Machine learning

Ensemble methods

Corso di Laurea Magistrale in Informatica

Università di Roma Tor Vergata

Prof. Giorgio Gambosi

a.a. 2021-2022

Spesso fanno riferimento a decision tree.

Un albero su un solo livello, quindi con un solo taglio, classifica comunque meglio di un classificatore random.

Un weak learner è un learner che impara in aìmaniera debole a classificare.

Quando usiamo un classificatore, ci basiamo su cosa ci dice. Possiamo costrurire k classificatori dello stesso tipo ma non identici per classificare un elemento lo diamo a tutti e k e vediamo il voto di maggioranza.
In fully Bayesian ogni voto è pesato, qui invece no e c'è un'altra questione ovvero che spesso questi classificatori sono weak learner.

Quindi predicono su una base molto semplice, metto insieme le predizioni e vedo se quello che ottengo è meglio di un classificatore più complesso.

1) Come li rendiamo diversi fra loro? Istanziandoli in modo diverso
2) Come li istanzio in modo diverso? Do dei training set diversi, visto che ne ho uno posso partizionarlo in dei pezzi ed usare ogni pezzo per trainare ogni singolo predittore. È il BAGGING

Posso anche usare l'insieme di predittori in modo diverso: parto dal primo e vedo come si comporta, deve predirre un valore associato. Per un certo x sbaglia poco, per un x' sbaglia tanto.
Costruisco allora un nuovo predittore sulla base del comoprtamento del primo, in modo che sia più preciso dove il primo sbaglia.

Poi ne faccio un 3° sulla base di 1° e 2°, spinto a classificare bene quelli che i primi 2 classificavano male e procedo così. Ho un insieme di classificatori e vedo se lo classificano bene o male, se questo punto è mal classificato è un punto critico quindi costruisco il nuovo classificatore spinto per classificare bene proprio quello.

Non sono costruiti indipendentemente ma in sequenza, guardando come si comportano quelli costruiti prima, poi li interrogo tutti e vado per votazione ma li faccio sempre in mood che quelli dopo migliorino errori sistematici di quelli prima. È il BOOSTING

Ensemble methods

Improve performance by combining multiple models, in some way, instead of using a single model.

- train a *committee* of L different models and make predictions by averaging the predictions made by each model on dataset samplings (bagging)
- train different models in sequence: the error function used to train a model depend on the performance of previous models (boosting)

→ viene ad essere dipendente da come classifica-
no quelli prima.

Bagging

- Classifiers (especially some of them, such as decision trees) may have low performances due to their high variance: their behavior may largely differ in presence of slightly different training sets (or even of the same training set).
- For example, in trees, the separations made by splits are enforced at all lower levels: hence, if the data is perturbed slightly, the new tree can have a considerably different sequence of splits, leading to a different classification rule

Bootstrap

Campionamento con ripetizione

- The bootstrap is a fundamental resampling tool in statistics. The basic underlying idea is to estimate the true distribution of data $\mathscr{F}$ by the so-called empirical distribution $\hat{\mathscr{F}}$
- Given the training data $(\mathbf{x}_i, t_i)$, $i = 1, \dots, n$, the empirical distribution function $\hat{\mathscr{F}}$ is defined as

$$\hat{p}(\mathbf{x}, t) = \begin{cases} \frac{1}{n} & \text{if } \exists i \, : \, (\mathbf{x}, t) = (\mathbf{x}_i, t_i) \\ 0 & \text{otherwise} \end{cases}$$

- This is just a discrete probability distribution, putting equal weight $\frac{1}{n}$ on each of the observed training points

Devo prendere i k elementi per il training set del primo classficatore, tolti questi non li userò più.
In questo caso l'idea di riusarli di nuovo, quindi non sono più limtato a k elementi come prima (perché ero limitato).

