ROUEN NORMANDIE

Machine Learning

Boosting

Simon BERNARD
simon.bernard@univ-rouen.fr

Outline

Chapters :

1. From decision trees to decision forests
2. Random Forests
3. Boosting
    - Introduction to boosting
    - Adaboost
    - Gradient Boosting Machine

Introduction to boosting

Introduction to boosting

- $f : \mathcal{X} \rightarrow \mathcal{Y}$ is the relation we want to learn
- We seek to find a model $h \in \mathcal{H}$ that "explain" f, $\mathcal{H}$ being a hypothesis space
- $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ is a loss function to measure the effectiveness of a given $h \in \mathcal{H}$
- The (generalization) error, also called the true risk, of a given h is :

$$R(h) = E_{(\mathbf{x},y)\sim\mathcal{D}}\left[\ell(y, h(\mathbf{x}))\right]$$

- Here, we make the *realizability assumption* : $f \in \mathcal{H}$ meaning $\exists h^\star \in \mathcal{H}$ s.t. $R(h^\star) = 0$
- For simplicity, consider binary classification tasks only

Introduction to boosting

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Strong learnability

A problem is strongly PAC-learnable if there exists an algorithm $\mathcal{A}$ such that, $\forall f$, $\forall \epsilon > 0$, $\forall \delta > 0$, if $\mathcal{A}$ is given $n = poly(\frac{1}{\epsilon}, \frac{1}{\delta})$ instances, then it outputs h such that :

$$P(R(h) \leq \epsilon) \geq 1 - \delta$$

Weak learnability

A problem is weakly PAC-learnable if $\exists \gamma > 0$ and there exists an algorithm $\mathcal{A}$ such that, $\forall f$, $\forall \delta > 0$, if $\mathcal{A}$ is given $n = poly(\frac{1}{\delta})$ instances, then it outputs h such that :

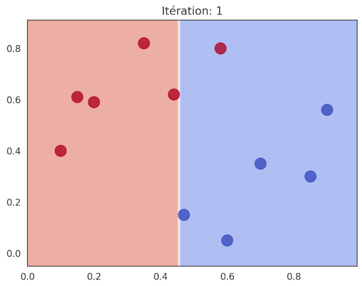
$$P(R(h) \leq \frac{1}{2} - \gamma) \geq 1 - \delta$$

- Weak learnability only requires $\mathcal{A}$ to supply a h better than a purely random prediction
- By extension, We call $\mathcal{A}$ a weak learner and h a weak classifier

Introduction to boosting

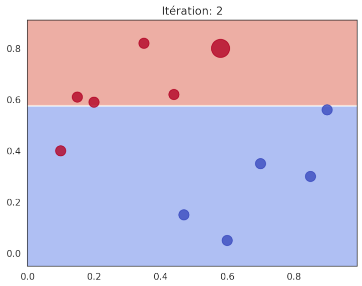
- A key question : can we transform any weak learner $\mathcal{A}$ into a strong learner $\mathcal{A}'$?
- Long story short, the best answers are Boosting algorithms
- Key idea : since $\mathcal{A}$ gives a weak h for a given $\mathcal{D}$, create several different $\mathcal{D}_k$ to obtain several different weak h_k and combine them afterward
- Difference with bagging is that $\mathcal{D}_k$ are created to focus on the errors of h_{k-1}

Introduction to boosting



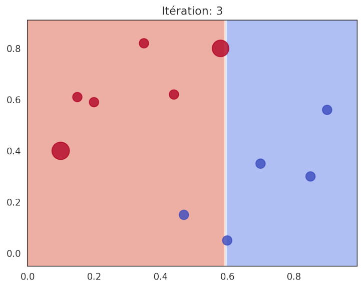
- Init. : all instances have the same weight ($\sim$ same importance for learning)
- Learn a stump classifier : 1 red instance wrongly classified

