École Centrale de Nantes

PROJET : Option Mathématiques et applications

Hyperparameter Optimization

Élèves :
Youssef SALHI
Saâd AZIZ ALAOUI

Encadrants :
Bertrand MICHEL

31 janvier 2024

Table des matières

1 Introduction

L'optimisation d'hyperparamètres est un processus essentiel en apprentissage automatique, visant à déterminer les meilleures valeurs des hyperparamètres d'un modèle pour maximiser ses performances. Ces hyperparamètres, préalablement définis avant l'entraînement, ne sont pas appris à partir des données. En effet, les **paramètres** sont inhérents au modèle et sont appris au cours de la formation. Par exemple, dans un modèle de régression linéaire, les coefficients de chaque caractéristique sont les paramètres du modèle. Les **hyperparamètres** quant à eux sont des paramètres externes que nous décidons avant d'entraîner le modèle. Les exemples incluent le taux d'apprentissage dans un réseau neuronal ou la profondeur d'un arbre de décision. Leur ajustement impacte significativement la précision des prédictions et la capacité de généralisation du modèle. De plus, l'importance de cette tâche est en constante croissance puisque les modèles de Machine Learning deviennent de plus en plus complexes avec un nombre de paramètres croissants. Les réseaux neuronaux profonds, par exemple, dépendent d'un large éventail de choix d'hyperparamètres concernant leur architecture, leur régularisation et leur optimisation. Différentes stratégies, telles que le Grid Search, le Random Search et l'Optimisation Bayésienne, sont employées pour naviguer efficacement dans l'espace des hyperparamètres, chacune adaptée aux spécificités du problème et aux ressources disponibles.

2 Etat de l'art

Dans notre exploration, nous allons étudier des méthodes classiques d'optimisation d'hyperparamètres, notamment le Grid Search et le Random Search. Ces approches représentent le point de départ pour affiner les modèles d'apprentissage automatique, offrant un équilibre entre simplicité et efficacité pour identifier les meilleurs combinaisons d'hyperparamètres. Nous aurions pu évoquer d'autres méthodes telles que le **Successive Halving** ou tout simplement le **Manual Search**, mais nous présentons les plus utilisées.

2.1 Grid Search

Le *Grid Search* est une approche systématique qui explore un ensemble prédéfini de combinaisons d'hyperparamètres. C'est la manière traditionnelle la plus naturelle et exhaustive quand on s'intéresse à ce genre de problématiques. Supposons que nous ayons un ensemble d'hyperparamètres

$$\Theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$$

à optimiser. Chaque hyperparamètre peut prendre des valeurs dans un ensemble discret. Le Grid Search divise l'espace des hyperparamètres en une grille, où chaque dimension correspond à un hyperparamètre, et chaque point dans la grille représente une combinaison unique d'hyperparamètres. Soit

$$S = \{\theta_1^{(1)}, \theta_2^{(1)}, \ldots, \theta_n^{(1)}\}, \{\theta_1^{(2)}, \theta_2^{(1)}, \ldots, \theta_n^{(1)}\}, \ldots, \{\theta_1^{(m)}, \theta_2^{(m)}, \ldots, \theta_n^{(m)}\}$$

l'ensemble des combinaisons possibles. C'est l'évaluation cartésienne du produit du set défini. Cette méthode souffre du fléau de la dimension car le nombre d'évaluations de fonctions requis croît de manière exponentielle avec la dimensionnalité de l'espace de

configuration. Un autre problème du *Grid Search* est que l'augmentation de la résolution de la discrétisation augmente considérablement le nombre requis d'évaluations de fonctions.

