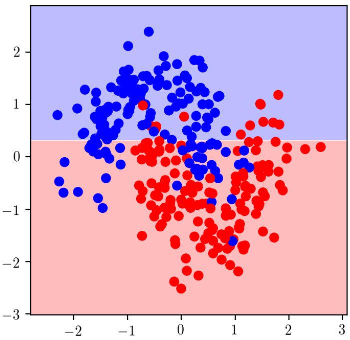
Ensemble Learning - 3
Boosting

John Klein

Goal : Build incrementally an ensemble so that new coming classifiers fix misclassification errors of the previous ones.
Turn a weak classifier into a strong ensemble.

- Use bagging in case of overfitting ($\mathcal{H}$ is big $\to$ high variance in trained predictors).
- Use boosting in case of underfitting ($\mathcal{H}$ is small $\to$ low variance in trained predictors but high bias).

Remember the **bad tree** ?



- This is a **weak** learner that underfits data...
- But achieves ≈ 78% accuracy which is a lot better than random guess.

Question : Can a set of weak learners create a single strong learner ? [Kearns 88]

Answer : Yes [Schapire 90] (regardless of how is trained the weak learner)

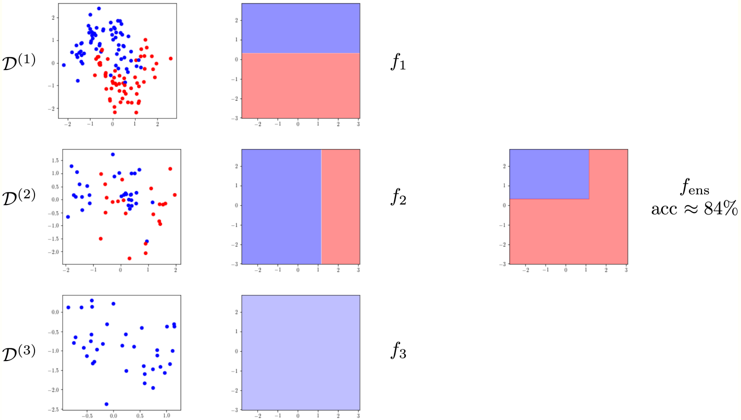
1. Train a "bad tree" f_1 on $\mathcal{D}^{(1)} \subsetneq \mathcal{D}$.
2. Train a "bad tree" f_2 on $\mathcal{D}^{(2)} \subsetneq \mathcal{D} \setminus \mathcal{D}^{(1)}$ s.t. half of the points in $\mathcal{D}^{(2)}$ are misclassified by f_1.
3. Train a "bad tree" f_3 on $\mathcal{D}^{(3)} \subsetneq \mathcal{D} \setminus \mathcal{D}^{(1+2)}$ s.t. it contains data points for which f_1 and f_2 disagree.

Then, the majority vote ensemble of (f_1, f_2, f_3) has a smaller risk than f_1 (with high prob.).

The Strength of Weak Learnability, Schapire 1990.

Boosting

boosted weak learner



Sequentially train **weak** classifiers for a binary task $y \in \{-1; +1\}$:

1. Train weak classifier f_m on the weighted training set $\mathcal{D}_{\text{train}} \times \left\{ w_m^{(1)}, \ldots, w_m^{(n)} \right\}$.
2. Evaluate the classification uncertainty on each data point and update weights accordingly.
3. Update classifier mixing coefficients α_m

Finally, return $f_{\text{ens}} = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m f_m \right)$.

$s(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m f_m(\mathbf{x}) \in \mathbb{R}$ is a score whereas each $f_m(\mathbf{x}) \in \mathcal{C}$ is a class label.

One **loss** to rule them all !

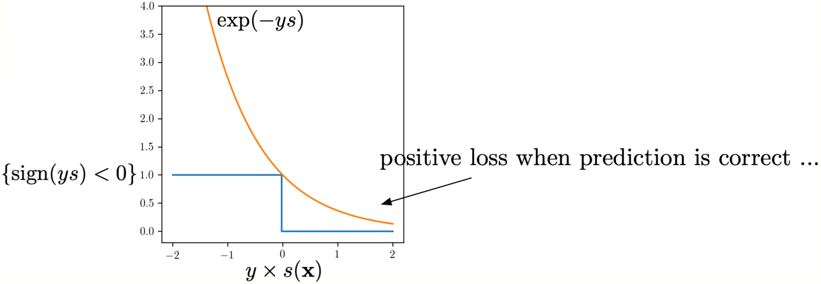
- In classification, the standard loss function is the **0-1 loss**.
- Boosting aims at minimizing a weighted **0-1 loss** over the training set :

$$J_m = \sum_{i=1}^{n} w_m^{(i)} \left(1 - \mathbb{1}_{y^{(i)}} \left(f_m \left(\mathbf{x}^{(i)}\right)\right)\right).$$

