Project FTML

Guillaume CARRIERE, Romain GREGOIRE, Alex POIRON, Tom THIL

16 juin 2022

1 Bayes estimator and Bayes risk

1.1 Question 1

Pour cette question, nous allons prendre le setting suivant :

- $\mathcal{X} = [0, 3]$
- $\mathcal{Y} = \mathbb{R}$
- $X \sim \mathcal{U}(\mathcal{X})$
- $Y = \begin{cases} \mathcal{U}[0,1] & \text{si } X < 1 \\ \mathcal{N}(X) & \text{si } X > 1 \end{cases}$

Dans ce contexte, en utilisant le **square loss**, nous avons donc comme **Bayes predictor** :

$$f^*(x) = \begin{cases} \frac{1}{2} & \text{si } X < 1 \\ X & \text{si } X > 1 \end{cases}$$

Le **Bayes Risk** associé équivaut à

$$\begin{aligned} R^* &= \mathbb{E}[l(Y, f^*(x))] \\ &= \frac{1}{3}\mathbb{E}[(\mathcal{U}[0,1] - \frac{1}{2})^2] + \frac{2}{3}\mathbb{E}[(\mathcal{N}(x,1) - x)^2] \end{aligned}$$

On calcule chacune de ces espérances :

$$\begin{aligned} \mathbb{E}[(\mathcal{U}[-\frac{1}{2}, \frac{1}{2}])^2] &= \mathbb{E}[(\mathcal{U}[0, \frac{1}{2}])]^2 \\ &= (\frac{1}{4})^2 \\ &= \frac{1}{8} \\ \mathbb{E}[(\mathcal{N}(x,1) - x)^2] &= \mathbb{E}[(\mathcal{N}[0,1])^2] \\ &= \mathbb{V}(\mathcal{N}[0,1]) \\ &= 1 \end{aligned}$$

Au final, nous avons

$$\begin{aligned} R^* &= \frac{1}{3} \times \frac{1}{8} + \frac{2}{3} \times 1 \\ &= \frac{1}{24} + \frac{2}{3} \\ &\approx 0.70 \end{aligned}$$

1.2 Question 2

Toujours avec ce setting choisi, nous proposons comme estimateur

$$\widetilde{f} = \begin{cases} \mathcal{X} \to \mathcal{Y} \\ x \to X \end{cases}$$

Après avoir lancé une simualtion, on obtient comme valeur pour notre estimateur $\widetilde{f} \approx 0.77$, ce qui est logiquement légèrement au-dessus du **Bayes Risk**

2 Bayes risk with absolute loss

2.1 Question 1

Prenons le setting suivant :

- $\mathcal{X} = [0,1]/\frac{1}{2}$
- $\mathcal{Y} = \mathbb{R}$
- $X \sim \mathcal{U}(\mathcal{X})$
- $Y \sim \mathcal{B}(X)$

Si dans ce contexte on prend le **square loss**, on obtient comme **Bayes predictor** :

$$f^*(x) = X$$

Pour l'**absolute loss**, on obtient :

$$f^*(x) = \begin{cases} 0 & \text{si } X < \frac{1}{2} \\ 1 & \text{si } X > \frac{1}{2} \end{cases}$$

2.2 Question 2

On a

$$\begin{aligned} f^*(x) &= \operatorname{argmin} \mathbb{E}[|y - z| | X = x] \\ &= \operatorname{argmin}(\int |y - z| P_{Y|X=x}(y) dy) \\ &= \operatorname{argmin}(g(z)) \end{aligned}$$

Développons $g(z)$

$$\begin{aligned} g(z) &= \int |y - z| P_{Y|X=x}(y) dy \\ &= \int_{-\infty}^{z} (z - y) P_{Y|X=x}(y) dy + \int_{z}^{+\infty} (y - z) P_{Y|X=x}(y) dy \\ &= z \int_{-\infty}^{z} P_{Y|X=x}(y) dy - \int_{-\infty}^{z} y P_{Y|X=x}(y) dy + \int_{z}^{+\infty} y P_{Y|X=x}(y) dy - z \int_{z}^{+\infty} P_{Y|X=x}(y) dy \end{aligned}$$

