13 Multiple Learners

Given a problem there is no a best way to solve it. The general idea is to instead of training a complex model/learner, TRAIN MANY MODELS/LEARNERS and combine their results.

You take all the models, you consider all of them. Whenever a new input appears you give it to the models and you combine the various predictions.

There are two different families of approaches:

1. $\boxed{\text{PARALLEL}}$ (voting or bagging)
2. $\boxed{\text{SEQUENTIAL}}$ (boosting)

- VOTING

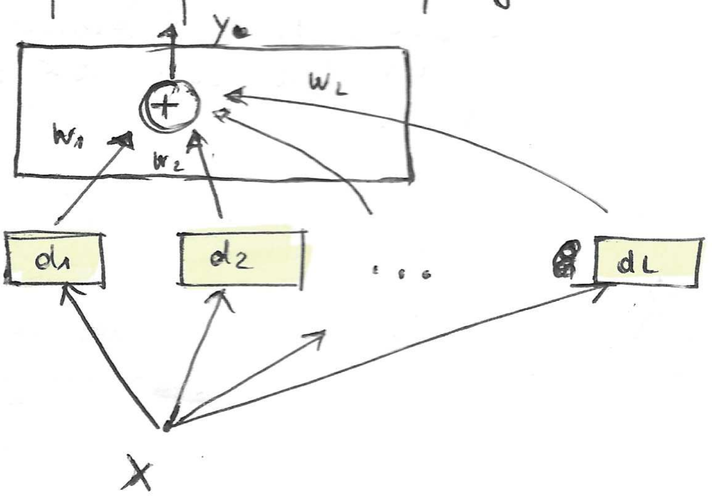
Given a dataset D, you use the same dataset to train different models. Then the prediction can be done by considering A WEIGHTED AVG OF THE PREDICTIONS of EACH MODEL.

(REGR.) $$y_{voting}(x) = \sum_{m=1}^{M} w_m y_m(x)$$

M = # classifiers/learners/models.

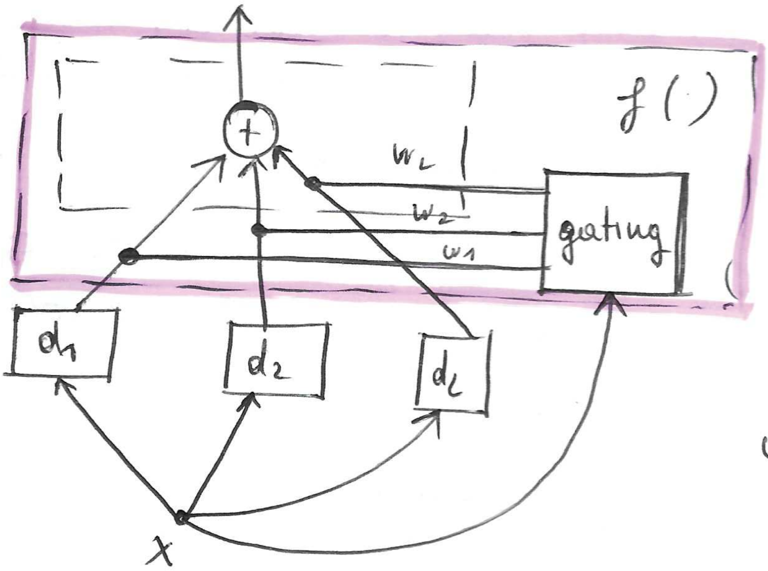
(CLASS.) $$y_{voting}(x) = \arg\max_{c} \sum_{m=1}^{M} w_m \delta(y_m(x) = c)$$

w_m = prior probability of models, $\sum w_m = 1$



The scheme. In the voting scheme the weights are fixed, but we have different approach where the weights might depend on the input like in MIXTURE SCHEME.

MIXTURE OF EXPERTS

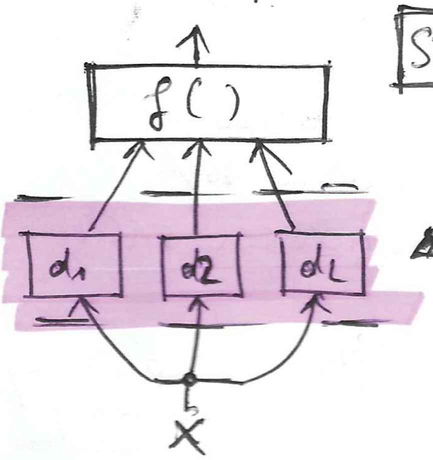


The weights depend on the input, for each input you have different w_i according to the gating function.