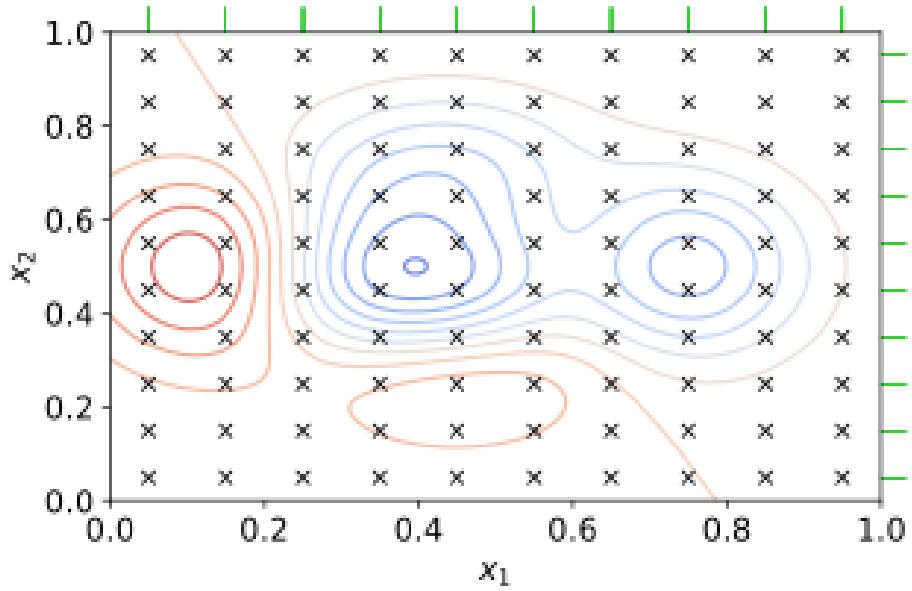


FIGURE 1 – Grid Search sur différentes valeurs de deux hyperparamètres. Pour chacun 10 valeurs différentes sont considérées, soit un total de 100 combinaisons. Les contours bleus indiquent les régions où les résultats sont bons, tandis que les contours rouges indiquent les régions où les résultats sont mauvais. Source : WIkipedia

Pour chaque combinaison d'hyperparamètres, le modèle est entraîné sur l'ensemble d'entraînement, puis évalué sur un ensemble de validation ou à l'aide d'une métrique de performance spécifiée $\mathcal{L}$. La fonction de perte ou la métrique de performance est souvent définie comme $\mathcal{L}(\theta)$ où θ représente l'ensemble des hyperparamètres.

$$\mathcal{L}(\Theta) = \text{Performance du modèle sur l'ensemble de validation}$$

L'objectif du Grid Search est de trouver la combinaison d'hyperparamètres $\mathcal{L}(\Theta)^*$ qui minimise ou maximise la fonction de perte ou la métrique de performance :

$$\Theta^* = \arg\min_{\Theta \in S} \mathcal{L}(\Theta)$$

Notons le fait que le fléau de dimension du Grid Search peut être réduit par de la parallélisation car cette technique est particulièrement simple à implémenter dans le cadre du Grid Search. En conclusion, c'est une technique simple et exhaustive, mais son coût peut être prohibitif dans des espaces de recherche complexes. Son utilisation dépend donc du contexte du problème ainsi que des ressources disponibles. Pour des espaces de recherche plus grands, d'autres méthodes d'optimisation peuvent être préférées. Nous allons voir maintenant une deuxième méthode qui est une sorte d'optimisation du Grid Search, que l'on nomme Random Search [7].

2.2 Random Search

Le Random Search est une méthode alternative et simplifiée pour l'optimisation des hyperparamètres. Cette technique s'appuie sur la sélection aléatoire des configurations d'hyperparamètres, en fixant un budget de recherche fixe. Elle est particulièrement efficace lorsque certains hyperparamètres sont plus influents que d'autres. Random Search bénéficie d'une meilleure adaptabilité aux espaces de configuration de grande dimension. L'algorithme de Random Search peut être décrit comme suit :

- **Étape 0 :** Initialiser les paramètres de l'algorithme Θ_0, les points initiaux X_0 dans l'espace de recherche, et l'indice d'itération $k = 0$.
- **Étape 1 :** Générer une collection de points candidats V_{k+1} dans l'espace de recherche S selon une distribution d'échantillonnage spécifique.
- **Étape 2 :** Mettre à jour X_{k+1} en fonction des points candidats V_{k+1}, des itérations précédentes et des paramètres algorithmiques Θ_{k+1}.
- **Étape 3 :** Si un critère d'arrêt est rencontré, arrêter l'exécution. Sinon, incrémenter k et retourner à l'Étape 1.