Pour chercher le minimum, nous devons regarder à l'endroit ou la derivée s'annule, c'est à dire quand $g'(z) = 0$:

$$\begin{aligned} g'(z) &= \int_{-\infty}^{z} P_{Y|X=x}(y) dy - \int_{z}^{+\infty} P_{Y|X=x}(y) dy \\ g'(z) = 0 &\leftrightarrow \int_{-\infty}^{z} P_{Y|X=x}(y) dy = \int_{z}^{+\infty} P_{Y|X=x}(y) dy \end{aligned}$$

On obtient ce résultat quand chaque côté de cette équation vaut $\frac{1}{2}$ puisque la somme de ces deux intégrales doit faire 1. Cela signifie que le Bayes predictor est égal à la médiane.

3 Expected value of empirical risk

3.1 Step 1

Montrons que $\mathbb{E}[R_n(\hat{\theta})] = \mathbb{E}_\epsilon[\frac{1}{n}\|(I_n X(X^T X)^{-1}X^T)\epsilon\|^2]$

$$\begin{aligned}
\mathbb{E}[R_n(\hat{\theta})] &= \mathbb{E}[\frac{1}{n}\|y - X\hat{\theta}\|_2^2] \\
&= \mathbb{E}[\frac{1}{n}\|y - X(X^T X)^{-1}X^T y\|^2] \\
&= \mathbb{E}[\frac{1}{n}\|(I_n - X(X^T X)^{-1}X^T)y\|^2] \\
&= \mathbb{E}_\epsilon[\frac{1}{n}\|(I_n - X(X^T X)^{-1}X^T)(X\theta^* + \epsilon)\|^2] \\
&= \mathbb{E}_\epsilon[\frac{1}{n}\|X\theta^* - X(X^T X)^{-1}X^T X\theta^* + (I_n - X(X^T X)^{-1}X^T)\epsilon)\|^2]
\end{aligned}$$

Or, $(X^T X)^{-1}X^T X = I_n$. On se retrouve au final avec

$$\mathbb{E}[R_n(\hat{\theta})] = \mathbb{E}_\epsilon[\frac{1}{n}\|X\theta^* - X\theta^* + (I_n - X(X^T X)^{-1}X^T)\epsilon)\|^2]$$

Donc

$$\mathbb{E}[R_n(\hat{\theta}) = \mathbb{E}_\epsilon[\frac{1}{n}\|(I_n X(X^T X)^{-1}X^T)\epsilon\|^2]$$

3.2 Step 2

$A \in \mathbb{R}^{n,n}$, montrons que $\sum_{(i,j\in[1,n]^2)} A_{i,j}^2 = \mathrm{tr}(A^T A)$

Par définition de la transposée, on a $A_{i,j}^T = A_{j,i}$

Par définition du produit matriciel, on a $A^T A_{i,j} = \sum_{k=1}^{n} A_{j,k}A_{i,k}$

On a donc

$$A^T A_{i,i} = \sum_{j=1}^{n} A_{i,j}A_{i,j} = \sum_{j=1}^{n} A_{i,j}^2$$

Or, $\mathrm{tr}(A^T A) = A^T A_{i,i}$, ce qui nous donne au final

$$\sum_{(i,j\in[1,n]^2)} A_{i,j}^2 = \mathrm{tr}(A^T A)$$

3.3 Step 3

Montrons que $E_\epsilon[\frac{1}{n}\|A\epsilon\|^2] = \frac{\sigma^2}{n} tr(A^T A)$

$$\begin{aligned} E_\epsilon[\frac{1}{n}\|A\epsilon\|^2] &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{n}\|A\epsilon_i\|^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{n}\epsilon_i\|A\|^2 \\ &= \frac{1}{n}\sigma^2 tr(ATA) \end{aligned}$$