- The boosted ensemble, however, minimizes the exponential loss :

$$J_{\text{ens}} = \sum_{i=1}^{n} \exp \left(-y^{(i)} \frac{1}{2} \sum_{m=1}^{M} \alpha_m f_m \left(\mathbf{x}^{(i)}\right)\right).$$

Question : does an exponential loss make sense ?



- Exponential loss = surrogate of the 0-1 loss

Proof that optimizing J_{ens} w.r.t. to f_m = optimizing J_m.

- We define weights as

$$J_{\text{ens}} = \sum_{i=1}^{n} \underbrace{\exp\left(-y^{(i)}\frac{1}{2}\sum_{k=1}^{m-1}\alpha_k f_k\left(\mathbf{x}^{(i)}\right)\right)}_{:=w_m^{(i)}} \times \exp\left(-\frac{1}{2}\alpha_m y^{(i)} f_m\left(\mathbf{x}^{(i)}\right)\right).$$

- We deduce weight update :

$$w_{m+1}^{(i)} = w_m^{(i)} \exp\left(-\frac{1}{2}\alpha_m y^{(i)} f_m\left(\mathbf{x}^{(i)}\right)\right).$$

- Depending on the correctness of the prediction, the weight grows or decreases.

Proof that optimizing J_{ens} w.r.t. to f_m = optimizing J_m.

Let's use the following decomposition of the ensemble loss :

- Denote $\mathcal{M}$ the set of misclassified points by f_m.
- Denote $\mathcal{T}$ the set of correctly classified points by f_m.
- We have

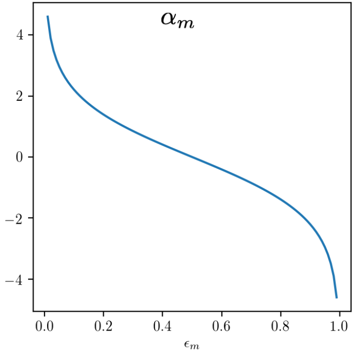
$$
\begin{aligned}
J_{\text{ens}} &= e^{\frac{-\alpha_m}{2}} \sum_{i \in \mathcal{T}} w_m^{(i)} + e^{\frac{\alpha_m}{2}} \sum_{i \in \mathcal{M}} w_m^{(i)}, \\
&= e^{\frac{-\alpha_m}{2}} \sum_{i=1}^{n} w_m^{(i)} + \left(e^{\frac{\alpha_m}{2}} - e^{\frac{-\alpha_m}{2}} \right) \sum_{i=1}^{n} w_m^{(i)} \\
&\qquad \times \mathbb{I}\left\{ f_m\left(\mathbf{x}^{(i)}\right) \neq y^{(i)} \right\}.
\end{aligned}
$$

Next problem : we need an incremental rule for f_m and α_m.

- Minimizing J_{ens} wrt mixing coefficients gives

$$\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right), \tag{1}$$

$$\epsilon_m = \frac{\sum_{i=1}^{n} w_m^{(i)} \mathbb{I}\left\{f_m\left(\mathbf{x}^{(i)}\right) \neq y^{(i)}\right\}}{\sum_{i=1}^{n} w_m^{(i)}}. \tag{2}$$



- If $\epsilon_m = 0.5$ then $\alpha_m = 0$, the classifier is discarded from the ensemble.

Final problem : how to minimize a weighted loss J_m ?

First possibility : weak learner is compatible.

ex : logistic regression

To learn, we differentiate the cross-entropy loss w.r.t. the model parameters.

Differentiating the weighted version is no harder.

Second possibility : weak learner is not compatible.