Il existe plusieurs variantes de random search, par exemple *Single Point Generators* où l'ensemble des candidats V_{k+1} comprend qu'un seul élement, ou encore les *Multiple Point Generators* où V_{k+1} contient plusieurs élements.

On donne un exemple d'un algorithme des *Single Point Generators* : D'abord, on initialise une position aléatoire dans l'espace de recherche. Ensuite, on échantillonne de nouvelle position y d'une sphère de centre x, si le point y offre de meilleur resultat, on prend $x = y$ et on itère le processus jusqu'à ce qu'une condition d'arrêt soit atteinte.

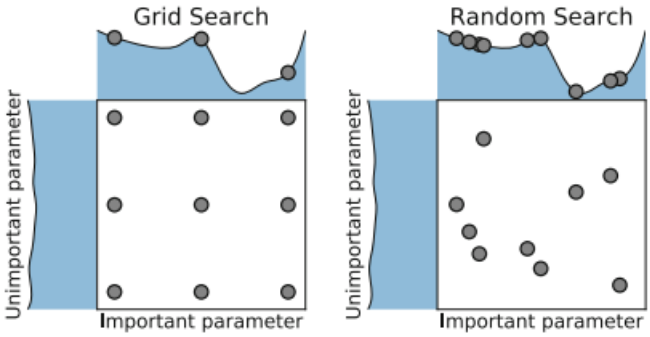


FIGURE 2 – Comparaison entre le grid search et le random search [9]

La figure 2 illustre la différence fondamentale entre Grid Search et Random Search dans l'optimisation d'hyperparamètres. Grid Search évalue chaque combinaison possible dans l'espace des hyperparamètres, ce qui peut être inefficace si le nombre de combinaisons est très élevé, surtout si certains hyperparamètres influencent peu les performances du modèle. En revanche, Random Search échantillonne aléatoirement des points dans l'espace des hyperparamètres, ce qui permet souvent de trouver de bonnes solutions plus rapidement et rend donc son utilisation en pratique plus fréquente.

3 Optimisation bayésienne

L'optimisation bayésienne est une méthode d'optimisation des hyperparamètres qui s'appuie sur un raisonnement bayésien. Contrairement aux méthodes présentées précé-

demment, les résultats antérieurs de test sont prises en compte. Ainsi, après avoir défini notre domaine de recherche pour les hyperparamètres (étape 1), un processus itératif est suivi de quatres étapes clés afin de trouver le prochain meilleur candidat [7].

On utilise une fonction **objectif** pour évaluer chaque ensemble d'hyperparamètres (étape 2). Cette fonction prend les valeurs des hyperparamètres et renvoie un score qui reflète la performance du modèle. Pour des raisons de simplicité, notre but sera de minimiser la fonction objectif par exemple dans le cas d'une minimisation du risque empirique.

Le **surrogate model**, est au cœur de l'optimisation bayésienne. Il est utilisé pour approximer la fonction objectif et comprendre la relation entre les hyperparamètres et la performance du modèle. Lors de l'étape 3, un tel modèle est construit. Nous allons explorer dans cette étude deux approches permettant d'obtenir un surrogate model, les **Processus Gaussiens (GP)** et le **Tree Parzen Estimator (TPE)**.

Dans les scénarios incertains, où l'optimisation bayésienne est employée, nous sommes confrontés au **compromis exploration-exploitation**. La fonction d'acquisition est une solution à ce problème (étape 4). L'optimisation bayésienne, comme évoqué précédemment, ne fait qu'approximer la fonction objectif. L'approximation étant imparfaite, nous ne voulons pas l'optimiser directement. Les modèles tels que les processus gaussiens sont conçus pour bien apprendre les incertitudes.Par exemple, l'**Upper Confidence Bound (UCB)** s'intéresse aux valeurs élevées en termes de moyenne μ et de l'incertitude σ, $\mu + \kappa\sigma$. On explore donc les domaines où l'on est le plus incertain jusqu'à ce que l'incertitude soit suffisamment réduite pour ne pas jouer un rôle important dans le résultat. Il en va de même pour d'autres fonctions d'acquisition telles que l'Expected Improvement ou la Probability of Improvement 3.2. Le Thompson Sampling résout ce problème en échantillonnant à partir de la distribution postérieure, donc en randomisant le processus en fonction des probabilités postérieures. La fonction d'acquisition vous oblige donc à explorer davantage plutôt qu'à vous concentrer sur l'exploitation pure et simple.

