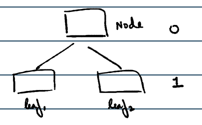
AdaBoost

Sunday 5 May 2024 2:40 PM

→ Stage wise Additive method

→ Weak learners (High Bias low variance)

→ Decision stumps



→ ~~1~~/0 $\overset{(+ve)}{1}$/$\overset{(-ve)}{-1}$

→ In Random Forest each tree has an equal vote on the final prediction but in a Forest of stumps made with AdaBoost, some stumps get more say in the final prediction than others.

→ In Random Forest each DT is made independently of each other. But, in a Forest of stumps made with AdaBoost, order is important. The errors that the first stump makes influence how the second stump is made and so on....

→ Trees are fit sequentially to improve error of previous trees

Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.

Algorithm

Data

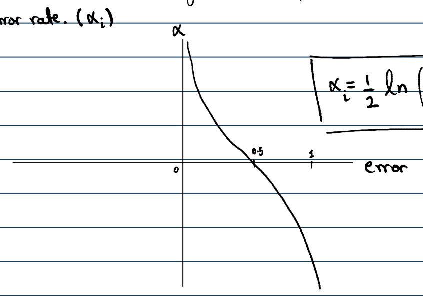
x_1	x_2	y
1	5	1
2	4	0
9	2	1
1	7	0
2	3	1

1) assign initial weight to each row.

weight = $\frac{1}{n}$ ← total # of rows

x_1	x_2	y	wt
1	5	1	1/5
2	4	0	1/5
9	2	1	1/5
1	7	0	1/5
2	3	1	1/5

2) train a Decision stump
- check performance on data [calc predictions]
- calculate model's weight, which depends on the error rate. (α_i)



$$\boxed{\alpha_i = \frac{1}{2} \ln\left(\frac{1-\text{error}}{\text{error}}\right)}$$

if error is 1 or 0 equation will panic add a very small number.

x_1	x_2	y	wt	$\hat{y}$
1	5	1	1/5	0
2	4	0	1/5	1
9	2	1	1/5	1
1	7	0	1/5	0
2	3	1	1/5	1

→ error = sum of weights of misclassified points

error = $\frac{2}{5}$ $\alpha_1 = \frac{1}{2} \ln\left(\frac{1 - 2/5}{2/5}\right) \approx 0.2$

3) Update weights [increasing the importance of misclassified points
decreasing the " " correctly classified "]

<u>For misclassified</u>

new_wt = curr_wt x e^{α_i}

<u>For correctly classified</u>

new_wt = curr_wt x $e^{-\alpha_i}$

x_1	x_2	y	wt	$\hat{y}$	new_wt	normalized
1	5	1	1/5	0	0.24	0.25
2	4	0	1/5	1	0.24	0.25
9	2	1	1/5	1	0.16	0.166
1	7	0	1/5	0	0.16	0.166
2	3	1	1/5	1	0.16	0.166
					0.96	

→ Normalize the weights

x_1	x_2	y	new_wt	$\hat{y}$	Range
1	5	1	0.25	0	0-0.25
2	4	0	0.25	1	0.25-0.50
9	2	1	0.166	1	0.50-0.666
1	7	0	0.166	0	0.666-0.832
2	3	1	0.166	1	0.832-1

4) Upsampling
- choose n random # b/w 0-1

e.g. 0.54, 0.2, 0.99, 0.06, 0.31
(3) (1) (5) (1) (2)

- check which range they fall in & add that row to new dataset

x_1	x_2	y	wt
9	2	1	1/5
1	5	1	1/5
2	3	1	1/5
1	5	1	1/5
2	4	0	1/5

→ reset weights to 1/n

sklearn.ensemble.AdaBoostClassifier

class sklearn.ensemble.**AdaBoostClassifier**(*estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None*) [source]

lr x α_i

↑ n_estimators ↓ learning_rate

5) Redo steps 2-4 for as many decision stumps in model.

6) $\hat{y} = \text{sign}(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x) + \ldots + \alpha_n h_n(x))$