Supponiamo che ci sia una distribuzione degli elementi socnosicuta che è quella della popolazione, conosciamo solo "l'immagine" che è il training set

Bootstrap

estraggo elementi dalla distribuzione empirica

- A bootstrap sample of size m from the training data is

$$(\mathbf{x}_i^*, t_i^*) \qquad i = 1, \dots, m$$

  where each $(\mathbf{x}_i^*, t_i^*)$ is drawn uniformly at random from $(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_n, t_n)$, with replacement

- This corresponds exactly to m independent draws from $\hat{\mathscr{F}}$: it approximates what we would see if we could sample more data from the true $\mathscr{F}$. We often consider $m = n$, which is like sampling an entirely new training set

Bagging

- Given a training set $(\mathbf{x}_i, y_i)$, $i = 1, \dots, n$, bagging averages the predictions done by classifiers of the same type (such as decision trees) over a collection of boostrap samples. For $b = 1, \dots, B$ (e.g., B = 100), n bootstrap items $(\mathbf{x}_i^b, y_i^b)$, $i = 1, \dots, n$ are sampled and a classifier is fit on this set.
- At the end, to classify an input x, we simply take the most commonly predicted class, among all B classifiers
- This is just choosing the class with the most votes
- In the case of regression, the predicted value is derived as the average among the predictions returned by the B regressors

Bagging variant

If the used classifier returns class probabilities $\hat{p}_k^b(\mathbf{x})$, the final bagged probabilities can be computed by averaging

$$p_k^b(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{p}_k^b(\mathbf{x})$$

the predicted class is, again, the one with highest probability

Bagging classification

- Why is bagging working?
- Let us consider, for simplicity, a binary classification problem. Suppose that for a given input $\mathbf{x}$, we have B independent classifiers, each with a given misclassification rate e (for example, $e = 0.4$). Assume w.l.o.g. that the true class at $\mathbf{x}$ is 1: so the probability that the b-th classifier predicts class 0 is $e = 0.4$
- Let $B_0 \leq B$ be the number of classifiers returning class 0 on input $\mathbf{x}$: the probability of B_0 is clearly distributed according to a binomial (if classifiers are independent)

$$B_0 \sim \text{Binomial}(B, e)$$

the misclassification rate of the bagged classifier is then

$$p\left(B_0 > \frac{B}{2}\right) = \sum_{k=\frac{B}{2}+1}^{B} \binom{B}{k} e^k (1-e)^{B-k}$$

which tends to 0 as B increases.

Bagging regression

- Expected error of one model $y_i(\mathbf{x})$ wrt the true function $h(\mathbf{x})$:

$$E_{\mathbf{x}}[(y_i(\mathbf{x}) - h(\mathbf{x}))^2] = E_{\mathbf{x}}[\varepsilon_i(\mathbf{x})^2]$$

- Average expected error of the models

$$E_{av} = \frac{1}{m}\sum_{i=1}^{m} E_{\mathbf{x}}[\varepsilon_i(\mathbf{x})^2]$$

- Committee expected error

$$E_c = E_{\mathbf{x}}\left[\left(\frac{1}{m}\sum_{i=1}^{m} y_i(\mathbf{x}) - h(\mathbf{x})\right)^2\right] = E_{\mathbf{x}}\left[\left(\frac{1}{m}\sum_{i=1}^{m} \varepsilon_i(\mathbf{x})\right)^2\right]$$

  If $E_{\mathbf{x}}[\varepsilon_i(\mathbf{x})\varepsilon_j(\mathbf{x})] = 0$ if $i \neq j$ (errors are uncorrelated) then $E_c = \frac{1}{m}E_{av}$.

- This is usually not verified: errors from different models are highly correlated.

Out-of-bag error

- Model evaluation can be performed by evaluating, for each item $\mathbf{x}_i$ in the data set, the prediction done by the set of models trained on bootstrap samples not including $\mathbf{x}_i$.
- If bootstrap samples have the same size of the dataset (i.e. $m = n$), there is a probability $.63$ that an item is included in a bootstrap sample: in fact, for each sample, the probability that item $\mathbf{x}_i$ is not selected is $1 - \frac{1}{n}$. Hence there is a probability $\left(1 - \frac{1}{n}\right)^n$ that it is never sampled. For large enough values of n, the probability is about $\lim_{n\to\infty}\left(1 - \frac{1}{n}\right)^n = \frac{1}{e} \approx .37$
- In out-of-bag evaluation, the prediction of an item is done by using approximately a fraction $.37$ of all the trees. For those trees the item can be considered as a test set member.

