construction, obtain a better approximation error on the learning set but can we guarantee that it will be the case for the generalization error ?

Boosting

We can see the boosting as a **gradient descent** procedure. Denote $L(y, F(x)) = (y - F(x))^2/2$ our loss function. We aime at minimizing :

$$J = \sum_{i=1}^{n} L(y_i, F(x_i))$$

Denote $v_i = F(x_i)$, we have

$$\frac{\partial J}{\partial v_i} = \frac{\partial \sum_{i=1}^{n} L(y_i, v_i)}{\partial v_i} = F(x_i) - y_i$$

and residuals can thus be seen as negative gradients :

$$r_i = y_i - F(x_i) = -\frac{\partial J}{\partial v_i}$$

Boosting

$$F(x_i) := F(x_i) + h(x_i)$$
$$F(x_i) := F(x_i) + y_i - F(x_i)$$
$$F(x_i) := F(x_i) - \frac{\partial J}{\partial v_i}$$
$$v_i^{k+1} := v_i^k - \rho \frac{\partial J}{\partial v_i}(v_i^k)$$

In the case of least square regression, residuals are the negative gradient of the empirical loss and adjusting h on it corresponds to update our original model with a gradient descent.

- The choice of h and the number of boosting iterations are essential
- the choice of the gradient step is crucial. A rule of thumb is to choose it small $\rho = 0.1$ and increase the nb of iterations.
- as in RF, random sampling can be done to improve the optimization (resulting also to useful OOB data).

Implementation of boosting

- package `gbm` : gradient boosting with CART as base-learners
- package `mboost` : gradient boosting with penalized regression splines as weak learners, B-splines, linear model, stumps (tree with 2 leafs).
- package `xgboost` : CART or linear regression as base-learners, the reference package for boosting trees, very efficient implementation.
- package `catboost` : gradient boosting with CART, discretization of continuous variable (compression), GPU parallelism.
- package `lightgbm` : gradient boosting with CART, clever tricks to accelerate tree computation (GOSS : Gradient BAsed One Side Sampling : inclue observations with high gradients+random spling of low gradient observations at each step)

Online learning

We consider now the problem of ensemble learning for time series. One interesting framework (but not the only one)is **online aggregation of experts**.

- we suppose to observe sequentially with time our target $y_1, ..., y_n$.
- we don't assume any stochastic model on the data generation process. Algorithms are designed to work on any sequence of observations/expert forecasts.

More precisely, at each round t

- we suppose to have access to N forecasts called **experts** who propose a forecast $f_{j,t}$ of y_t
- given a loss function, our goal is to produce the best aggregation of these experts based on $y_1, ..., y_{t-1}$ and $(f_{j,1}, ..., f_{j,t-1})_{j=1,..,N}$.

Experts can come from statistic or machine models, a physical model, a human expertise...
Original idea in 1957 [Han57] and [Bla+56] in the framework of game theory. In machine learning, this pb has been proposed in the early 90s by Vovk [Vov90].
Reference book : [CBL06].

Online aggregation of experts

Algorithm: Online aggregation of experts

Data: sequential target y_t and N experts $f_{j,t}$
Parameter: loss $l : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}_+$, aggregation strategy ***S***
for t=2,...T do

- we observe $y_1, ..., y_{t-1}$, $(f_{j,1}, ..., f_{j,t-1})_{j=1,..,N}$ and associated losses
- our aggregation strategy ***S*** compute weights :

$$p_t = (p_{1,t}, ..., p_{N,t})$$

- expert compute their forecast of y_t : $(f_{j,t}$

Result: Aggregation forecast : $\widehat{y}_t$:

$$\widehat{y}_t = \sum_{j=1}^{N} p_{j,t} f_{j,t}$$

where p_t is in $\mathbb{P} \in \mathbb{R}^N$, ie :

$$\sum_{j=1}^{N} p_{j,t} = 1; \; p_{j,t} \geq 0$$

end

Regret

To quantify the performance of our aggregation we need a reference. We thus define the notion of **regret**. Let denote $L_T(\boldsymbol{S}) = \sum_{t=1}^{T} l(y_t, \widehat{y}_t)$ the cumulative loss over $t = 1, ..., T$ of an aggregation strategy $\boldsymbol{S}$.