This is useful when a subspace can be represented with a particular method.

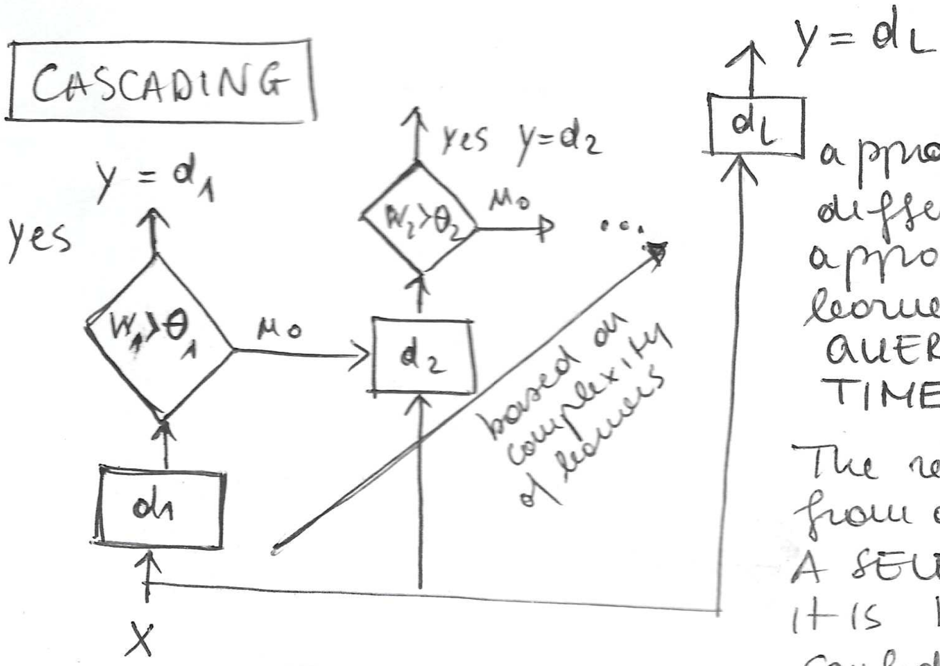
In general you have that the output of the multiple learns are combined in a function whose parameters are learned during the process (seems/reminds a neural network).

STACKING



in a given layer we don't have omogeneous units like in NNs, but eterogeneous units.

CASCADING



The output in this approach is used in a different way. In all befor approaches, you query all learners, IN CASCADING YOU QUERY ONE LEARNER AT TIME.

The result can come just from one model: THIS A SELECTION SCHEME. it is based on threshold, confidence.

BAGGING

You can generate subsets of the dataset and train each model with a different subset. I take my dataset and I generate a set of sub-set of the dataset (You can even use bagging with same learners). In general, this is better than training any individual model.

$$\boxed{Y_{bagging}(x) = \frac{1}{M} \sum_{m=1}^{M} y_m(x)}$$

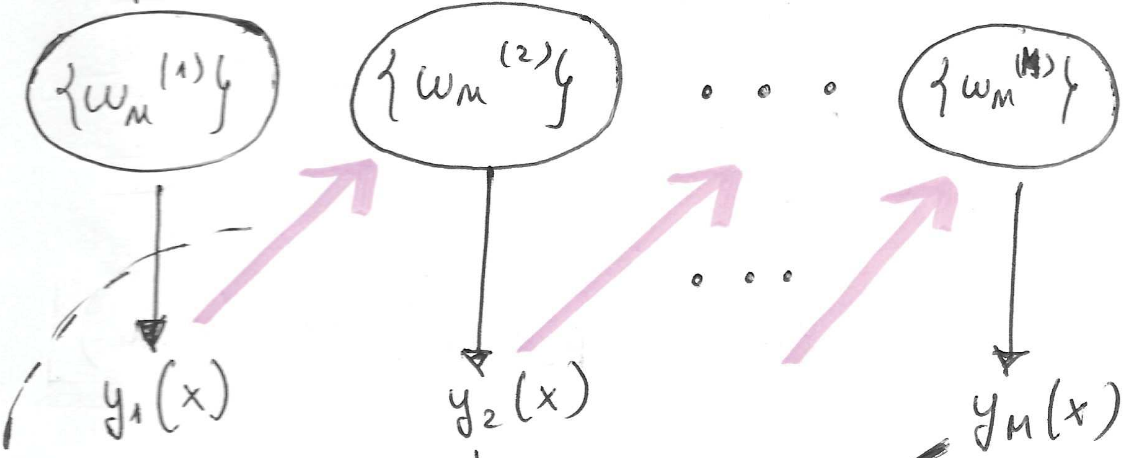
← do the average

To generate Bootstrap data sets, you can USE RANDOM SAMPLING WITH REPLACEMENT.