Random forest

Uno dei metodi più usati in ML. Il classificatore base è l'albero di decisione.

Application of bagging to a set of (random) decision trees: classification performed by voting.

1. For $b = 1$ to B:
   - 1.1 Bootstrap sample from training set
   - 1.2 Grow a decision tree T_b on such data by performing the following operations for each node:
     - 1.2.1 select m variables at random
     - 1.2.2 pick the best variable among them
     - 1.2.3 split the node into two children
2. output the collection of trees $T_1, \dots, T_B$

modo con cui si rendono diversi gli alberi.

Overall prediction is performed as majority (for classification) or average (for regression) among trees predictions.

Si introduce un altro livello di randomizzazione

Boosting

Costruiamo sempre un insieme di classificatori, ma in modo sequenziale: predittore K+1 e' costruito vedendo quanto bene si comporta il K.

- Boosting is a procedure to combine the output of many weak classifiers to produce a powerful $\overset{\text{insieme}}{\text{committee}}$.
- A weak classifier is one whose error rate is only slightly better than random guessing.
- Boosting produces a sequence of weak classifiers $y_{\overset{j}{m}}(x)$ for $\overset{j}{m} = 1, \dots, m$ whose predictions are then combined through a weighted majority to produce the final prediction

$$y(\mathbf{x}) = \underline{\operatorname{sgn}} \left(\sum_{j=1}^{m} \alpha_j y_j(\mathbf{x}) \right)$$

Consideriamo classificazione binaria, ogni classificatore da 1, -1.

- Each $\alpha_j > 0$ is computed by the boosting algorithm and reflects how accurately $y_{\overset{j}{m}}$ classified the data.

Può succedere che contestualmente venga definito un peso per i classificatori, questo e' misura di affidabilità. La predizione finale e' composizione pesata delle m predizioni.

Boosting

molteplicità dell'elemento aumenta (presente più di 1 volta)

Adaboost (adaptive boosting)

- Models are trained in sequence: each model is trained using a weighted form of the dataset
- Element weights depend on the performances of the previous models (misclassified points receive larger weights)
- Predictions are performed through a weighted majority voting scheme on all models

Peso associato ad un elemento è più alto se modelli precedenti l'hanno classificato male.
Il peso di quelli che il classificatore corrente classifica male aumenta ⇒ nuovo data set con pesi diversi e costruiamo un nuovo classificatore.

diminuisce se classificati bene.

Boosting

1° assegnazione dei pesi agli elementi, sarà 1 $\longrightarrow$

M fissato prima.

$$\begin{tikzcd}
\{w_n^{(1)}\} \arrow[d] & \{w_n^{(2)}\} \arrow[d] & \cdots & \{w_n^{(M)}\} \arrow[d] \\
y_1(\mathbf{x}) \arrow[ur] \arrow[dr] & y_2(\mathbf{x}) \arrow[ur] \arrow[d] & \cdots \arrow[ur] & y_M(\mathbf{x}) \arrow[dll] \\
& Y_M(\mathbf{x}) & &
\end{tikzcd}$$

1° classificatore $\longrightarrow$ $y_1(\mathbf{x})$

abbiamo anche α_1

α_2

α_M

$$\boxed{Y_M(\mathbf{x}) = \operatorname{sign}\left(\sum_{m}^{M} \alpha_m y_m(\mathbf{x})\right)}$$

$\Uparrow$

Nel bagging, $y_i(\mathbf{x})$ sarebbero stati derivati in maniera indipendente

Adaboost fornisce $y_i(\mathbf{x})$ ed α_i, la predizione si effettua così

Concettualmente: costruisco un classificatore, vedo dove sbaglia e ne costruisco un altro che cerca di compensare dove il primo sbaglia e la predizione è data sommando le due.
Si vedono poi 1° e 2° e se ne mette un 3° che cerca di compensare dove questi due sbagliano

Costruire "a mano" una sequenza di classificatori che via via cerchino di correggere le situazioni in cui quelli di prima si comportano male

La predizione è una somma, si chiamano infatti modelli additivi.