The regret of $\boldsymbol{S}$ with respect to q (weighted average with fixed weights q) after T sucessive forecast is

$$R_T^{conv}(\boldsymbol{S}) = \widehat{L}_T(\boldsymbol{S}) - \min_{q \in \mathbb{P}} L_T(q)$$

and the regret of $\boldsymbol{S}$ with respect to the best expert is :

$$R_T^{best}(\boldsymbol{S}) = \widehat{L}_T(\boldsymbol{S}) - \min_{j \in 1,\dots,N} L_T(\delta_j)$$

where δ_j is the aggregation strategy consisting in selecting the expert j at each round.

EWA

Algorithm: Exponentially Weighted Aggregation (EWA)

Initialisation : **initial weights** $p_{j,1} = 1/N$**, expert forecasts** $f_{j,1}$ **and aggregation** $\widehat{y}_1 = \sum_{j=1}^{N} p_{j,1} f_{j,1}$

Data: sequential target y_t and N experts $f_{j,t}$

Parameter: loss $l : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}_+$, $\eta > 0$

for t=2,...T do

- compute weights :

$$p_{j,t} = \frac{e^{-\eta L_{t-1}(\delta_j)}}{\sum_{q=1}^{N} e^{-\eta L_{t-1}(\delta_q)}}$$

- then aggregation :

$$\widehat{y}_t = \sum_{j=1}^{N} p_{j,t} f_{j,t}$$

end

EWA Oracle bound

This algorithm and its associated aggregation strategy $\boldsymbol{E}_\eta$ achieves the following oracle bound on the regret.
We suppose that :

- $l : \mathbb{X} \times \mathbb{Y} \to [0, M]$ is upper bounded
- l is convex in its first argument

Then $\forall \eta$,

$$\sup \widehat{L}_T(\boldsymbol{E}_\eta) - \min_j L_T(\delta_j) \leq \frac{\ln(N)}{\eta} + \frac{\eta M^2}{8} T$$

The supremum is taken over all the possible trajectories of y and experts. If we choose $\eta = 1/M\sqrt{8\ln(N)/T}$ minimizing the bound, we have :

$$\sup \widehat{L}_T(\boldsymbol{E}_\eta) - \min_j L_T(\delta_j) \leq M\sqrt{\frac{T}{2}\ln(N)}$$

see [Sto10] for a proof.

EG

Aggregation strategy $\boldsymbol{E}_\eta$ compete with the best expert. We could ambition to compete with e.g. the best convex aggregation.
If we suppose that $l(.,y)$ is differentiable on $\mathbb{X}$, $\forall y \in \mathbb{Y}$ then :
$\forall y$, $\exists\, \partial(.,y)$ a gradient of l such that $\forall\ (u,v) \in \mathbb{X}$:

$$l(u,y) - l(v,y) \leq \partial(u,v).(u-v)$$

and we denote $\tilde{l}_{j,t} = \partial l(\sum_{k=1}^{N} p_{j,t} f_{j,t}, y_t) f_{j,t}$ et $\tilde{L}_T(\delta_j) = \sum_{t=1}^{T} \tilde{l}_{j,t}$.
That allows us to introduce an new algorithm : **Exponential Gradient**.

EG

Algorithm: Exponentially Gradient (EWA)

Initialisation : **initial weights** $p_{j,1} = 1/N$**, expert forecasts** $f_{j,1}$ **and aggregation** $\widehat{y}_1 = \sum_{j=1}^{N} p_{j,1} f_{j,1}$

Data: sequential target y_t and N experts $f_{j,t}$

Parameter: loss $l : \mathbb{X} \times \mathbb{Y} \to \mathbb{R}_+$, $\eta > 0$

for t=2,...T do

- compute weights :

$$p_{j,t} = \frac{e^{-\eta \tilde{L}_{t-1}(\delta_j)}}{\sum_{q=1}^{N} e^{-\eta \tilde{L}_{t-1}(\delta_q)}}$$

- then aggregation :

$$\widehat{y}_t = \sum_{j=1}^{N} p_{j,t} f_{j,t}$$

end

EG oracle bound

This algorithm and its associated aggregation strategy $\boldsymbol{E}_\eta$ achieves the following oracle bound on the regret.

$$\sup \widehat{L}_T(\boldsymbol{E}_\eta^{grad}) - \min_q L_T(q) \leq \frac{\ln(N)}{\eta} + \frac{\eta C^2}{2} T$$

pseudo-loss $\tilde{l}_{j,t} \in]-C, C[$.