Enfin (Étape 5) , l'historique des observations est utilisé pour mettre à jour le surrogate model après chaque itération, permettant ainsi à l'algorithme de s'améliorer au fil du temps. Après le nombre d'itération initialisé, l'algorithme renvoie le meilleur set d'hyperparamètres trouvé.

L'optimisation bayésienne est particulièrement utile dans les scénarios où l'évaluation de la fonction objectif est coûteuse, car elle vise à minimiser le nombre d'évaluations nécessaires pour atteindre une performance optimale.

3.1 Surrogate model

Dans cette section, nous allons nous concentrer sur les surrogate models. cette modélisation revient souvent à trouver la loi prédictive $p(y|x)$. Deux méthodes prédominantes seront étudiées : les GPs et le TPE. Chacune de ces méthodes offre une perspective unique et des avantages spécifiques dans la prédiction de la performance des hyperparamètres.

3.1.1 Processus Gaussien (GP)

Le problème d'optimisation d'hyperparamètres est souvent un problème dit de **Black-box** où **Boîte Noire**, la forme explicite de la fonction objectif n'est pas donnée, ainsi la fonction à optimiser n'est accessible qu'à travers des évaluations directes, l'objectif de cette partie est de fournir des modèles approximant cette dite fonction.
Le problème à résoudre se reformule comme suit :

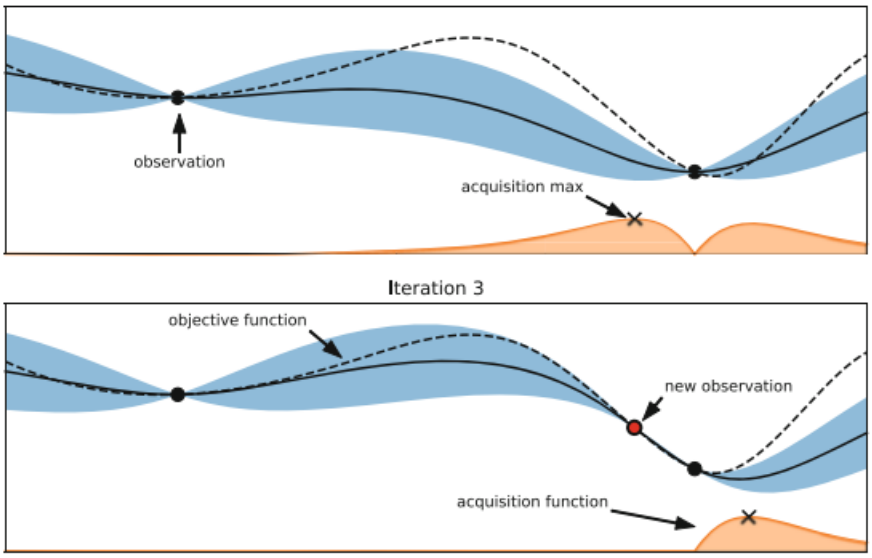


FIGURE 3 – illustration de l'optimisation bayesienne [9]

$$y = f(x) + \varepsilon \tag{1}$$

avec

— ε : un bruit gaussien de moyenne nulle et variance σ^2

— f la fonction objectif

— x un hyperparamètre donné

On rappelle qu'un processus gaussien est une collection de variables aléatoires tel que chaque vecteur fini construit à partir des élements de ce processus suit une loi normale multidimensionnelle. L'idée de la modélisation avec les GPs [3] est de raisonner directement sur la distribution de la fonction f (Nous verrons dans la section suivante qu'il est aussi possible de considérer des modèles "paramétriques" pour aboutir aux mêmes résutlats). Ainsi, nous considérons que $\{f(x), x \in \Theta\}$, avec Θ l'espace des hyperparamètres, est un processus gaussien. Ce dernier est complétement défini par sa moyenne et sa covariance. On note alors $f(x) \sim GP(m(x), k(x,x))$ avec :