È cosa accade con i metodi di discesa del gradiente, dove modifichiamo i parametri del modello effettuando una serie di somme fino al valore che è quello che fa si che la predizione vada meglio.
Una serie di somme successive modificano il comportamento precedente per migliorare quello successivo.

Adaboost

Binary classification, dataset $(\mathbf{X}, \mathbf{t})$ of size n, with $t_i \in \{-1, 1\}$. The algorithm maintains a set of weights $w(\mathbf{x}) = (w_1, \dots, w_n)$ associated to the dataset elements.

- Initialize weights as $w_i^{(0)} = \frac{1}{n}$ for $i = 1, \dots, n$ ← iniziamente il dataset non e' pesato.
- For $j = 1, \dots, m$:
  - Train a weak learner $y_j(\mathbf{x})$ on $\mathbf{X}$ in such a way to minimize the weighted misclassification wrt to $w^{(j)}(\mathbf{x})$. → es: albero
  - Let

vorremmo che fosse basso →
$$e^{(j)} = \frac{\sum_{\mathbf{x}_i \in \mathcal{E}^{(j)}} w_i^{(j)}}{\sum_i w_i^{(j)}}$$
← peso degli elementi classificati male
← peso di tutti gli elementi

where $\mathcal{E}^{(j)}$ is the set of dataset elements misclassified by $y_j(\mathbf{x})$.

- If $e^{(j)} > \frac{1}{2}$, consider the reverse learner, which returns opposite predictions for all elements.
- $e^{(j)}$ can be interpreted as the probability that a random item from the training set is misclassified, assuming that item $\mathbf{x}_i$ can be sampled with probability $\frac{w_i^{(j)}}{\sum_i w_i^{(j)}}$ ← somma delle prob. degli elementi classificati male.

va talmente male che conviene scambiare le classi

Adaboost

rapporto di probabilità.

- Compute the learner confidence as log odds of a random item being well classified $(1 - e^{(j)})$ vs being misclassified $e^{(j)}$

$$\alpha_j = \frac{1}{2} \log \frac{1 - e^{(j)}}{e^{(j)}} > 0$$

$1 - e^{(j)} > e^{(j)}$ altrimenti scambio le classi.

- For each $\mathbf{x}_i$, update the corresponding weight as follows

$$w_i^{(j+1)} = w_i^{(j)} e^{-\alpha_j t_i y_j(\mathbf{x}_i)}$$

$t_i = -1, 1$ $\quad y_j(\mathbf{x}_i) = -1, 1 \Rightarrow$ prodotto e' positivo se ci azzecca. Esponente quindi e' negativo se ci prende, positivo altrimenti.

which results into

$$w_i^{(j+1)} = \begin{cases} w_i^{(j)} e^{\alpha_j} > w_i^{(j)} & \text{if } \mathbf{x}_i \in \mathcal{E}^{(j)} \\ w_i^{(j)} e^{-\alpha_j} < w_i^{(j)} & \text{otherwise} \end{cases}$$

- Normalize the set of $w_i^{(j+1)}$ by dividing each of them by $\sum_{i=1}^n w_i^{(j+1)}$, in order to get a distribution

Adaboost

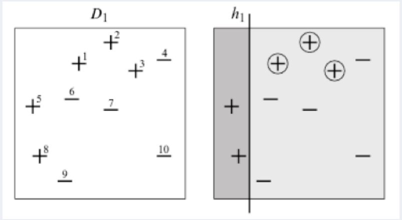
The overall prediction is

$$y(\mathbf{x}) = \text{sgn}\left(\sum_{j=1}^{m} \alpha_j y_j(\mathbf{x})\right)$$

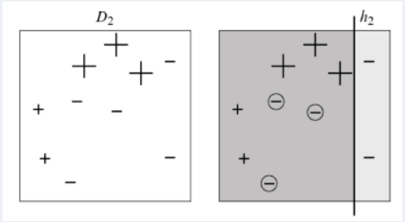
since $y_j(\mathbf{x}) \in \{-1, 1\}$, this corresponds to a voting procedure, where each learner vote (class prediction) is weighted by the learner confidence.