Introduction to boosting



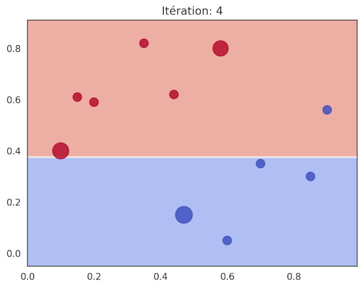
- Increase the weight of this red instance so that it has a greater impact on learning
- Learn a stump classifier : a different red instance wrongly classified

Introduction to boosting



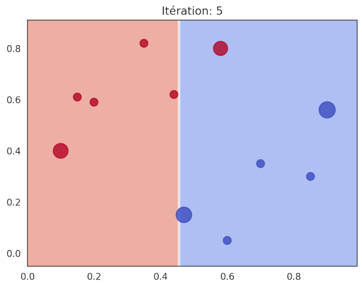
- Update weights to incorporate this new error for learning the next classifier
- Learn a stump classifier : 1 blue instance wrongly classified

Introduction to boosting



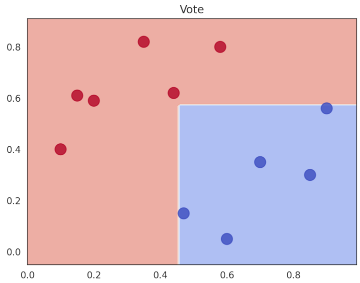
- Update weights : 3 (difficult) instances with a higher weight
- Learn a stump classifier : 1 new blue instance wrongly classified

Introduction to boosting



- Update weights
- Learn a stump classifier : back to the first classifier

Introduction to boosting



- Combine the classifiers by weighted voting
- Diversity is created by having classifiers focus on different instances

AdaBoost

AdaBoost

- Initially for two-class classification tasks where $y \in \{-1, 1\}$
- Requires a weak learner $\mathcal{A}$ that can take instance weights into account (e.g. decision tree)
- Each training instance $\mathbf{x}_i$ is assigned a weight $w_i \in [0, 1]$, with

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

- The empirical error rate of a classifier h is thus :

$$\hat{\epsilon} = \sum_{i=1}^{n} w_i \mathbb{1}_{y_i \neq h(\mathbf{x}_i)}$$

AdaBoost

- AdaBoost is designed to minimize the exponential loss (for $y \in \{-1, 1\}$) :

$$\ell(y, h(\mathsf{x})) = e^{-yh(\mathsf{x})}$$

- At iteration k, a new h_k is learnt and added to the combination s.t. it minimizes the loss :

$$\begin{aligned} H_k(\mathsf{x}) &= H_{k-1}(\mathsf{x}) + \alpha_k h_k(\mathsf{x}) \\ \mathcal{L} &= \sum_{i=1}^{n} \ell(y_i, H_k(\mathsf{x}_i)) = \sum_{i=1}^{n} e^{-y_i H_k(\mathsf{x}_i)} \\ &= \sum_{i=1}^{n} e^{-y_i H_{k-1}(\mathsf{x}_i)} e^{-y_i \alpha_k h_k(\mathsf{x}_i)} \end{aligned}$$

AdaBoost

- At iteration k, we focus on finding h_k :

$$\mathcal{L} = \sum_{i=1}^{n} w_i^{(k)} e^{-y_i \alpha_k h_k(\mathbf{x}_i)}$$

  where

$$w_i^{(k)} = e^{-y_i H_{k-1}(\mathbf{x}_i)}$$

- We can split this summation between correct and incorrect predictions :

$$\begin{aligned}\mathcal{L} &= \sum_{y_i = h_k(\mathbf{x}_i)} w_i^{(k)} e^{-\alpha_k} + \sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)} e^{\alpha_k} \\ &= \sum_{i=1}^{n} w_i^{(k)} e^{-\alpha_k} + \sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)} \left(e^{\alpha_k} - e^{-\alpha_k}\right)\end{aligned}$$