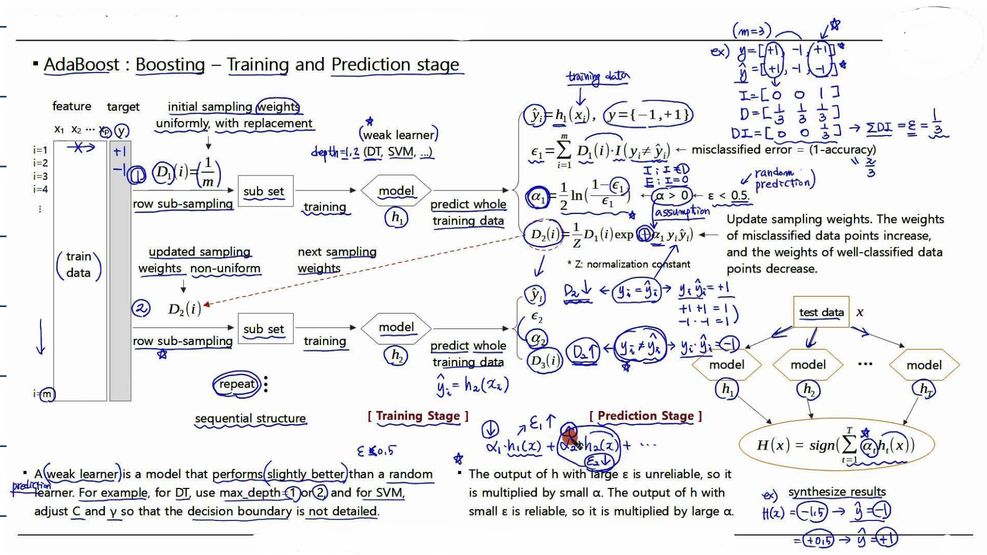
AdaBoost Revisited

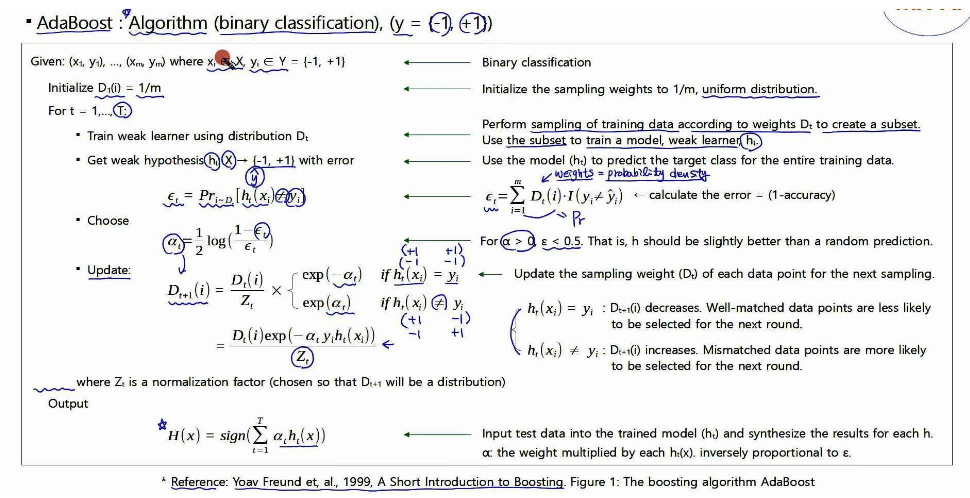
Saturday 25 May 2024 1:01 AM

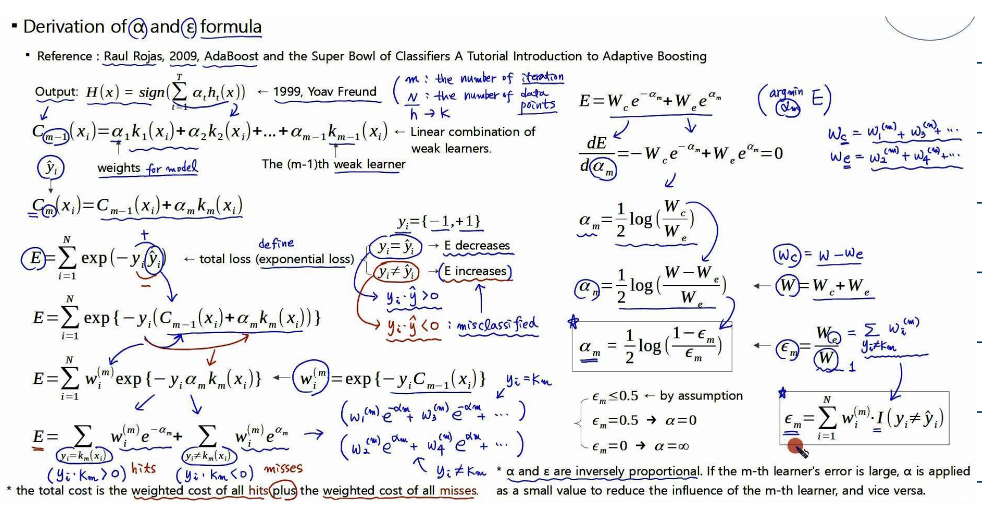
→ Boosting is a method of gradually strengthening a weak learner model.