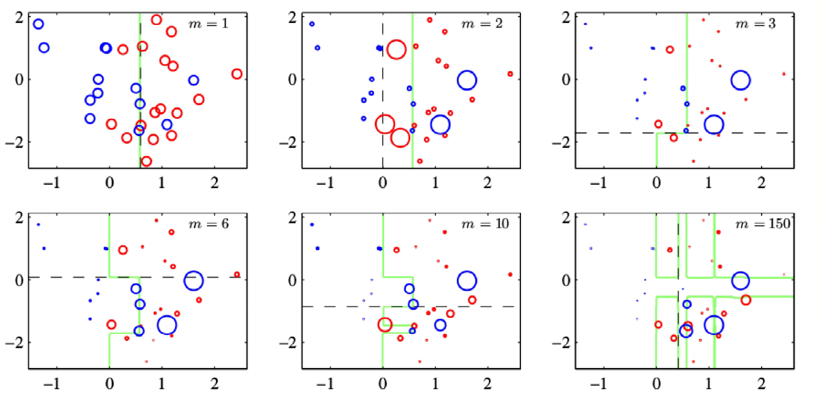
Sample a bootstrap sample from $\mathcal{D}$ where each data point $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ is selected with probability

$$p_i = \frac{w_m^{(i)}}{\sum_{i'} w_m^{(i')}}.$$

Train your new base learner on this bootstrap sample.

Remark : in both cases we do not really minimize J_m which relies on the 0-1 loss.

Illustration : boosted trees.



[Bishop 2006]

Theoretical guarantees

- Bound on the ensemble empirical risk $\hat{e}_{\text{ens}}$

$$\hat{e}_{\text{ens}} = \sum_{i=1}^{n} \mathbb{1}_{y^{(i)}} \left(f_{\text{ens}} \left(\mathbf{x}^i \right) \right) \leq e^{-2 \sum_{m=1}^{M} \left(\frac{1}{2} - \epsilon_m \right)^2}.$$

- PAC-bound on the risk... but it increases w.r.t. M !

- PAC-bound on the margin (regardless of M).

$\rightarrow$ More on this : *The Boosting Approach to Machine Learning An Overview*, Schapire 2001.

Limitations

- Outliers $\rightarrow$ much slower convergence.
- No probabilistic embedding.
- No obvious way to regularize.
- Limited interpretability.

The general idea :

- We want to train incrementally an ensemble whose aggregate at previous step is $f_{\text{ens}}^{(m)}$
- The best additive corrective learner in $\mathcal{H}$ that we can add is given by

$$f_{\text{ens}}^{(m+1)} = f_{\text{ens}}^{(m)} + \arg\min_{h \in \mathcal{H}} \mathbb{E}_{x,y} \left[L\left(y, f_{\text{ens}}^{(m)}(\mathbf{x}) + h(\mathbf{x}) \right) \right]. \quad (3)$$

☹ optimization problem too hard

The general idea :

- We want to train incrementally an ensemble whose aggregate at previous step is $f_{\text{ens}}^{(m)}$
- Let's do something sub-optimal but workable :

$$f_{\text{ens}}^{(m+1)} = f_{\text{ens}}^{(m)} - \gamma_m \times \nabla_f \big|_{f=f_{\text{ens}}^{(m)}} \mathbb{E}_{x,y} \left[L\left(y, f\left(\mathbf{x}\right)\right)\right]. \tag{4}$$

☺ gradient descent, yes we can !
This gradient step will improve things because we directly differentiate the loss.

- Denote gradient as $g_m = \nabla_f|_{f=f_{\text{ens}}^{(m)}} \mathbb{E}_{x,y}\left[L\left(y, f\left(\mathbf{x}\right)\right)\right]$.
- g_m is actually a **function** because this is a functional derivative !
- It can be proved that the gradient is the following conditional expectation

$$g_m\left(\mathbf{x}\right) = \mathbb{E}_{y|x}\left[\frac{\partial L\left(y, f_{\text{ens}}^{(m)}\left(\mathbf{x}\right)\right)}{\partial f_{\text{ens}}^{(m)}\left(\mathbf{x}\right)}\right] \tag{5}$$

- ☹ We only know this function at points $\mathbf{x}$ belonging to $\mathcal{D}_{\text{train}}$.
- $\rightarrow$ Approximate g_m by regressing it with a function chosen from $\mathcal{H}$!

- The gradient step size γ_m is chosen as

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n_{\text{train}}} L\left(y^{(i)}, f_{\text{ens}}^{(m)}\left(\mathbf{x}^{(i)}\right) - \gamma \hat{g}_m\left(\mathbf{x}^{(i)}\right)\right) \tag{6}$$

Best γ w.r.t. the empirical risk / train error.

- Gradient boosting in practice $\rightarrow$ specify L and $\mathcal{H}$.