Donc

$$E_\epsilon[\frac{1}{n}\|A\epsilon\|^2] = \frac{\sigma^2}{n} tr(A^T A)$$

3.4 Step 4

On a $A = I_n - X(X^T X)^{-1}X^T$. Montrons que $A^T A = A$

$$\begin{aligned} A^T A &= (I_n - X(X^T X)^{-1}X^T)^T A \\ &= (I_n^T - (X(X^T X)^{-1}X^T)^T)A \\ &= (I_n^T - X^{T^T}(X(X^T X)^{-1})^T)A \\ &= (I_n^T - X((X^T X)^{-1})^T X^T)(I_n - X(X^T X)^{-1}X^T) \end{aligned}$$

On développe cette expression

$$A^T A = I_n^T I_n - X(X^T X)^{-1}X^T - X((X^T X)^{-1})^T X^T + X((X^T X)^{-1})^T X^T X(X^T X)^{-1}X^T$$

Or $I_n^T I_n = I_n$ et $X^T X(X^T X)^{-1} = I_n$

$$A^T A = I_n - X(X^T X)^{-1}X^T - X((X^T X)^{-1})^T X^T + X((X^T X)^{-1})^T X^T$$

On reduit l'expression pour obtenir

$$\begin{aligned} A^T A &= I_n - X(X^T X)^{-1}X^T \\ A^T A &= A \end{aligned}$$

3.5 Step 5

Pour rappel, on souhaite prouver que $E[R_X(\hat{\theta})] = \frac{n-d}{n}\sigma^2$

$$E[R_X(\hat{\theta})] = E_\epsilon[\frac{1}{n}\|(I_n X(X^T X)^{-1}X^T)\epsilon\|^2]$$

On prend $A = I_n - X(X^T X)^{-1}X^T$

$$\begin{aligned} E[R_X(\hat{\theta})] &= E_\epsilon[\frac{1}{n}\|A\epsilon\|^2] \\ &= \frac{\sigma^2}{n} tr(A^T A) \\ &= \frac{\sigma^2}{n} tr(A) \\ &= \frac{\sigma^2}{n}(tr(I_n) - tr(X(X^T X)^{-1}X^T)) \end{aligned}$$

Or $tr(X(X^TX)^{-1}X^T) = tr(X^TX(X^TX)^{-1})$

$$E[R_X(\hat{\theta})] = \frac{\sigma^2}{n}(tr(I_n) - tr(X^TX(X^TX)^{-1}))$$

On sait que $X \in \mathbb{R}^{n,d}$ donc le produit matriciel $XTX \in \mathbb{R}^{d,d}$. Ce qui fait que $X^TX(X^TX)^{-1} = I_d$

$$\begin{aligned} E[R_X(\hat{\theta})] &= \frac{\sigma^2}{n}(tr(I_n) - tr(I_d)) \\ &= \frac{\sigma^2}{n}(n-d) \end{aligned}$$

Donc

$$E[R_X(\hat{\theta})] = \frac{n-d}{n}\sigma^2$$

3.6 Step 6

On veut calculer $E[\frac{||y-X\hat{\theta}||_2^2}{n-d}]$

On remarque que $E[\frac{||y-X\hat{\theta}||_2^2}{n-d}] = E[R_{n-d}(\hat{\theta})]$. Au vu des résultats précédents et dans ce même contexte, on peut dire que

$$\begin{aligned} E[R_{n-d}(\hat{\theta})] &= \frac{n-d}{n-d}\sigma^2 \\ &= \sigma^2 \end{aligned}$$

Donc

$$E[\frac{||y-X\hat{\theta}||_2^2}{n-d}] = \sigma^2$$

3.7 Step 7

Ce qu'on peut conclure avec la simulation, c'est qu'en choissisant un σ^2 tel que

$$\sigma^2 = 3$$

Nous pouvons approximer cette valeur en la retrouvant telle que

$$\begin{aligned} \sigma^2 &= V(\epsilon) \\ &\approx 2.97 \end{aligned}$$

4 Regression

4.1 Preprocessing

Avant d'appliquer une régression aux données, nous avons d'abord appliqué quelques traitements à celles-ci. Nous avons normalisé les inputs. La normalisation standardise la moyenne et l'écart-type de tout type de distribution de données, ce qui permet de simplifier le problème d'apprentissage en s'affranchissant de ces deux paramètres.