— $m(x) = \mathbf{E}(f(x))$

— $k(x,x') = \mathbf{E}((f(x) - m(x))(f(x') - m(x'))$

Pour $X = [x_1 \dots x_n]$ des points données de Θ on a :

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{m}, \begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_1,x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n,x_1) & k(x_n,x_2) & \cdots & k(x_n,x_n) \end{bmatrix}\right) \tag{2}$$

avec : $\boldsymbol{m} = [m(x)\, m(x_1) \dots m(x_n)]^T$

Nous avons $y = f + \varepsilon$. Par définition d'un processus gaussien, $f \sim \mathcal{N}(0, K(X,X))$ où $K(X,X)$ est une matrice $n \times n$ formée en évaluant notre fonction de covariance (aussi appelée noyau) sur toutes les paires possibles d'entrées $x_i, x_j \in X$. ε est simplement un vecteur constitué d'échantillons iid de $\mathcal{N}(0, \sigma^2)$ et a donc la distribution $\mathcal{N}(0, \sigma^2 I)$. Ainsi, y est la somme de deux variables gaussiennes multivariées indépendantes, ayant pour

distribution $\mathcal{N}(0, K(X,X) + \sigma^2 I)$. On a également $\text{cov}(f_*, y) = K(x_*, X)$ où $K(x_*, X)$ est une matrice $1 \times n$ formée en évaluant le noyau sur toutes les paires d'entrées de test et d'entraînement et $f_* = f(x_*)$.

Nous pouvons ensuite utiliser les identités gaussiennes standards pour trouver la distribution conditionnelle à partir de la distribution conjointe,

$$f_*|y, X, x_* \sim \mathcal{N}(m_*, S_*)$$

où

$$m_* = K(x_*, X)[K(X,X) + \sigma^2 I]^{-1} y$$

et

$$S = K(x_*, x_*) - K(x_*, X)[K(X,X) + \sigma^2 I]^{-1} K(X, x_*)$$

On remarque que dans le cadre d'un modèle non bruité $\sigma^2 = 0$ et lorsque $X = x$ et $x_* = x$ la distribution a pour moyenne $m_* = y$ et $S = 0$, ainsi toutes les fonctions générées passent par les points déjà observés.

Fonction de covariance :

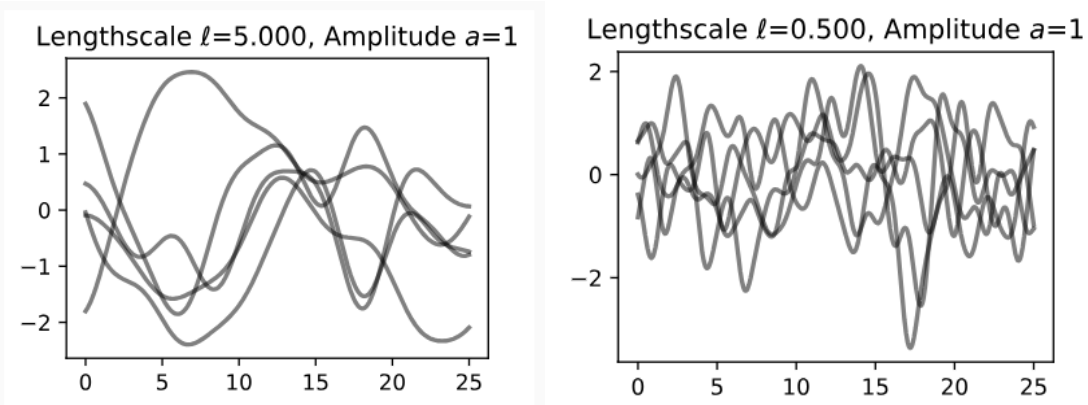
Un processus gaussien étant entièrement défini par sa moyenne et sa covariance, le choix de la fonction de covariance s'avère cruciale, on en cite quelques-uns :

— constant : $k(x, x') = C$

— linéaire : $k(x, x') = x^T x'$

— noyau RBF : $k(x, x') = a \exp\left(-\frac{1}{2\ell^2}\|x - x'\|^2\right)$

Le noyau RBF est la fonction de covariance la plus utilisée pour les GPs (et même pour les SVMs), elle introduit des paramètres a et ℓ qui contrôlent l'amplitude et le taux de variations des fonctions générées respectivement, la figure ci-dessus illustre l'effet du paramètre ℓ [8].