AdaBoost

- In this equation

$$\mathcal{L} = \sum_{i=1}^{n} w_i^{(k)} e^{-\alpha_k} + \sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)} \left(e^{\alpha_k} - e^{-\alpha_k}\right)$$

  for a fixed $\alpha_k > 0$, the only term that depends on h_k is $\sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)}$
- Thus, the h_k that minimizes $\mathcal{L}$ is the one that minimizes $\sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)}$
- This is the weighted error of h_k with weights :

$$w_i^{(k)} = e^{-y_i H_{k-1}(\mathbf{x}_i)}$$

AdaBoost

- Input : a training set $\mathcal{D}$, a weak learner $\mathcal{A}$, a number L of iterations
- Output : a combining classifier

$$H_L(\mathrm{x}) = sign\left(\sum_{k=1}^{L} \alpha_k h_k(\mathrm{x})\right)$$

- Initialization :

$$W^{(1)} = \left(w_1^{(1)}, w_2^{(1)}, \dots, w_n^{(1)}\right)$$

with

$$w_i^{(1)} = \frac{1}{n}, \ \forall i = 1, \dots, n$$

AdaBoost

- Loop : for $k = 1$ to L
  1. Learn the k^{th} classifier $h_k = \mathcal{A}(\mathcal{D}, W^{(k)})$
  2. Compute its weighted error rate :

$$\hat{\epsilon}_k = \sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)}$$

  3. Compute α_k
  4. Update the instance weights for the next iteration $(W^{(k+1)})$:

$$\begin{aligned} w_i^{(k+1)} &= w_i^{(k)} . \frac{1}{Z_k} e^{(-\alpha_k y_i h_k(\mathbf{x}_i))} \quad \forall i = 1, \ldots, n \\ &= w_i^{(k)} . \frac{1}{Z_k} \begin{cases} e^{-\alpha_k} & if\ h_k(\mathbf{x}_i) = y_i \\ e^{\alpha_k} & else \end{cases} \end{aligned}$$

  where Z_k is a normalization coefficient

AdaBoost

- To determine the α_k that minimizes $\mathcal{L}$ with the chosen h_k, we differentiate :

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \alpha_k} &= \frac{\partial}{\partial \alpha_k}\left(\sum_{y_i = h_k(\mathbf{x}_i)} w_i^{(k)} e^{-\alpha_k} + \sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)} e^{\alpha_k}\right) \\ &= -\sum_{y_i = h_k(\mathbf{x}_i)} w_i^{(k)} e^{-\alpha_k} + \sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)} e^{\alpha_k}\end{aligned}$$

- Setting this to zero lead to :

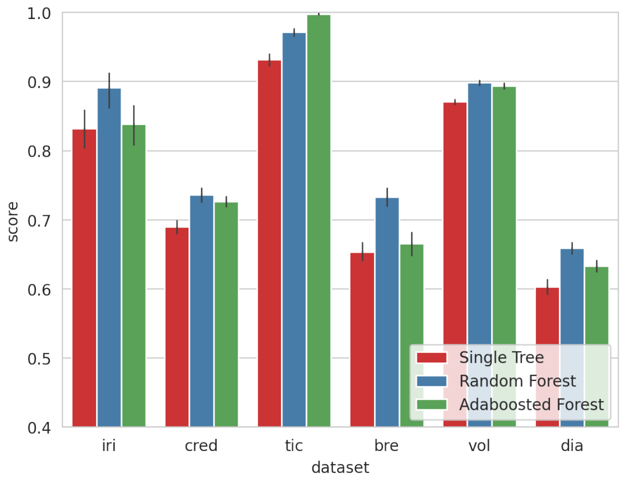
$$\alpha_k = \frac{1}{2}\ln\left(\frac{\sum_{y_i = h_k(\mathbf{x}_i)} w_i^{(k)}}{\sum_{y_i \neq h_k(\mathbf{x}_i)} w_i^{(k)}}\right) = \frac{1}{2}\ln\left(\frac{1-\hat{\epsilon}_k}{\hat{\epsilon}_k}\right)$$

Discussion

- Theoretical results shows that there is a risk of overfitting if :
  - L is too big compared to n
  - accentuated if the weak classifiers are complex
- In practice, overfitting is quite rare with boosting because even if the resulting classifier is more and more complex, it is also more and more confident in its prediction [1]

1. Can be explained through the margin maximization perspective, but we won't go into these details here

AdaBoost



(Note : for multi-class problems, the variant called Adaboost.SAMME has been used)