$\hookrightarrow \alpha$ corrispondente.

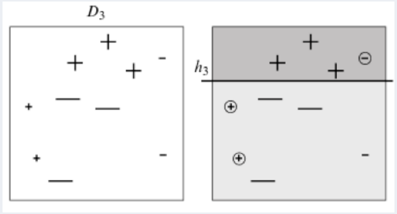
Adaboost



Albero di altezza 1

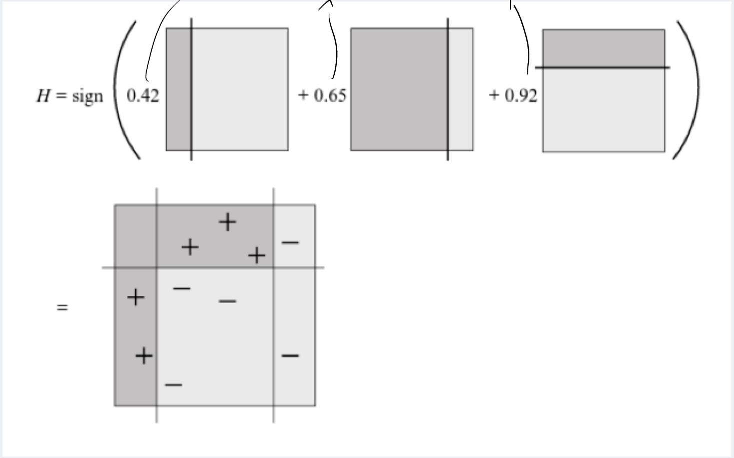


Nuova versione del dataset, nuovo albero



Adaboost

gli α_i



Why does it work?

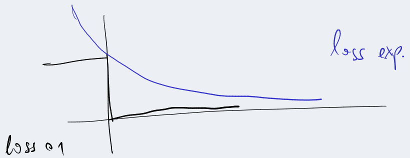
- It minimizes a loss function related to classification error
- Suppose we have a classifier $y(\mathbf{x}) = \operatorname{sgn} f(\mathbf{x})$
- We know that 0/1 loss

$$l(y(\mathbf{x}), t) = \begin{cases} 0 & \text{if } t f(\mathbf{x}) > 0 \\ 1 & \text{otherwise} \end{cases}$$

  has drawbacks (non convex, gradient 0 almost everywhere). We need a surrogate loss.
- Exponential loss

$$l(y(\mathbf{x}), t) = e^{-t f(\mathbf{x})}$$

consideremo questa.



Additive models

- Additive models are defined as the additive composition of simpler "base" predictors

la predizione non e' altro che la somma pesata delle predizioni

$$y(\mathbf{x}) = \boxed{\sum_{j=1}^{m} c_j \overline{y}(\mathbf{x})} \longrightarrow$$

Consideriamo come un modello.

where c_j's are weights and $\overline{y}_j(\mathbf{x}) = \overline{y}(\mathbf{x}; \mathbf{w}) \in \mathbb{R}$ is a simple functions of the input $\mathbf{x}$ parameterized by $\mathbf{w}$

- in this case, the predictors are binary classifiers; that is, $\overline{y}_j(\mathbf{x}) = \overline{y}(\mathbf{x}; \mathbf{w}) \in \{-1, 1\}$

predizione effettuata su x con i coefficienti che caratterizzano quel predittore

Fitting additive models

(rischio empirico)

- As usual, an additive model is fit by minimizing a loss function averaged over the training data:

$$\min_{c_j, \mathbf{w}_j} \sum_{i=1}^{n} L(t_i, \sum_{k=1}^{m} c_k \overline{y}(\mathbf{x}_i; \mathbf{w}_k))$$

predizione che effettua il modello

Sono i coefficienti ed i parametri del modello. Minimizzo rispetto a questi $\forall j$.

- For many loss functions L and/or predictors $\overline{y}$ this is too hard

Assumiamo che i modelli siano fatti tutti allo stesso modo, $\bar{y}$ uguale per tutti (cambiano i coefficienti).