Pour effectuer cette transformation, on soustrait aux données leur moyenne empirique et on les divise par leur écart-type σ.

$$X_{normalise} = \frac{X-m}{\sigma}$$

Pour calculer et estimer si la régression utilisée est efficace, nous avons de plus séparé les données en un dataset de train et un de test.

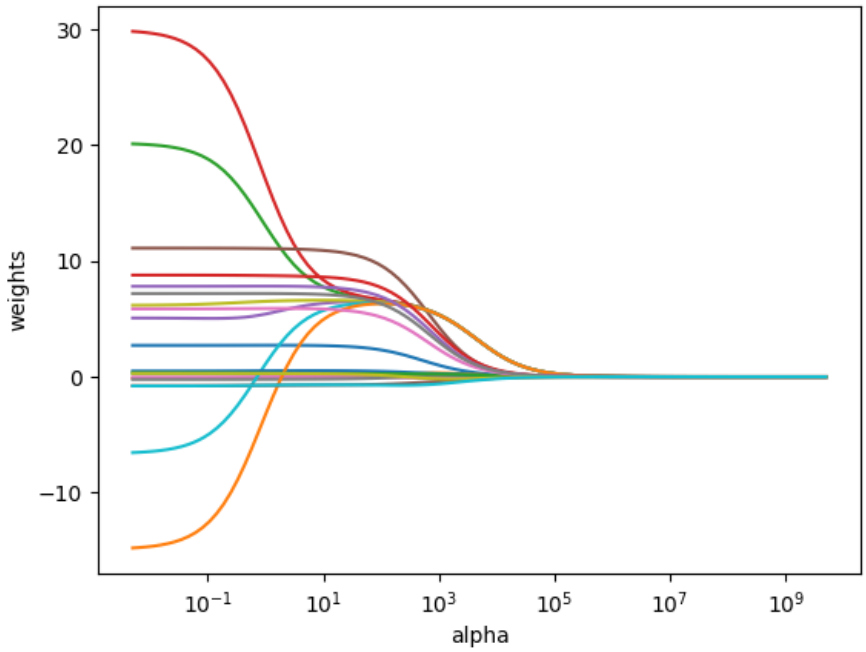

FIGURE 1 – Ridge Regression

4.2 Ridge Regression

La régression de Ridge est une méthode d'estimation des coefficients des modèles de régression multiple dans des scénarios où les variables linéairement indépendantes sont fortement corrélées.

Nous avons généré un tableau de valeurs alpha allant de très grand à très petit, couvrant essentiellement la gamme complète de scénarios, du modèle simple contenant uniquement l'interception, à l'ajustement des moindres carrés.

Sans utiliser pour l'instant de fonction optimisation, on obtient le graphique suivant :

4.3 Optimisation

4.3.1 K-fold

La procédure de validation croisée k-fold est une méthode standard pour estimer les performances d'un algorithme ou d'une configuration d'apprentissage automatique sur un ensemble de données. Cela implique simplement de répéter plusieurs fois la procédure de validation croisée et de rapporter le résultat moyen de toutes les exécutions. Ce résultat moyen devrait être une estimation plus précise de la véritable performance moyenne du modèle sur l'ensemble de données.

4.3.2 Ridge Cross-Validation

Pour optimiser l'alpha et trouver celui qui est le plus pertinent pour apprendre dans notre cas, nous avons utilisé la fonction RidgeCV de scikit-learn et avons utilisé le KFold comme fonction de validation croisé.

4.4 Résultat

Avec toutes les optimisations et le tuning des hyperparamètres, en moyenne nous arrivons à un résultat de 0.884 R^2 pour la régression de Ridge et au maximum 0.907.