Opera

All these algorithms and more are implemented in the R package `opera`.

opera: Online Prediction by Expert Aggregation

Misc methods to form online predictions, for regression-oriented time-series, by combining a finite set of forecasts provided by the user. See Cesa-Bianchi and Lugosi (2006) <doi:10.1017/CBO9780511546921> for an overview.

Version:	1.2.0
Depends:	R (≥ 3.5.0)
Imports:	Rcpp, htmltools, rAmCharts, htmlwidgets, pipeR, alabama, methods, Rdpack
LinkingTo:	Rcpp, RcppEigen
Suggests:	quantreg, quadprog, RColorBrewer, testthat, splines, caret, mgcv, survival, knitr, gbm, rmarkdown, magrittr
Published:	2021-12-06
Author:	Pierre Gaillard [cre, aut], Yannig Goude [aut], Laurent Plagne [ctb], Thibaut Dubois [ctb], Benoit Thieurmel [ctb]
Maintainer:	Pierre Gaillard <pierre at gaillard.me>
BugReports:	https://github.com/dralliag/opera/issues
License:	LGPL-2 | LGPL-2.1 | LGPL-3 [expanded from: LGPL]
Copyright:	EDF R&D 2012-2015
URL:	http://pierre.gaillard.me/opera.html
NeedsCompilation:	yes
Materials:	README NEWS
In views:	TimeSeries
CRAN checks:	opera results

Documentation:

Reference manual:	opera.pdf
Vignettes:	'opera' package "Hierarchical Forecasting with 'opera'

Opera Vignettes

```
library( opera )
vignette("opera-vignette")
vignette("regional forecasting")
```

How to choose the experts ?

as in the ensemble methods in general, diversity matters :

$$(y_t - \widehat{y}_t)^2 = \frac{1}{K}\sum_{k=1}^{K}(y_t - x_{k,t})^2 - \frac{1}{K}\sum_{k=1}^{K}(x_{k,t} - \widehat{y}_t)^2$$

Diversity can be obtained :

- use diverse data sets : data split, bagging, covariates, spatial/temporal resolution
- use different ML methods/algorithms : statistical model (GLM, GAM, kernel based...), ML (forest, boosting, deep learning, stacking)...
- various loss functions : quantile loss, L2, L1...

References I

Casper Solheim Bojer and Jens Peder Meldgaard. « Kaggle forecasting competitions: An overlooked learning opportunity ». In: **International Journal of Forecasting** 37.2 (2021), pp. 587–603.

Leo Breiman. « Bagging predictors ». In: **Machine learning** 24.2 (1996), pp. 123–140.

Sylvain Arlot and Robin Genuer. « Analysis of purely random forests bias ». In: **arXiv preprint arXiv:1407.3939** (2014).

Pierre Geurts, Damien Ernst, and Louis Wehenkel. « Extremely randomized trees ». In: **Machine learning** 63.1 (2006), pp. 3–42.

Fei Tony Liu, Kai Ming Ting, and Wei Fan. « Maximizing tree diversity by building complete-random decision trees ». In: **Pacific-Asia Conference on Knowledge Discovery and Data Mining**. Springer. 2005, pp. 605–610.

Robert E Schapire. « The strength of weak learnability ». In: **Machine learning** 5.2 (1990), pp. 197–227.

Yoav Freund, Robert E Schapire, et al. « Experiments with a new boosting algorithm ». In: **icml**. Vol. 96. Citeseer. 1996, pp. 148–156.

James Hannan. « Approximation to Bayes risk in repeated play ». In: **Contributions to the Theory of Games** 3 (1957), pp. 97–139.

David Blackwell et al. « An analog of the minimax theorem for vector payoffs. » In: **Pacific Journal of Mathematics** 6.1 (1956), pp. 1–8.

References II

Volodimir G Vovk. « Aggregating strategies ». In: **Proc. of Computational Learning Theory, 1990** (1990).

Nicolo Cesa-Bianchi and Gábor Lugosi. **Prediction, learning, and games**. Cambridge university press, 2006.

Gilles Stoltz. « Agrégation séquentielle de prédicteurs: méthodologie générale et applications à la prévision de la qualité de l'air et à celle de la consommation électrique ». In: **Journal de la Société française de Statistique** 151.2 (2010), pp. 66–106.