Forward stagewise additive modeling

Cerchiamo prima l'ottimo per (c_1, w_1), poi in base a quello per (c_2, w_2) . . . .

More simply, one can greedily add one predictor at a time in the following fashion.

- Set $y_0(\mathbf{x}) = 0$
- For $k = 1, \dots, m$:
  - Compute

$$(\hat{c}_k, \hat{\mathbf{w}}_k) = \underset{c_k, \mathbf{w}_k}{\operatorname{argmin}} \sum_{i=1}^{n} L(t_i, y_{k-1}(\mathbf{x}_i) + c_k \overline{y}(\mathbf{x}_i; \mathbf{w}_k))$$

Minimizza il rischio empirico della loss sulla predizione che risulta dai precedenti + quella del nuovo predittore con parametri w_k e coefficiente c_k

  - Set $y_k(\mathbf{x}) = y_{k-1}(\mathbf{x}) + \hat{c}_k \overline{y}(\mathbf{x}; \hat{\mathbf{w}}_k)$ $\longrightarrow$ quello di prima + nuovo contributo.

That is, fitting is performed not modifying previously added terms (greedy paradigm)

Faccio sempre il miglior passo avanti rispetto a dove sono arrivato.

Adaboost as additive model

Abbiamo definito il modello additivo con una certa loss L, in Adaboost L e' l'exponential loss

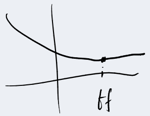
Adaboost can be interpreted as fitting an additive model with exponential loss

$$L(y, f(\mathbf{x})) = e^{-t\boxed{f(\mathbf{x})}} \longrightarrow$$ predizione effettuata, e' il valore su cui poi vediamo il segno $(\in \mathbb{R})$

that is, minimizing

$$\boxed{\sum_{i=1}^{n} e^{-t_i \sum_{k=1}^{m} \alpha_k \underbrace{\bar{y}(\mathbf{x}_i;\mathbf{w}_k)}_{-1,1}}} \longrightarrow \in \mathbb{R}$$

with respect to $\mathbf{w}_1, \dots, \mathbf{w}_m$ and $\alpha_1, \dots, \alpha_m$



Adaboost as additive model

approccio greedy

Applying forward stagewise additive modeling, at each step k we compute

$$
\begin{aligned}
(\hat{\alpha}_k, \hat{\mathbf{w}}_k) &= \underset{\alpha_k, \mathbf{w}_k}{\operatorname{argmin}} \sum_{i=1}^{n} e^{-t_i y(\mathbf{x}_i)} \\
&= \underset{\alpha_k, \mathbf{w}_k}{\operatorname{argmin}} \sum_{i=1}^{n} e^{-t_i (y_{k-1}(\mathbf{x}_i) + \alpha_k \overline{y}(\mathbf{x}_i; \mathbf{w}_k))} \\
&= \underset{\alpha_k, \mathbf{w}_k}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^{(k)} e^{-\alpha_k t_i \overline{y}(\mathbf{x}_i; \mathbf{w}_k)}
\end{aligned}
$$

where $w_i^{(k)} = e^{-t_i y_{k-1}(\mathbf{x}_i)} = e^{-\frac{1}{2} t_i \sum_{r=1}^{k-1} \alpha_r \overline{y}_r(\mathbf{x}_i)}$ is a constant wrt α_k and $\mathbf{w}_k$

The approach can be extended to the case of different loss functions

Find the next learner and related weight

We may decompose the weighted loss function as follows

$$\sum_{i=1}^{n} w_i^{(k)} e^{-\alpha_k t_i \overline{y}_k(\mathbf{x}_i)} = \sum_{i=1}^{n} w_i^{(k)} e^{-\alpha_k t_i \overline{y}(\mathbf{x}_i;\mathbf{w}_k)} = \sum_{\mathbf{x}_i \in \mathscr{E}^{(k)}} w_i^{(k)} e^{\alpha_k} + \sum_{\mathbf{x}_i \notin \mathscr{E}^{(k)}} w_i^{(k)} e^{-\alpha_k}$$

where $\mathscr{E}^{(k)}$ is the set of elements misclassified by $\overline{y}_k$