- Regression task and loss is square loss :

$$L(y, f(\mathbf{x})) = \frac{1}{2}(y - f(\mathbf{x}))^2,$$

$$\text{hence } \frac{\partial L\left(y, f_{\text{ens}}^{(m)}(\mathbf{x})\right)}{\partial f_{\text{ens}}^{(m)}(\mathbf{x})} = f(\mathbf{x}) - y.$$

- $\mathcal{H}$ = trees with a given max depth.

- Training algorithm = CART.

Our data $\mathcal{D}_{\text{train}}$:

x_1 height	x_2 age	x_3 gender	y weight
1.8	22	m	70
1.7	28	m	70
1.65	25	f	55
1.77	32	f	68
1.92	22	m	72
1.85	19	m	93

- Initialize : $f_{\text{ens}}^{(0)}(\mathbf{x}) = 71.33$
  (average of $y^{(i)}$ in the training set)

Our data $\mathcal{D}_{\text{train}}$:

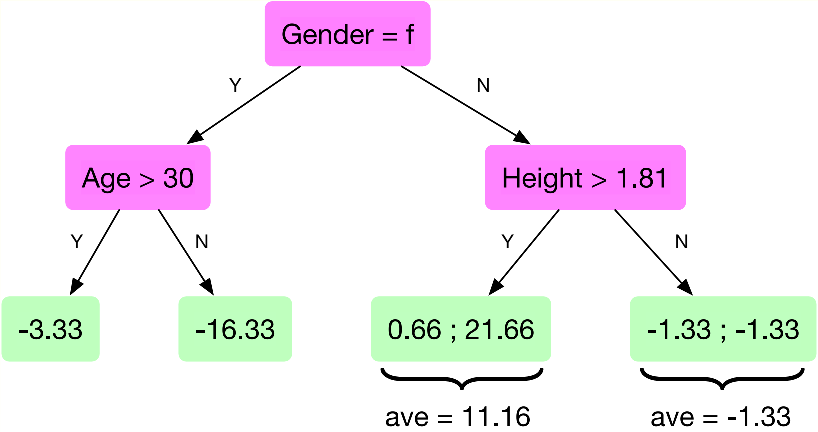
x_1 height	x_2 age	x_3 gender	y weight	r_i resid.
1.8	22	m	70	-1.33
1.7	28	m	70	-1.33
1.65	25	f	55	-16.33
1.77	32	f	68	-3.33
1.92	22	m	72	0.66
1.85	19	m	93	21.66

- step $m = 0$
- Compute pseudo-residuals = neg. gradient

$$r_i = -\frac{\partial L\left(y, f_{\text{ens}}^{(m)}(\mathbf{x})\right)}{\partial f_{\text{ens}}^{(m)}(\mathbf{x})} = f_{\text{ens}}^{(m)}(\mathbf{x}) - y$$

- Train $\hat{g}_m$ to regress $-r_i$ from inputs using CART.

- Suppose max depth = 2, and CART yields



- Obtain new ensemble as

$$f_{\text{ens}}^{(m+1)} = f_{\text{ens}}^{(m)} - \gamma_m \hat{g}_m$$

- Take $m = 0$, $i = 3$ and suppose $\gamma_0 = 1$

$$\begin{aligned} f_{\text{ens}}^{(1)}\left(\mathbf{x}^{(3)}\right) &= f_{\text{ens}}^{(0)}\left(\mathbf{x}^{(3)}\right) - r_3, \\ &= 71.33 - 16.33 \\ &= 55 \end{aligned}$$

- Perfect match ! .... but this is overfitting $\rightarrow$ need to regularize by using a **learning rate** $\eta < 1$!

- **Question** : what is the justification behind averages for leafs with multiple outputs ?

- Define input space regions that are mapped by the newly trained tree $\hat{g}_m$ to a single leaf as $(R_j)_{j=1}^{J_{\text{leaf}}}$ (terminal regions).

- Solving $\arg\min_{\gamma} \sum_{\mathbf{x}^{(i)} \in R_j} L\left(y^{(i)}, f_{\text{ens}}^{(m)}\left(\mathbf{x}^{(i)}\right) + \gamma\right)$ gives those averages.

- Slightly different from the theory which involves the global minimization (6) to find γ_m.

- **Implementation** : XGBoost.
- Open source software library heavily relying on gradient boosted trees + few tricks.
- Available in Python, R and Julia.
- A popular resource in ML competition... (XGBoost + Centralien = ♡).