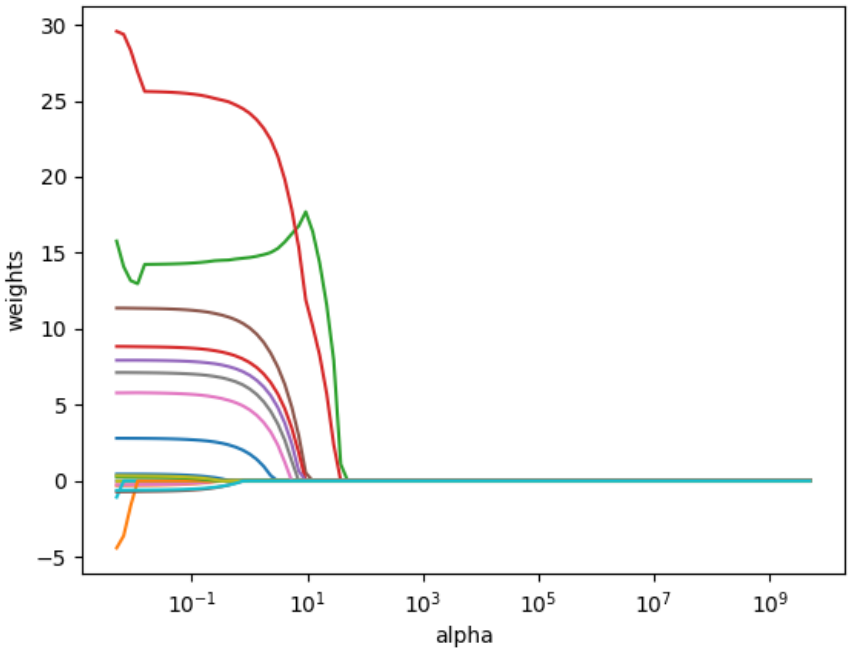

FIGURE 2 – Lasso Regression

4.5 Lasso Regression

La régression Lasso est un type de régression linéaire qui utilise le rétrécissement. Le rétrécissement est l'endroit où les valeurs des données sont réduites vers un point central, comme la moyenne. La procédure de Lasso encourage les modèles simples et clairsemés (c'est-à-dire les modèles avec moins de paramètres). Ce type particulier de régression est bien adapté aux modèles présentant des niveaux élevés de colinéarité.

Nous avons généré un tableau de valeurs alpha allant de très grand à très petit, couvrant essentiellement la gamme complète de scénarios, du modèle simple contenant uniquement l'interception, à l'ajustement des moindres carrés.

Sans utiliser pour l'instant de fonction optimisation, on obtient le graphique suivant :

4.6 Optimisation

4.6.1 K-fold

La procédure de validation croisée k-fold est une méthode standard pour estimer les performances d'un algorithme ou d'une configuration d'apprentissage automatique sur un ensemble de données. Cela implique simplement de répéter plusieurs fois la procédure de validation croisée et de rapporter le résultat moyen de toutes les exécutions. Ce résultat moyen devrait être une estimation plus précise de la véritable performance moyenne du modèle sur l'ensemble de données.

4.6.2 Lasso Cross-Validation

Pour optimiser l'alpha et trouver celui qui est le plus pertinent pour apprendre dans notre cas, nous avons utilisé la fonction LassoCV de scikit-learn et avons utilisé le KFold comme fonction de validation croisé.

4.7 Résultat

Avec toutes les optimisations et le tuning des hyperparamètres, en moyenne nous arrivons à un résultat de 0.862 R^2 pour la régression de Lasso et au maximum 0.91. En optimisant et ayant un peu de chance, la régression de Lasso a été légèrement meilleure par rapport à Ridge.

5 Classification

5.1 Preprocessing

Avant d'appliquer une régression aux données, nous avons d'abord appliqué quelques traitements à celles-ci. Nous avons normalisé les inputs. La normalisation standardise la moyenne et l'écart-type de tout type de distribution de données, ce qui permet de simplifier le problème d'apprentissage en s'affranchissant de ces deux paramètres.