Gradient boosting

Gradient boosting

- Many ML models can be written as a linear combination of simpler models :

$$H(\mathbf{x}) = \sum_{k=1}^{L} \alpha_k h(\mathbf{x}, \theta_k)$$

- E.g., $h(\mathbf{x}, \theta_k)$ is the k-th decision trees which gives output $\in [-1, 1]$
- The (α_k, θ_k) are to be estimated by minimizing a loss function ℓ :

$$(\alpha_k^*, \theta_k^*)_1^L = \operatorname*{arg\,min}_{\{\alpha_k, \theta_k\}_1^L} \sum_{i=1}^{n} \ell\left(y_i, \sum_{k=1}^{L} \alpha_k h_k(\mathbf{x}_i, \theta_k)\right)$$

- However, directly optimizing this loss function is often difficult

Gradient boosting

- Instead, we usually use a method called *Forward Stagewise Additive Modeling* (FSAM) :
    - Initialize $H_0(\mathsf{x}) = 0$
    - for $k = 1$ to L :
        1. Compute
        $$(\alpha_k, \theta_k) = \operatorname*{arg\,min}_{\alpha,\theta} \sum_{i=1}^{n} \ell\left(y_i, H_{k-1}(\mathsf{x}_i) + \alpha h(\mathsf{x}_i, \theta)\right)$$
        2. Set
        $$H_k(\mathsf{x}) = H_{k-1}(\mathsf{x}) + \alpha_k h(\mathsf{x}, \theta_k)$$
- Adaboost is a special case with $\ell(y, h) = e^{-yh}$

Gradient boosting

- This is somehow similar to gradient descent :

$$\theta_k = \theta_{k-1} - \eta \nabla_{\theta_{k-1}} \ell\left(y_i, h(\mathrm{x}_i, \theta_{k-1})\right)$$

  - search in the parameter space
  - update to the opposite direction of the gradient (w.r.t. the parameters)

- *Forward Stagewise Additive Modeling* :

$$\begin{aligned} H_k(\mathrm{x}) &= H_{k-1}(\mathrm{x}) - \eta \nabla_{H_{k-1}(\mathrm{x}_i)} \ell\left(y_i, H_{k-1}(\mathrm{x}_i)\right) \\ &= H_{k-1}(\mathrm{x}) - \eta \left[\frac{\partial \ell(y,h)}{\partial h}\right]_{y=y_i, h=H_{k-1}(\mathrm{x}_i)} \end{aligned}$$

  - search in the hypothesis space
  - update to the opposite direction of the gradient (w.r.t. the model)

Gradient boosting

- Gradient boosting :

$$
\begin{aligned}
H_k(\mathbf{x}) &= H_{k-1}(\mathbf{x}) - \eta \nabla_{H_{k-1}(\mathbf{x}_i)} \ell \left(y_i, H_{k-1}(\mathbf{x}_i) \right) \\
&= H_{k-1}(\mathbf{x}) + \eta h(\mathbf{x}, \theta_k)
\end{aligned}
$$

  where $h(\mathbf{x}, \theta_k)$ is learned to approximate the negative gradient
- $h(\mathbf{x}, \theta_k)$ is a regressor (even for classification tasks), most often small regression trees
- Reasons are that it leads to simplifications[2] [3] and that it gives good performances

2. J.H. Friedman, 'Greedy function approximation : a Gradient Boosting Machine', The Annals of Statistics, 2001"
3. `https://xgboost.readthedocs.io/en/stable/tutorials/model.html`

Gradient Boosting

- Inputs : $\mathcal{D}$, $\mathcal{A}$, L and η
- Output : $H_L(\mathbf{x})$ for regression or $sign(H_L(\mathbf{x}))$ for classification
- Initialization : $H_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^{n} \ell\left(y_i, \gamma\right)$ [4]
- For $k = 1$ to L :
  1. Compute :
  $$\tilde{y}_i = -\left[\frac{\partial \ell(y,h)}{\partial h}\right]_{y=y_i, h=H_{k-1}(\mathbf{x}_i)} \quad \forall (\mathbf{x}_i, y_i) \in \mathcal{D}$$
  2. Build $\mathcal{D}' = \{(\mathbf{x}_i, \tilde{y}_i)\}$, from all $(\mathbf{x}_i, y_i) \in \mathcal{D}$
  3. $h_k(\mathbf{x}) = \mathcal{A}(\mathcal{D}')$
  4. $H_k(\mathbf{x}) = H_{k-1}(\mathbf{x}) + \eta h_k(\mathbf{x})$