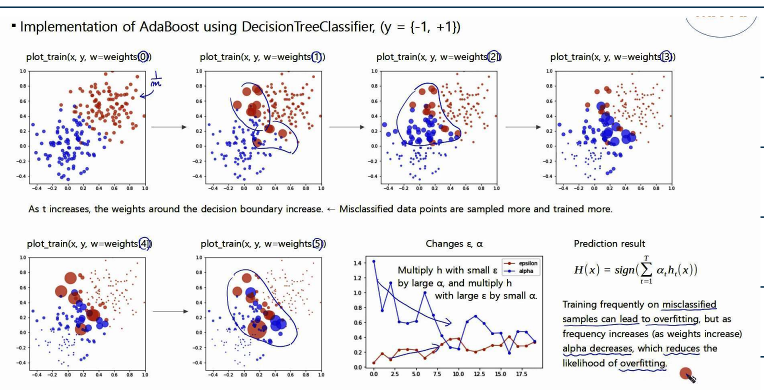
→ In estimation stage, the results of multiple models are combined.

→ AdaBoost is an algorithm that increases accuracy by focusing more on misclassified data points during training stage.









▪ AdaBoost : Algorithm (binary classification), (y = {0, 1})

- Use y = {0, 1} instead of y = {-1, +1}. It can be easily extended to multiclass classification.

Algorithm 1: AdaBoost (Freund & Schapire 1997)

1. Initialize the observation weights $w_i = 1/n, \ i=1,2,\dots,n$
2. **For** m = 1 **to** M:
   a) Fit a classifier $T^{(m)}(x)$ to the training data using weights w_i
   b) Compute

   actual class (y) · training data

   $$err^{(m)} = \sum_{i=1}^{n} w_i \cdot I(c_i \neq T^{(m)}(x_i)) \ / \ \sum_{i=1}^{n} w_i$$

   ε · predicted class ($\hat{y}$)

   c) Compute

   $$\alpha^{(m)} = \log \frac{1-err^{(m)}}{err^{(m)}}$$

   d) Set

   $$w_i \leftarrow w_i \cdot \exp(\alpha^{(m)} \cdot I(c_i \neq T^{(m)}(x_i))), \quad i=1,2,\dots,n$$

   e) Re-normalize w_i

   $y \neq \hat{y} \rightarrow I=1 \rightarrow w \uparrow$
   $y = \hat{y} \rightarrow I=0 \rightarrow w \rightarrow$ ↓

3. Output

$$C(x) = \underset{k}{argmax} \sum_{m=1}^{M} \alpha^{(m)} \cdot I(T^{(m)}(x)=k)$$

* Reference: Ji Zhu, et, al., 2006, Multi-class AdaBoost, Algorithm 1

In this form, multiple class classification is also possible, but α <0 problem occurs if $err^{(m)} > 0.5$. In binary classification, α <0.5 is possible even in weak learners, but in three classes, α <0 occurs easily because the accuracy of random prediction is 1/3 and $err^{(m)} = 0.67$. → Need to be improved → SAMME algorithm in the next video.

It is always 1 because it is normalized.

Not multiplied by 1/2. The reason was not explained.

Use argmax instead of the sign function.

$\hat{y}$ · $\varepsilon\uparrow$ · α
m=1 : 0 · [1 0] × 0.1 = [0.1 0]
2 : 0 · [1 0] × 0.1 = [0.1 0]
3 : 1 · $\varepsilon\downarrow$ · [0 1] × 0.9 = [0 0.9]
4 : 0 · [1 0] × 0.1 = [0.1 0]
majority = 0 · Σ = [0.3 0.9]
($\hat{y}$ = 1)

(example)

test data · $\hat{y}$

$T^{(m)}(x)$ · $I(T^{(m)}(x)=k)$

	$\hat{y}$		k=0 1	$\alpha^{(m)}$		
m=1:	0	→	[1 0]	* 0.5	=	[0.5 0]
m=2:	0	→	[1 0]	* 0.7	=	[0.7 0]
m=3:	1	→	[0 1]	* 0.2	=	[0 0.2]
m=4:	0	→	[1 0]	* 0.6	=	[0.6 0]
					sum =	[1.8 0.2]

$\hat{y}$ → OHE · argmax = 0 ← C(x)

▪ Multiclass Classification – SAMME Algorithm (Stagewise Additive Modeling using a Multi-class Exponential loss function)

- In 2006, Ji Zhu et al. proposed the following algorithm to solve the multiclass problem of Freund's (1995) AdaBoost algorithm.
- The err(m) < 0.5 condition in Freund's model was modified to err(m) < 1 – 1/K (K: the number of classes). If K=3, then err(m) < 0.67.