By adding and subtracting $\sum_{\mathbf{x}_i \in \mathscr{E}^{(k)}} w_i^{(k)} e^{-\alpha_k}$ we obtain the weighted loss function

$$\sum_{\mathbf{x}_i \in \mathscr{E}^{(k)}} w_i^{(k)} (e^{\alpha_k} - e^{-\alpha_k}) + e^{-\alpha_k} \sum_{\mathbf{x}_i} w_i^{(k)}$$

to be minimized wrt $\mathbf{w}^{(k)}$ and α_k

Find the next learner

To derive the best learner coefficients $\mathbf{w}^{(k)}$, we observe that they affect, through $\mathscr{E}^{(k)}$, only the first term

$$\sum_{\mathbf{x}_i \in \mathscr{E}^{(k)}} w_i^{(k)} (e^{\alpha_k} - e^{-\alpha_k}) \longrightarrow$$ costante se consideriamo gli α come costanti

has to be considered, since the other one is constant.

Since α_k is considered as a constant here, we have to minimize the sum of the current weights of misclassified items

$$\sum_{\mathbf{x}_i \in \mathscr{E}^{(k)}} w_i^{(k)}$$

$\uparrow$

wrt $\mathbf{w}^{(k)}$, which is what is done in Adaboost

Cerchiamo il classificatore che minimizzi questa somma, quella dei pesi degli elementi mal classificati

Find the next learner weight

To derive the best learner weight α_k, we need to take into account the whole loss function.

By setting to 0 the derivative of the loss function wrt α_k, we get

$$\alpha_k = \frac{1}{2} \log \frac{1 - e^{(k)}}{e^{(k)}}$$

which again corresponds to what is done in Adaboost

Updating the element weights

Abbiamo trovato il migliore classificatore da aggiungere col relativo peso. Come cambiano i pesi:

By introducing the new learner $\overline{y}_k$ with weight α_k, the overall predictor turn out to be

$$y_k(\mathbf{x}) = y_{k-1}(\mathbf{x}) + \alpha_k \overline{y}_k(\mathbf{x}) = y_{k-1}(\mathbf{x}) + \alpha_k \overline{y}(\mathbf{x}; \mathbf{w}_k)$$

Since by definition $w_i^{(k)} = e^{-t_i y_{k-1}(\mathbf{x}_i)}$ we have for the new weights $w_i^{(k+1)}$

$$w_i^{(k+1)} = e^{-t_i \underline{y_k(\mathbf{x}_i)}} = e^{-t_i(y_{k-1}(\mathbf{x}_i) + \alpha_k \overline{y}(\mathbf{x}_i; \mathbf{w}_k))} = w_i^{(k)} \underline{e^{-t_i \alpha_k \overline{y}(\mathbf{x}_i; \mathbf{w}_k)}}$$

again, as in Adaboost

rispetto ad $y_{k-1}(\mathbf{x})$ c'è un termine in più

Aggiornamento di Adaboost

Gradient boosting

General idea:

- Fit an additive model $\sum_{j=1}^{m} \alpha_j y_j(\mathbf{x})$ in a forward stage-wise manner.
- At each stage, introduce a weak learner to compensate the shortcomings of existing ones.
- Shortcomings are identified by high-weight data points.

Gradient boosting

Nel caso specifico della regressione viene naturale: aggiungo un regressore che ha nel training set le stesse feature ma il target sarà il residuo (l'errore) del 1°. La predizione sarà la somma.