Il existe plusieurs méthodes pour estimer les paramètres a et ℓ, nous proposons dans cette étude de maximiser la fonction log-vraisemblance donnée par :

$$\log p(\mathbf{y}|\theta, X) = -\frac{1}{2}\mathbf{y}^\top [K_\theta(X,X) + \sigma^2 I]^{-1}\mathbf{y} - \frac{1}{2}\log|K_\theta(X,X)| - \frac{n}{2} log(2\pi) \qquad (3)$$

Il est possible de calculer le gradient à partir de la formule ci-dessus afin d'appliquer une descente de gradient, cependant, la fonction est souvent non convexe, ainsi, il faut itérer le processus plusieurs fois en changeant le point de départ à chaque itération,la méthode

peut être alors computationnellement exigeant vu qu'il faut inverser une matrice pour trouver la formule explicite.

Modèle paramétrique :

Une autre façon pour modéliser notre problème est de proposer des formules explicites de la fonction objectif [4], l'un des modèles les plus simples auquel on peut penser est la regression linéaire, ainsi $f(x) = w^T x$ avec $w \sim \mathcal{N}(0, \Sigma_p)$ pour rester dans le formalisme gaussien. Cependant, la relation entre l'entrée et la sortie n'est pas toujours linéaire, une façon pour améliorer le modèle est de projeter l'entrée sur un espace de plus grande dimension (voire un espace de Hilbert de dimension infinie). On considère alors les fonctions de la forme :

$$f(x) = w^T \phi(x)$$

avec ϕ une fonction qui sert à projeter les entrées sur des espaces plus "riches". Il est possible d'aboutir aux mêmes résultats de la partie précedente en suivant cette approche, on considère :

$$f(x) = \sum_{i=1}^{n} w_i \phi_i(x), \quad w_i \sim \mathcal{N}\left(0, \frac{\sigma_w^2}{n}\right), \quad \phi_i(x) = \exp\left(-\frac{(x - c_i)^2}{2\ell^2}\right).$$

Considérons maintenant ce qui se passe lorsque nous prenons un nombre de paramètres (et de fonctions ϕ_i) qui tend vers l'infini. Prenons $c_{i+1} - c_i = \Delta c = \frac{1}{n}$. Lorsque $n \to \infty$, la fonction de covariance devient en utilisant les sommes de Riemann :

$$k(x, x') = \frac{\sigma_w^2}{n} \sum_{i=1}^{n} \phi_i(x)\phi_i(x') \to \sigma_w^2 \int_{c_0}^{c_\infty} \phi_c(x)\phi_c(x')dc.$$

En prenant $c_0 = -\infty$, $c_\infty = \infty$ et $\Delta c \to 0$:

$$k(x, x') = \sigma_w^2 \int_{-\infty}^{\infty} \exp\left(-\frac{(x - c)^2}{2\ell^2}\right) \exp\left(-\frac{(x' - c)^2}{2\ell^2}\right) dc = \sqrt{\pi}\ell\sigma_w^2 \exp\left(-\frac{(x - x'))^2}{2(\sqrt{2}\ell)^2}\right) \propto k_{RBF}(x, x').$$

3.1.2 Tree Parzen Estimator (TPE)

La méthode de *Tree Parzen Estimator* est une variante de celle basée sur les processus gaussiens qui change la modélisation de la fonction objectif *Etape 3*. Elle s'appuie sur des modèles par morceaux pour représenter la distribution conditionnelle de la performance d'une configuration donnée, sachant les observations précédentes. Comme auparavant, le principe général consiste à réduire le nombre de fois où la fonction objectf doit être exécutée en choisissant uniquement l'ensemble d'hyperparamètres le plus prometteur à évaluer sur la base des appels précédents. L'ensemble suivant d'hyperparamètres est sélectionné sur la base d'un modèle de la fonction objectif, le *surrogate*. Le *TPE* a été présenté par **James Bergstra et al** dans leur papier [2] afin de proposer autre chose que les processus gaussiens qui dominaient la littérature jusqu'ici.

