Boosting

Définition du Boosting

Le **boosting** est une méthode d'ensemble visant à combiner plusieurs modèles dits *faibles* (weak learners) afin d'obtenir un modèle global plus robuste et performant.

- **But** : Transformer un apprenant faible A dont la précision est juste un peu meilleure que le hasard, en un apprenant fort très performant.
- **Principe** :
  - Générer successivement des modèles faibles, chaque nouveau modèle se concentrant davantage sur les exemples mal classés par les précédents.
  - Ajuster les poids ou résidus de chaque observation pour orienter le processus.
  - Combiner les prédictions des modèles faibles (typiquement par somme pondérée) pour obtenir une prédiction finale.

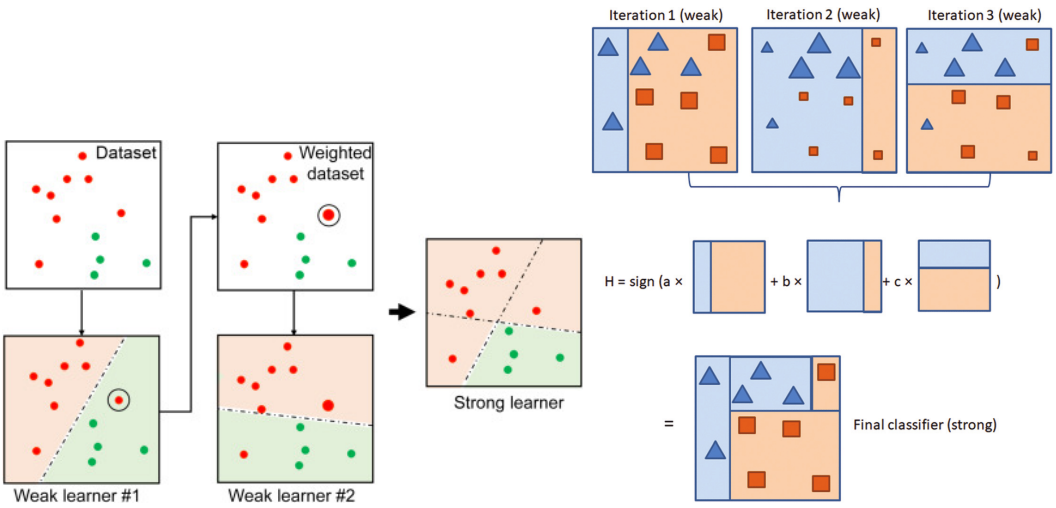
Introduction aux concepts clés

- **Espace d'hypothèses** H : Ensemble de tous les modèles (hypothèses) envisageables.
- **loss** $\ell(y, h(x))$: Mesure l'erreur commise par le modèle h sur une observation (x, y).
- **Erreur vraie (ou risque)** :
$$R(h) = \mathbb{E}_{(x,y)\sim D}[\ell(y, h(x))].$$
- **Boosting vs Bagging** :
  - **Boosting** : Les modèles sont construits séquentiellement, chaque nouveau modèle corrigeant les erreurs du précédent.
  - **Bagging** : Les modèles sont entraînés en parallèle sur des échantillons bootstrap, puis agrégés (moyenne, vote, etc.) sans prise en compte explicite des erreurs passées.

AdaBoost (Adaptive Boosting)

Objectif : Minimiser la loss exponentielle $\ell(y, h(x)) = e^{-yh(x)}$.

- **Utilisation** : Apprenant faible typique : arbres de décision peu profonds. Principalement pour la classification binaire ($y \in \{-1, 1\}$).



- **Fonctionnement** :
  1. **Initialisation** : $\forall i, w_i = \frac{1}{n}$.

Hugo Tondenier

2. **Itération** k :
   (a) Entraîner un classificateur faible h_k sur les données pondérées (x_i, y_i, w_i).
   (b) Calculer l'erreur pondérée :
   $$\epsilon_k = \sum_{i=1}^{n} w_i \mathbb{1}[y_i \neq h_k(x_i)].$$
   (c) Calculer le poids α_k du classificateur :
   $$\alpha_k = \frac{1}{2} \ln\left(\frac{1-\epsilon_k}{\epsilon_k}\right).$$
   (d) Mettre à jour les poids des observations :
   $$w_i^{(k+1)} = w_i^{(k)} \cdot e^{-\alpha_k y_i h_k(x_i)},$$
   puis normaliser $\{w_i\}$.

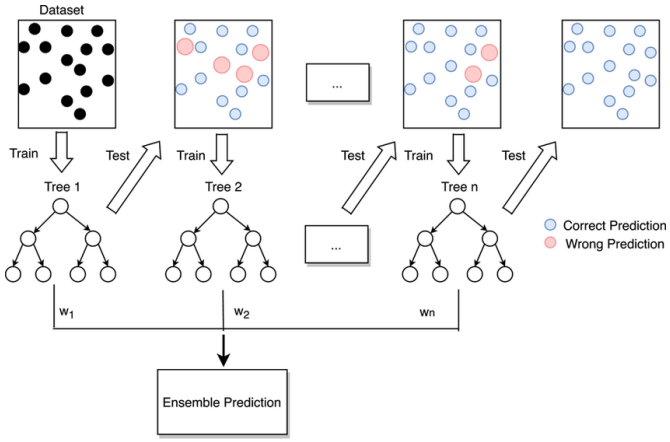
- **Combinaison finale** :
$$H(x) = \text{sign}\left(\sum_{k=1}^{L} \alpha_k h_k(x)\right).$$

- **En Pratique** :
  – Avantage : Facile à mettre en oeuvre, souvent performant pour une large gamme de problèmes.
  – Limite : Sensible aux valeurs aberrantes et au bruit, peut surapprendre si trop d'itérations.