- You are given $(\mathbf{x}_i, t_i)$, $i = 1, \dots, n$, and the task is to fit a model $y(\mathbf{x})$ to minimize square loss.
- Assume a model $y^{(1)}(\mathbf{x})$ is available, with residuals $t_i - y^{(1)}(\mathbf{x}_i) = t_i - y_i^{(1)}$
- A new dataset $(\mathbf{x}_i, t_i - y_i^{(1)})$, $i = 1, \dots, n$ can be defined, and a model $\overline{y}^{(1)}(\mathbf{x})$ can be fit to minimize square loss wrt such dataset
- Clearly, $y_2(\mathbf{x}) = y_1(\mathbf{x}) + \overline{y}_1(\mathbf{x})$ is a model which improves $y_1(\mathbf{x})$
- The role of $\overline{y}_1(\mathbf{x})$ is to compensate the shortcoming of $y(\mathbf{x})$
- If $y_2(\mathbf{x})$ is unsatisfactory, we may define new models $\overline{y}_2(\mathbf{x})$ and $y_3(\mathbf{x}) = y_2(\mathbf{x}) + \overline{y}_2(\mathbf{x})$

Posso andare avanti finché ho del residuo da correggere. C'è l'effetto cumulativo dei modelli additivi.

Gradient boosting

How is this related to gradient descent?

- Let us consider the squared loss function $L(t, y) = \frac{1}{2}(t - y)^2$
- We want to minimize the $\overset{\text{empirico}}{\text{risk}}$ $R = \sum_{i=1}^{n} L(t_i, y_i)$ by adjusting $y_i, \dots, y_n$
- Consider y_i as parameters and take derivatives

$$\frac{\partial R}{\partial y_i} = y_i - t_i$$

So, we can consider residuals as negative gradients

$$t_i - y_i = -\frac{\partial R}{\partial y_i}$$

- Model $\overline{y}(\mathbf{x})$ can then be derived by considering the dataset

$$(\mathbf{x}_i, t_i - y_i) = \left(\mathbf{x}_i, -\frac{\partial R}{\partial y_i}\right) \qquad i = 1, \dots, n$$

al posto del target, metto il gradiente del rischio rispetto al valore calcolato.

Concetto generale, aggiungo un "pezzetto" che minimizza la f. di loss.

Vogliamo sempre minimizzare il rischio:

- prima discesa del gradiente rispetto ai singoli parametri

- qui, ho i modelli precedenti che danno predizioni. Vediamo come modificare la predizione fatta dai modelli per minimizzare il rischio

Gradient boosting for regression

Modello restituisce un valore, aggiungo un "pezzetto" che minimizzi la loss che dipende da y_i e che posso combinare.

The following algorithm results

- Set $y^{(1)}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} t_i$
- For $k = 1, \dots, m$:
  - Compute negative gradients

$$-g_i^{(k)} = -\frac{\partial R}{\partial y_i}\Big|_{y_i = y^{(k)}(\mathbf{x}_i)} = -\frac{\partial}{\partial y_i} L(t_i, y_i)\Big|_{y_i = y^{(k)}(\mathbf{x}_i)} = t_i - y^{(k)}(\mathbf{x}_i)$$

  - Fit a weak learner $\overline{y}^{(k)}(\mathbf{x})$ to negative gradients, considering dataset $(\mathbf{x}_i, -g_i^{(k)}), i = 1, \dots, n$
  - Derive the new classifier $y^{(k+1)}(\mathbf{x}) = y^{(k)}(\mathbf{x}) + \overline{y}^{(k)}(\mathbf{x})$

Gradient boosting for regression

- The benefit of formulating this algorithm using gradients is that it allows us to consider other loss functions and derive the corresponding algorithms in the same way.
- For example, square loss is easy to deal with mathematically, but not robust to outliers, i.e. pays too much attention to outliers.
- Different loss functions
  - Absolute loss
    - $L(t, y) = |t - y|$
    - $-g = \text{sgn}(t - y)$
  - Huber loss
    - $$L(t, y) = \begin{cases} \frac{1}{2}(t-y)^2 & |t-y| \leq \delta \\ \delta(|t-y|) - \frac{\delta}{2} & |t-y| > \delta \end{cases}$$
    - $$-g = \begin{cases} y - t & |t-y| \leq \delta \\ \delta \cdot \text{sgn}(t-y) & |t-y| > \delta \end{cases}$$

Pratica: ha molti parametri ed iper-parametri definibili quindi model selection e' laboriosa.

In pratica e' un metodo di classificazione che funziona bene in piccola.

Which weak learners?

- Regression trees (special case of decision trees)
- Decision stumps (trees with only one node)