BOOSTING or sequential approach

More efficient, better!

In a sequential approach, each model is ~~[illegible]~~ trained on a dataset that depends on the training of previous iteration.



$$Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$$

The generation of the second dataset depends on the other training

Base classifiers are trained in sequence using a weighted data set where weights are based on the performance of previous classifiers. WE TRY TO SPECIALIZE LEARNERS, you give more importance (in the next phase training) to misclassified data by previous learner.

Ada Boost Algorithm

Given $D = \{(x_i, t_i)_{i=1}^{N}\}$, where $x_n \in X$, $t_n \in \{+1, -1\}$

1. Initialize $w_n^{(1)} = 1/N$, $n = 1, \ldots, N$.
2. For $m = 1, \ldots, M$

   any learner better than random (accuracy ≥ 0.51)

   Train a weak learner $y_m(x)$ by minimizing the weighted error function:

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)$$

WEIGHTED ERROR FUNCTION

where $I(e) = \begin{cases} 1 & \text{if } e \text{ true} \\ 0 & \text{otherwise} \end{cases}$

Evaluate: $\varepsilon_m = \dfrac{\sum_{n=1}^{N} w_n^{(m)} I(y_m(x_n) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$

RESIDUAL ERROR

how good has been the classifier

and $\alpha_m = \ln\left[\dfrac{1 - \varepsilon_m}{\varepsilon_m}\right]$

update $w_n^{(m+1)} = w_n^{(m)} \exp\left[\alpha_m I(y_m(x_n) \neq t_n)\right]$

3. Output: the final classifier

$$Y_M(\vec{x}) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right)$$

AdaBoost IS OPTIMIZING AN EXPONENTIAL ERROR FUNCTION. Consider the error function:

$$E = \sum_{n=1}^{N} \exp\left[-t_n f_M(x_n)\right]$$

Where

$$f_M(x) = \frac{1}{2} \sum_{i=1}^{M} \alpha_m y_m(x) \quad , \quad t_n \in \{-1, +1\}$$

GOAL: minimize E wrt $\alpha_m, y_m(x)$
$\forall\ m = 1, \dots, M$.

Sequential minimization: Instead of minimizing E globally

- assume $y_1(x), \dots, y_{M-1}(x)$ and $\alpha_1, \dots, \alpha_{M-1}$ fixed
- minimize wrt α_M and $y_M(x)$

The idea is to consider at each step, some parameters fixed

Making $y_M(x)$ and α_M explicit:

$$E = \sum_{i=1}^{N} \exp\left[-t_n f_{M-1}(x_n) - \frac{1}{2} t_n \alpha_M y_M(x_n)\right]$$

$$= \sum_{n=1}^{N} w_n^{(M)} \exp\left[-\frac{1}{2} t_n \alpha_M y_M(x_n)\right]$$

with $w_n^{(M)} = \exp\left[-t_n f_{M-1}(x_n)\right]$ constant since we are optimizing wrt α_M and $y_M(x)$.

From sequential minimization of E we obtain:

$$w_n^{(m+1)} = w_n^{(m)} \exp\left[\alpha_m I\left(y_m(x_n) \neq t_n\right)\right]$$

predictions are made with:

$$\text{sign}\left(f_M(x)\right) = \text{sign}\left(\frac{1}{2} \sum_{m=1}^{M} \alpha_m y_m(x)\right)$$

which is equivalent to:

$$Y_M(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m y_m(x)\right) \quad \square$$

WITH THE ASSUMPTION OF OPTIMIZING WRT α_M and y_M WE SIMPLIFY A LOT THE PROBLEM.

(+) FAST, simple and easy to program.

(+) No prior requirements to learners (best than random)

(+) No parameter to tune, except for M. A theoretical result says that performance improve with an higher M

(+) practical evidence to be good (better performance wrt to individual learners).

(−) Performance depends on data and the base learners (can fail with insufficient data or when base learners are too weak)

(−) sensitive to noise.