Pour présenter cette méthode, nous allons utiliser une nouvelle *Acquisition Function*, nommée l'*Expected Improvement (EI)*. Les détails autour de cette fonction sont présents dans la partie 3.2. L'idée cruciale de la méthode est d'utiliser la règle de Bayes. Alors que l'approche basée sur les processus gaussiens modélisait directement $p(y|x)$, cette stratégie modélise $p(x|y)$ et $p(y)$ [6] :

$$p(y|x) = \frac{p(x|y) \cdot p(y)}{p(x)} \tag{4}$$

$p(x|y)$ représente la probabilité des hyperparamètres compte tenu du score de la fonction objectif. Le *TPE* définit $p(x|y)$ en transformant le processus génératif que l'on retrouve sur la méthode cf partie 3.1 par des estimations de densités non paramétriques. On définit $p(x|y)$ en utilisant deux groupes de densités :

$$p(x \mid y) = \begin{cases} \ell(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{5}$$

où $l(x)$ est la densité formée en utilisant les observations $x(i)$ telles que la perte correspondante $f(x^{(i)})$ était inférieure à y^* et $g(x)$ est la densité formée en utilisant les observations restantes.

Alors que l'approche basée sur le processus gaussien favorise une valeur assez agressive pour y^* *(typiquement inférieure à la meilleure perte observée)*, l'algorithme TPE dépend d'un y^* qui est plus grand que la meilleur f(x) observé, de sorte que certains points peuvent être utilisés pour former l(x). L'algorithme TPE choisit y^* comme un quantile γ des valeurs y observées, de sorte que $p(y < y^*) = \gamma$ sans qu'aucun modèle spécifique pour p(y) ne soit nécessaire. Il faut maintenant éclaircir le calcul de l(x) et g(x). Tout d'abord, définissions le type de distributions qu'utilise *TPE* par type d'hyperparamètres :

- **Variables catégorielles** : Distribution catégorielle, ie, fonction de masse.
- **Variables réelles ou entières :**

  Sampler dans l'espace original : Mixture de gaussiennes tronquées.

  Sampler dans le log-espace : Mixture de gaussiennes tronquées dans le log-espace.

Nous tronquons les gaussiennes car la première étape de l'algorithme TPE est de préciser les intervalles de recherche pour chaque hyperparamètre. De fait, on force cette contrainte de par des gaussiennes tronquées pour mettre automatiquement une probabilité de 0 sur les valeurs d'hyperparamètres en dehors des ces intervalles. La mixture de gaussiennes part du principe que chaque point (ou valeur d'hyperparamètre) sera gaussien.

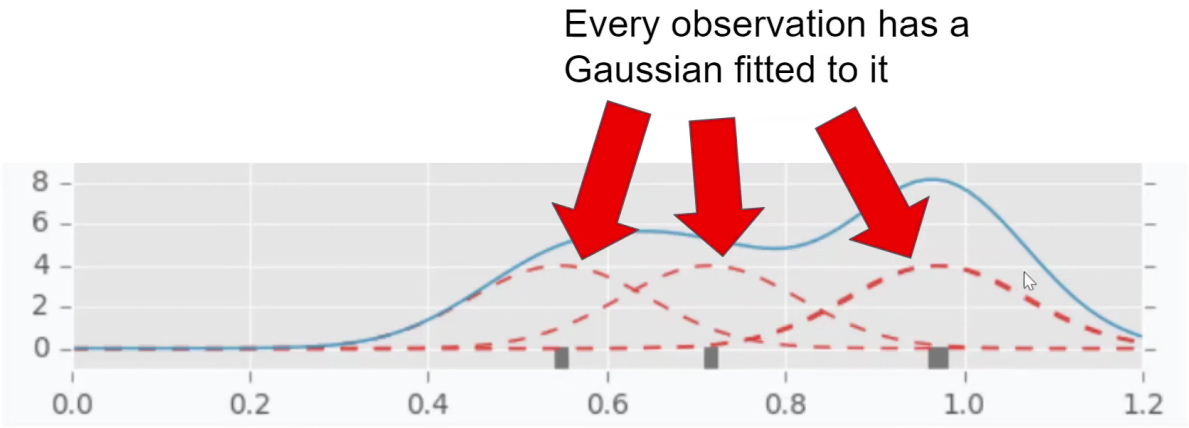


FIGURE 5 – Gaussian Mixture in *TPE* [1]

On voit sur la figure que chaque valeur d'hyperparamètre a une gaussienne associée dont la moyenne est la valeur de l'hyperparamètre et dont l'écart-type est la plus grande

distance (à droite ou à gauche) aux valeurs voisines. La densité finale est la moyenne de toutes les gaussiennes définies.