ε

Multi-class AdaBoost

Ji Zhu, Department of Statistics University of Michigan Ann Arbor, MI 48109
Saharon Rosset, Data Analytics Group IBM Research Center Yorktown Heights, NY 10598
Hui Zou, School of Statistics University of Minnesota Minneapolis, MN 55455
Trevor Hastie, Department of Statistics Stanford University Stanford, CA 94305
January 12, 2006

Abstract

Boosting has been a very successful technique for solving the two-class classification problem. In going from two-class to multi-class classification, most algorithms have been restricted to reducing the multi-class classification problem to multiple two-class problems. In this paper, we propose a new algorithm that naturally extends the original AdaBoost algorithm to the multiclass case without reducing it to multiple two-class problems. Similar to AdaBoost in the two-class case, this new algorithm combines weak classifiers and only requires the performance of each weak classifier be better than random guessing (rather than 1/2). We further provide a statistical justification for the new algorithm using a novel multi-class exponential loss function and forward stage-wise additive modeling. As shown in the paper, the new algorithm is extremely easy to implement and is highly competitive with the best currently available multi-class classification methods.

Algorithm 2: **SAMME** · y = {0, 1} → y = {0, 1, 2 ⋯ K-1}

1. Initialize the observation weights $w_i = 1/n, \ i=1,2,\dots,n$
2. **For** m = 1 **to** M:
   a) Fit a classifier $T^{(m)}(x)$ to the training data using weights w_i · if K=2, err < 1 − 1/2 = 0.5
   b) Compute · if K=3, err < 1 − 1/3 = 0.67

   $$err^{(m)} = \sum_{i=1}^{n} w_i \cdot I(c_i \neq T^{(m)}(x_i)) \ / \ \sum_{i=1}^{n} w_i$$

   c) Compute

   $$\alpha^{(m)} = \log \frac{1-err^{(m)}}{err^{(m)}} + \log(K-1)$$

   This part was added to the Freund (1997) model.

   For $\alpha^{(m)} > 0, \quad err^{(m)} < 1 - \frac{1}{K}$

   d) Set

   $$w_i \leftarrow w_i \cdot \exp(\alpha^{(m)} \cdot I(c_i \neq T^{(m)}(x_i))), \ i=1,2,\dots,n$$

   e) Re-normalize w_i
3. Output

$$C(x) = \underset{k}{argmax} \sum_{m=1}^{M} \alpha^{(m)} \cdot I(T^{(m)}(x)=k)$$

proof

$$\log \frac{1-err^{(m)}}{err^{(m)}} + \log(K-1) > 0$$

$$\frac{1-err^{(m)}}{err^{(m)}} > \frac{1}{K-1}$$

$$\frac{1}{err^{(m)}} > \frac{K}{K-1}$$

▪ AdaBoost Regression - Algorithm

3. BOOSTING: In bagging, each training example is equally likely to be picked. In boosting, the probability of a particular example being in the training set of a particular machine depends on the performance of the prior machines on that example. The following is a modification of Adaboost.R [Freund and Schapire (1996a)].

1. Initialize, to each training pattern we assign a weight $w_i=1/N_1$, i =1, …,N_1 (uniform)
   Pick N_1 samples (with replacement) to form the training set (= subset)

Repeat the following while the average loss L defined below is less than 0.5 .

2. Construct a regression machine t from that training set. Each machine makes a hypothesis: h_t: x→y
3. Pass every member of the (entire) training set through this machine to obtain a prediction $y_i^{(p)}(x_i)$, i =1, …N_1.
4. Calculate a loss for each training sample $L_i = L[|y_i^{(p)} - y_i|]$ (predicted, actual value). The loss L may be of any functional form as long as L ∈ [0, 1]. If we let

$$D = \sup |y_i^{(p)}(x_i) - y_i| \quad i=1,\dots,N_1$$

then we have three candidate loss functions:

$$L_i = \frac{|y_i^{(p)} - y_i|}{D} \quad \text{(linear)}$$

$$L_i = \frac{|y_i^{(p)} - y_i|^2}{D^2} \quad \text{(square law)}$$

$$L_i = 1 - \exp\left(\frac{-|y_i^{(p)} - y_i|}{D}\right) \quad \text{(exponential)}$$

* classification

$$err^{(m)} = \sum_{i=1}^{n} w_i \cdot I(c_i \neq T^{(m)}(x_i)) \ / \ \sum_{i=1}^{n} w_i$$

$$\alpha^{(m)} = \log \frac{1-err^{(m)}}{err^{(m)}}$$

$$w_i \leftarrow w_i \cdot \exp(\alpha^{(m)} \cdot I(c_i \neq T^{(m)}(x_i)))$$

5. Calculate an average loss: $\bar{L} = \sum_{i=1}^{N_1} L_i w_i$
6. Form $\beta = \frac{\bar{L}}{1-\bar{L}}$ ← Proportional to average loss

   β ($\bar{L}$) · β^{1-L_i} (L_i)

   β is a measure of confidence in the predictor. Low β means high confidence in the prediction.