Pour effectuer cette transformation, on soustrait aux données leur moyenne empirique et on les divise par leur écart-type σ.

$$X_{normalise} = \frac{X - m}{\sigma}$$

Pour calculer et estimer si l'algorithme de classification utilisé est efficace, nous avons de plus séparé les données en un dataset de train et un de test.

5.2 Decision Tree

L'algorithme Decision Tree appartient à la famille des algorithmes d'apprentissage supervisé. Contrairement à d'autres algorithmes d'apprentissage supervisé, l'algorithme d'arbre de décision peut également être utilisé pour résoudre des problèmes de régression et de classification.

L'objectif de l'utilisation d'un arbre de décision est de créer un modèle de formation qui peut être utilisé pour prédire la classe ou la valeur de la variable cible en apprenant des règles de décision simples déduites de données antérieures (données de formation).

Dans les arbres de décision, pour prédire une étiquette de classe pour un enregistrement, nous partons de la racine de l'arbre. Nous comparons les valeurs de l'attribut racine avec l'attribut de l'enregistrement. Sur la base de la comparaison, nous suivons la branche correspondant à cette valeur et sautons au nœud suivant.

5.3 Optimisation

5.3.1 Entropy

Il existe de nombreuses façons d'implémenter la mesure d'impureté, scikit-learn ont implémenté deux façons de la calculer, le gain d'information et l'impureté de Gini ou l'indice de Gini.

Le gain d'information utilise la mesure d'entropie comme mesure d'impureté et divise un nœud de telle sorte qu'il donne le plus de gain d'information. Alors que l'impureté de Gini mesure les divergences entre les distributions de probabilité des valeurs de l'attribut cible et divise un nœud de telle sorte qu'il donne le moins d'impuretés.

Nous avons utilisé l'entropie qui nous donnait de meilleurs resultats que Gini dans notre cas.

5.4 Résultat

Avec toutes les optimisations et le tuning des hyperparamètres, en moyenne nous n'arrivons pas au résultat de 0.85 d'accuracy demandé pour la classification mais arrivons à un maximum de 0.877. Nous avons pas exploré plus loin cette classification au vu du peu de résultat qu'elle nous a apporté.

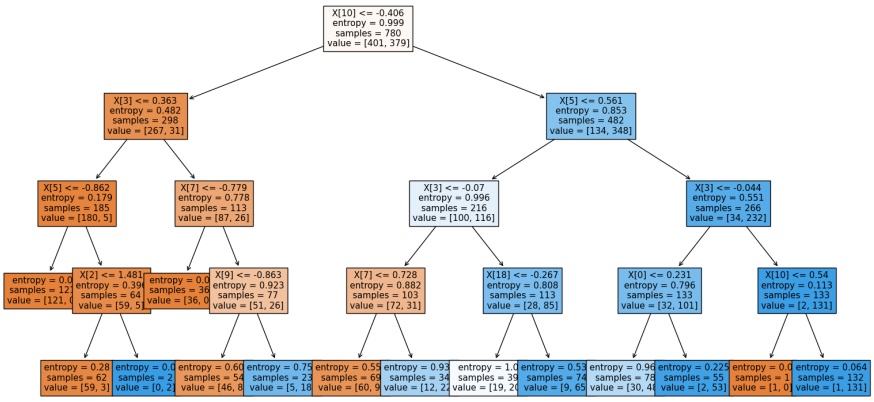

FIGURE 3 – Decision Tree

5.5 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) est un algorithme d'optimisation simple mais efficace utilisé pour trouver les valeurs des paramètres/coefficients des fonctions qui minimisent une fonction de coût. En d'autres termes, il est utilisé pour l'apprentissage de classificateurs linéaires avec des fonctions de loss convexes telles que SVM et la régression logistique.

