Decision Trees

Quels types de nodes ?

- **root node** : c'est le premier noeud, représentant l'ensemble complet des données. À ce stade, on n'a pas encore fait de séparation
- **decision node** : ce sont les noeuds intermédiaires qui contiennent une question sur une feature. Chaque fois qu'on split, on crée un nouveau decision node, tant qu'il reste des données à séparer.
- **leaf/terminal node** : ce sont les noeuds qui n'ont pas d'enfants. Ils représentent la prédiction finale (pour un problème de classification, ce sera une classe ; pour un problème de régression, ce sera une valeur).

Pre/Post Pruning

Une fois l'arbre entièrement construit, on peut constater qu'il est trop "profond" et qu'il overfit.

On veut retirer certains noeuds (des decision nodes très bas dans l'arbre) pour simplifier l'arbre et améliorer sa capacité de généralisation.

pre-pruning : arrêter la croissance de l'arbre avant qu'il n'atteigne une profondeur trop importante, par exemple en fixant une profondeur maximale, un nombre minimal d'échantillons dans un noeud pour pouvoir le scinder, etc.

post-pruning : construire l'arbre jusqu'au bout (potentiellement très complexe), puis retirer certains sous-arbres qui n'apportent pas suffisamment de gain (en se basant sur un critère de validation).

La difficulté est de choisir les décisions rules. Pour ça, on va itérer sur toutes les décisions rules possibles, et voir l'information gagnée.

Comment mesurer l'information gagnée ?

On peut utiliser une mesure comme Gini impurity qui va mesurer à quel point notre groupe est composée d'une classe unique (est pur).

Ginity impurity, mesure l'impurité d'un noeud

$$\text{Gini impurity } = 1 - \sum_{k=1}^{C} p_k^2$$

où C est le nombre de classes et p_k la proportion d'exemples de la classe k dans ce noeud.

- si tous les échantillons sont de la même classe, alors $p_k = 1$ pour une classe et 0 pour les autres, donc l'impurité vaut 0.
- si le noeud contient 50% d'une classe et 50% d'une autre, alors $1 - 0.5^2 - 0.5^2 = 0.5$.
- plus Gini est proche de zéro, plus le noeud est pur.

Shannon impurity

$$\text{Shannon impurity } = -\sum_{k=1}^{K} p_k \log_2 p_k$$

Comment l'utiliser pour faire un choix ?

Pour chaque division/critère possible, on calcule le gain :

$$\text{Gain } = \text{ impurity du parent } - \left(\frac{N_{\text{gauche}}}{N_{\text{parent}}} \cdot \text{ impurity gauche } + \frac{N_{\text{droite}}}{N_{\text{parent}}} \cdot \text{ impurity droite} \right)$$

Random Forests

L'idée

On veut construire un **ensemble (un "bag") d'arbres de décision** construits sur des **sous-échantillons aléatoires** des données, des **bootstrapped dataset**, (avec remplacement!) puis les faire voter pour créer un strong classifier.

Original Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease

To create a bootstrapped dataset that is the same size as the original, we just randomly select samples from the original dataset.

The important detail is that we're allowed to pick the same sample more than once.

Attention à bien sélectionner **avec replacement**, sinon tous les arbres se ressembleraient et on ne gagnerait pas beaucoup.

Chaque arbre est différent car :

- Il est entraîné sur un jeu de données différent. (**bootstrapped dataset**)
- À chaque nœud, plutôt que sélectionner toutes les features possibles, on ne considère qu'un **sous-ensemble aléatoire de features** (par exemple $\sqrt{d}$ au lieu de d).

Pourquoi c'est mieux ?

- **Moins d'overfitting** que les arbres seuls.
- **Robuste** aux bruits.
- Peut estimer la probabilité d'appartenance à une classe.

Pour classifier, on demande à chaque tree de classifier et on prend le plus de votes. On aggrege les données. **Bootstrapped dataset + aggregation = bagging**.

Les données qui ne finissent dans aucun arbre s'appellent des **Out-of-Bag Datasets**. On peut mesurer l'accuracy de notre modèle en utilisant les out-of-bags dataset (on fait prédire à notre arbre les valeurs de out of bags).

Gradient Boosted Trees (GBT, ou XGBoost, LightGBM)

On ajoute **des arbres faibles (très peu profonds, par exemple 3-4 de profondeurs) un par un** pour corriger les erreurs du modèle précédent.

> ⓘ **à la différence de la random forest, GBT construit donc les arbres séquentiellement.**

Pour de la régression :

- On dit que le premier arbre T_1 va renvoyer au début la valeur moyenne des données et on obtient des "prédictions" bêtes $\hat{y}_1$.
- On calcule des résidus (pour chaque point, de combien se trompe l'arbre précédent, $r = y_{\text{true}} - y_{\text{pred}}$).
- et ensuite, on entraîne un autre arbre pour prédire ces résidus, et $y_{\text{étape 2}} = y_1 + \eta \cdot T_2(x)$, où η est le learning rate, ou shrinkage.
- etc. on répète et à la fin on a :

$$\hat{y}(x) = \sum_{m=1}^{M} \eta \cdot T_m(x)$$

Pour de la classification (multi-classe, K classes):

- Généralement, on commence par calculer la proportion de chaque classe, puis on assigne à $p_{\text{classe k}}(x) = \text{ proportion de la classe k}$.
- Ensuite, à chaque itération, on construit K arbres pour corriger les pseudo-residus de chaque classe.
  - Pour chaque sample i et chaque classe k, on calcule:$r_{i,k} = I\left(y_i = k\right) - p_{i,k}$.
  - Pour chaque classe k, on créé un arbre faible qui va cibler $(x_i, r_{i,k})$
  - On met à jour les probabilités pour chaque sample et classe i, k.

$$\hat{y}(x) = \arg\max_{\text{k = 1,..,K}} \hat{p}_k(x)$$