7. Update the weights: $w_i \rightarrow w_i \beta$ ** $(1-L_i)$, where ** indicates exponentiation. The smaller the loss the more the weight is reduced making the probability smaller that this pattern will be picked as a member of the training set for the next machine in the ensemble.

training stage

prediction stage

8. For a particular input x_i, each of the T machines makes a prediction h_t, t=1, …,T. Obtain the cumulative prediction h_f using the T predictors:

$$h_f = \inf\left\{ y \in Y : \sum_{t:h_t \leq y} \log\left(\frac{1}{\beta_t}\right) \geq \frac{1}{2}\sum_t \log\left(\frac{1}{\beta_t}\right)\right\}$$

This is the weighted median.

Weight of each machine (α role in classification). It is inversely proportional to the average loss. The weight of machines with large losses is reduced.

▪ AdaBoost Regression - Algorithm

- In algorithm step 8, the weighted median is used for prediction.
- Here we will try both weighted average and weighted median.

model weights

$$w_t = \underbrace{\log\left(\frac{1}{\beta_t}\right)}_{\alpha} \Big/ \sum_t \log\left(\frac{1}{\beta_t}\right) \quad \leftarrow \text{Weight for the value estimated by each model}$$

(normalize)

1. Using weighted average

$$h_f = \sum_t \hat{y}_t w_t \quad \leftarrow \text{weighted average}$$

test data → h_0, h_1 ⋯ h_{T-1} models → ($\hat{y}_0$ $\hat{y}_1$ $\hat{y}_{T-1}$) × $\log\frac{1}{\beta}$ → (w_0 w_1 w_{T-1})

```
# Method-1: Using weighted average
wavg_pred = np.sum(y_pred * w, axis=1)
```

y_pred → 2D: ($\hat{y}_0^{(1)}$ $\hat{y}_1^{(1)}$ ⋯ $\hat{y}_{T-1}^{(1)}$ / $\hat{y}_0^{(2)}$ $\hat{y}_1^{(2)}$ ⋯ $\hat{y}_{T-1}^{(2)}$) × [w_0 w_1 ⋯ w_{T-1}]

2. Using weighted median as in the paper

$$h_f = \inf\left\{ y \in Y : \sum_{t:h_t \leq y} \log\left(\frac{1}{\beta_t}\right) \geq \frac{1}{2}\sum_t \log\left(\frac{1}{\beta_t}\right)\right\}$$

$$h_f = \inf\left\{ y \in Y : \sum_{t:h_t \leq y} w_t \geq \frac{1}{2}\sum_t w_t\right\}$$

If y is sorted in ascending order, the y at the position where the sum of the lower w is half of the total sum of w becomes the final output.

The y value estimated by the first model.

the number of models (ex: T=50)

ex)

	0 1 2 3 4 5 6 7 8 9 10 11 … 49
y_pred	$\hat{y}_0$ $\hat{y}_1$ $\hat{y}_2$ …
	$\hat{y}_3$ $\hat{y}_0$ argsort, ascending · 23 $\hat{y}_{34}$
i_pred	3 0 9 11 5 4 8 6 1 10 2 7 … 34 …
w	w_3 w_0 w_9 w_{11} w_5 …
accumulated w (w-acc)	a_0 a_1 a_2 a_3 a_4 … … a_{49}
	is $(a_i \geq 0.5 \times a_{49})$? 23
False or True	F F F … … F T T T …

$\sum_{i=1}^{T} w_i$

the position of weighted median (ex: 23)

weighted median of y_pred = $\hat{y}_{34}$ ← final estimate

```
# weighted median: (sum of the lower w ≥ half of the total sum of w)
i_pred = np.argsort(y_pred, axis=1)
w_acc = np.cumsum(w[i_pred], axis=1)        # accumulated w
is_med = w_acc >= 0.5 * w_acc[:, -1][:, np.newaxis]
i_med = is_med.argmax(axis=1)               # 23
y_med = i_pred[np.arange(n_test), i_med]   # 34
wmed_pred = np.array(y_pred[np.arange(n_test), y_med]) # final estimate
```