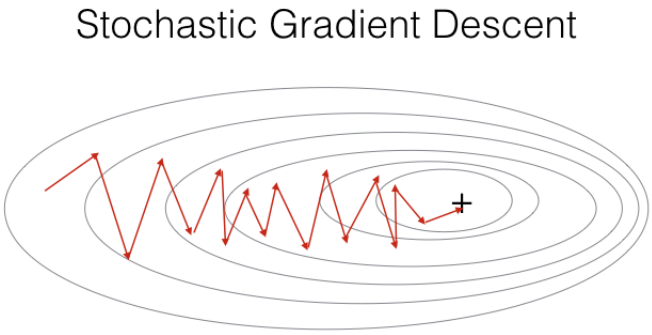

FIGURE 4 – Stochastic Gradient Descent

5.6 Optimisation

5.6.1 Hinge

Dans l'apprentissage automatique, la Hinge loss est une fonction de perte utilisée pour les classificateurs. La hinge loss est utilisée pour la classification "Maximal-Margin", notamment pour les machines à vecteurs de support (SVM).

Pour une sortie prévue t = ± 1 et un score de classificateur y , la hinge loss de la prédiction y est définie comme

l(y) = max (0 , 1 - t . y)

Nous avons utilisé cette fonction pour tuner nos hyperparamètres, nous donnant de meilleurs résultats que les autres proposées par scikit-learn.

5.6.2 Hyperparamètre

Les paramètres que nous avons fine-tuner sont pour cette fonction alpha, la tolérance et le nombre maximal d'itérations pour l'entraînement.

5.7 Résultat

Avec la fonction d'optimisation et le tuning des hyperparamètres, en moyenne nous arrivons à un résultat de 0.864 d'accuracy et arrivons à un maximum de 0.917.

5.8 Gradient Boosting

Le Gradient Boosting construit un modèle additif de manière progressive. Il permet l'optimisation de fonctions de perte différentiables arbitraires. À chaque étape, n arbres de régression sont ajustés sur le gradient négatif de la fonction de perte. La classification binaire est un cas particulier où un seul arbre de régression est induit.

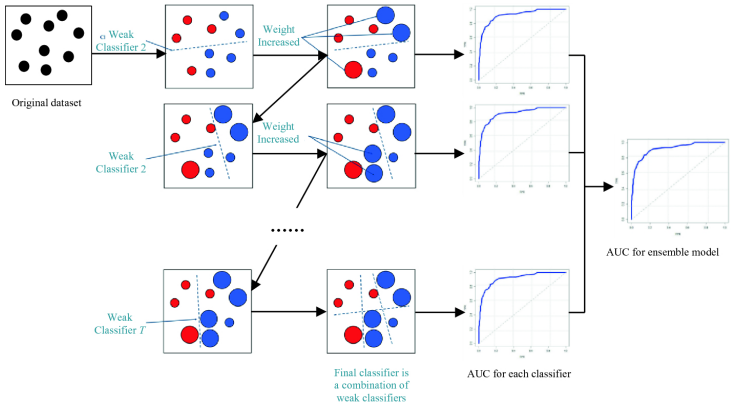

FIGURE 5 – Gradient Boosting

5.9 Optimisation

5.9.1 Log loss

La log loss indique à quel point la probabilité de prédiction est proche de la valeur réelle/vraie correspondante (0 ou 1 en cas de classification binaire). Plus la probabilité prédite s'écarte de la valeur réelle, plus la valeur de perte logarithmique est élevée. C'est la meilleure dans notre cas pour de la classification.

5.9.2 Friedman mean square error

Pour notre gradient boosting, nous utilisons la Friedman mse qui est la plus optimale pour le gradient boosting. Friedman a proposé qu'à chaque itération de l'algorithme, un apprenant de base soit ajusté sur un sous-échantillon de l'ensemble d'apprentissage tiré au hasard sans remise. Cela augmente substantiellement la descente de gradient par rapport à une mean square error.

5.10 Résultat

Avec toutes les optimisations et le tuning des hyperparamètres, comme le learning rate de l'algorithme, en moyenne nous arrivons à un résultat de 0.884 R^2 pour le Gradient Boosting et à un maximum 0.927. En optimisant et en moyenne, le Gradient Boosting est le meilleur algorithme que nous avons utilisé.