4. In practice, $H_0(\mathbf{x})$ is set to $\bar{y}$ for regression and to $\log\left(n_+/n_-\right)$ for classification, where n_+ (resp. n_-) is the number of training instances that belong to the positive class (resp. negative class).

Gradient Boosting

- One can use any loss provided that we can compute $\frac{\partial \ell(y,h)}{\partial h}$
- For example :
  - squared-error loss (regression) :

$$\ell(y,h) = (y-h)^2 \;\rightarrow\; \frac{\partial \ell(y,h)}{\partial h} = y - h$$

  - two-class log loss (classification) :

$$\ell(y,h) = \log(1+\exp(-2yh)) \;\rightarrow\; \frac{\partial \ell(y,h)}{\partial h} = -\frac{2y}{1+\exp(2yh)}$$

  (multiclass variant also available [5])

5. "J.H. Friedman, 'Greedy function approximation : a Gradient Boosting Machine', The Annals of Statistics, 2001"

Discussion

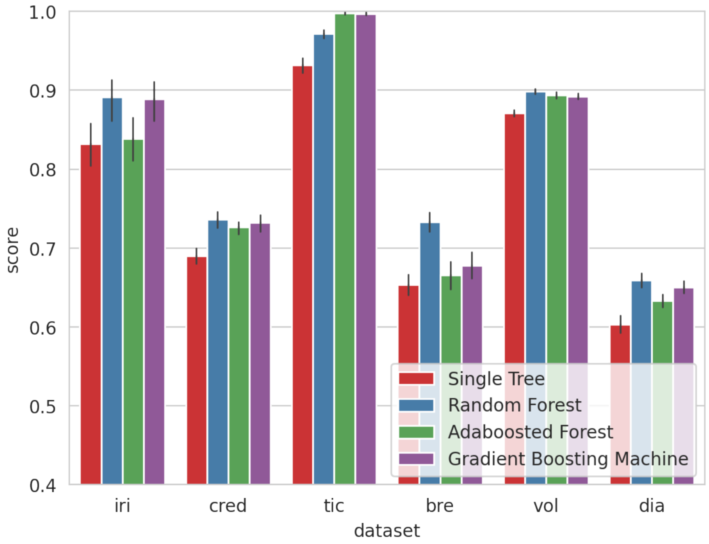
- The learning rate η allows to control the overfitting risk :
    - when η is small, error convergence is slower but overfitting is limited
    - the lower η, the higher L should be
- Number of weak classifiers
    - Depend on η and on the problem
    - No theoretical, nor empirical rules, but the more the better usually
- Decision tree depth
    - stump may be too simple (too weak), but deeper tree tends to overfit
    - Usually, AdaBoost uses small trees (depth=1,3) and GBM slightly deeper trees

Famous implementations

- XGBoost[6] : the most famous and most often winning ML technique in Kaggle competitions
- LightGBM[7] : designed for larger datasets
- CatBoost[8] : designed to handle categorical features
- Most of them are based on specific optimization tricks to make the learning procedure faster and more efficient

6. https://xgboost.ai/
7. https://lightgbm.readthedocs.io/en/stable/
8. https://catboost.ai/

Gradient Boosting



Takeaways

Pros :

- Very solid theoretical framework and numerous theoretical results/guarantees
- AdaBoost is a baseline with some interesting variants (e.g. LogitBoost)
- Gradient Boosting Machines are much more accurate and versatil
- State-of-the-art performances for tabular data

Cons :

- Computational and memory complexity
- Hyper-parameter tuning is hell
- Overfitting is still a concerns (regulation strategies may interact with each other)