Gradient Boosting

Objectif : Minimiser une fonction de loss générale $\ell(y, h)$ via une approche additive, en approximant la descente de gradient de la fonction de coût.

- **Principe général** : Le gradient boosting considère le modèle final comme une somme de modèles faibles :
$$H(x) = H_0(x) + \sum_{k=1}^{L} \eta h_k(x),$$
où $H_0(x)$ est un modèle initial (souvent une constante), h_k sont des apprenants faibles (typiquement des arbres de décision) et η est le taux d'apprentissage.
L'idée est d'ajouter, à chaque itération, un nouvel apprenant qui pointe dans la direction du gradient négatif de la loss afin de la réduire.



- **Détails de l'algo** :
  1. **Initialisation** :
  $$H_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} \ell(y_i, \gamma).$$
  Le modèle initial est souvent la prédiction moyenne (en régression) ou le log-odds moyen (en classification binaire).

2. **Calcul des pseudo-résidus (gradients)** : À l'itération k, on calcule pour chaque observation (x_i, y_i) le pseudo-résidu :
$$r_i^{(k)} = -\frac{\partial \ell(y_i, H_{k-1}(x_i))}{\partial H_{k-1}(x_i)}$$
Ces résidus indiquent la direction dans laquelle ajuster le modèle pour réduire la loss.
3. **Entraînement de l'apprenant faible** : On ajuste un apprenant faible $h_k(x)$, généralement un arbre de régression, pour prédire les pseudo-résidus $\{r_i^{(k)}\}$. L'objectif est d'obtenir $h_k(x)$ tel que :
$$h_k = \arg\min_{h \in H} \sum_{i=1}^{n} (r_i^{(k)} - h(x_i))^2$$
(ou un critère adapté en fonction du type de loss).
4. **Recherche de l'optimum** γ_k : Une fois $h_k(x)$ construit, on peut affiner la mise à jour du modèle en cherchant un coefficient γ_k (si nécessaire) qui minimise la loss :
$$\gamma_k = \arg\min_{\gamma} \sum_{i=1}^{n} \ell(y_i, H_{k-1}(x_i) + \gamma h_k(x_i)).$$
5. **Mise à jour du modèle** : Le modèle est alors mis à jour comme :
$$H_k(x) = H_{k-1}(x) + \eta \gamma_k h_k(x),$$
où η est le taux d'apprentissage (learning rate) qui contrôle la vitesse de convergence et permet d'éviter le surapprentissage.

- **Fonctions de loss courantes** :
  - Régression : $\ell(y, h) = (y - h)^2$.
  - Classification binaire : Log Loss $\ell(y, h) = \log(1 + e^{-2yh})$.
  - Toute loss différentiable : le gradient boosting est très flexible.
- **Avantages et limites** :
  - Avantages :
    - **Flexibilité** : Utilise différentes fonctions de loss et types d'apprenants faibles.
    - **Performance élevée** sur données tabulaires, souvent utilisé en pratique (XGBoost, LightGBM, CatBoost).
  - Limites :
    - **Complexité** : Le tuning des hyperparamètres (taux d'apprentissage, profondeur des arbres, nombre d'itérations) est crucial.
    - **Temps de calcul** : Peut être long sur de grands ensembles de données sans implémentation optimisée.
    - **Surapprentissage** : Possible si le modèle n'est pas régularisé ou si le taux d'apprentissage est trop élevé.

Comparaison AdaBoost vs Gradient Boosting

Critères	AdaBoost	Gradient Boosting
Approche théorique	Minimise la loss exponentielle	Minimise une loss différentiable générale
Gestion des erreurs	Ajuste les poids des observations	Corrige via les gradients (pseudo-résidus)
Flexibilité	Principalement classification binaire	Supporte régression, classification, etc.
Sensibilité au bruit	Plus sensible	Moins sensible (avec bon tuning)

Conseils pour utiliser le Boosting

- **Hyperparamètres** :
  - **AdaBoost** :
    - Limiter le nombre d'itérations L.
    - Utiliser des apprenants faibles simples (ex. petits arbres).
  - **Gradient Boosting** :
    - Choisir un **taux d'apprentissage (η)** faible (ex. 0.1 ou moins).
    - Limiter la **profondeur des arbres** (3-6).
    - Ajuster le nombre d'itérations et utiliser l'*early stopping*.
    - Ajouter de la régularisation (par ex. pénaliser les feuilles trop nombreuses).
- **Implémentations avancées** : XGBoost, LightGBM, CatBoost offrent des gains en vitesse et en performance.
- **Validation** :
  - Utiliser une validation croisée ou un jeu de validation distinct.
  - Surveiller l'erreur de validation pour détecter le surapprentissage.