Pour finir, précisons le calcul de l'optimisation de l'*EI* dans l'algorithme TPE. La paramétrisation de $p(x, y)$ en $p(y)p(x \mid y)$ dans *TPE* a été choisie pour faciliter l'optimisation de l'*EI*.

$$\mathrm{EI}_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\, p(y \mid x) dy = \int_{-\infty}^{y^*} (y^* - y)\, \frac{p(x \mid y)p(y)}{p(x)} dy \tag{6}$$

Par construction, $\gamma = p\,(y < y^*)$ and $p(x) = \int_R p(x \mid y)p(y)dy = \gamma\ell(x) + (1-\gamma)g(x)$. Ainsi,

$$\int_{-\infty}^{y^*} (y^* - y)\, p(x \mid y)p(y)dy = \ell(x) \int_{-\infty}^{y^*} (y^* - y)\, p(y)dy = \gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y)dy, \tag{7}$$

Finalement,

$$EI_{y^*}(x) = \frac{\gamma y^* \ell(x) - \ell(x) \int_{-\infty}^{y^*} p(y)dy}{\gamma\ell(x) + (1-\gamma)g(x)} \propto \left(\gamma + \frac{g(x)}{\ell(x)}(1-\gamma)\right)^{-1}. \tag{8}$$

La dernière expression montre que pour maximiser l'*EI*, on cherchera des points x avec grande probabilité sous $\ell(x)$ et petite probabilité sous $g(x)$. L'algorithme retournera le candidat x^* avec la meilleure *EI*.

La revue de la littérature montre que l'optimisation bayésienne sous *PG* et *TPE* dépassent souvent les résultats d'optimisation grâce à *Random Search* [2]. Cependant, il y'a certainement des contextes dans lesquels ces algorithmes ne devraient pas donner de bons résultats. Il est possible qu'ils soient arbitrairement mauvais avec un mauvais choix de p(y|x). Il est également possible d'être plus lent que le *Random Search*pour trouver un optimum global avec un p(y|x) plutôt bon, s'il extrait une structure dans l'ensemble des hyperparamètres qui ne mène qu'à un optimum local. Finalement, les avantages de l'optimisation bayésienne sont leur *surrogate* qui dépend des données (et donc utilise nos résultats à chaque itération) et sa rapidité computationnelle par rapport à *Random Search*. Cependant, le défaut principal est la dépendance à l'entraînement du surrogate.

3.2 Exemples d'Acquisition Functions

3.2.1 Upper confidence bound (UCB)

On commence par donner l'un des exemples d' *Acquisition function* le plus simple : "upper confidence bound" (UCB)

$$a_\lambda(x) = \mu(x) + \lambda\sigma(x)$$

Ce dernier sépare explicitement les termes contrôlant l'exploration et l'exploitation, permettant une personnalisation précise de l'équilibre entre ces deux derniers. En intégrant la moyenne et la variance des prédictions, l'UCB encourage la sélection de points à la fois prometteurs et incertains, ainsi en ajustant la valeur de lambda, on peut manipuler la fonction d'acquisition pour favoriser l'exploration ou l'exploitation. Un lambda élevé privilégiera l'exploration (le terme de la variance devient plus important), le modèle alors considère des régions moins connues de l'espace des hyperparamètres. Inversement, un lambda plus faible favorise l'exploitation (le terme de la variance devient négligeable devant la moyenne), optimisant les performances dans les régions où le modèle donne de bons résultats.

3.2.2 Probability of Improvement (PI)

Nous allons analyser cette autre *Acquisition Function* dans le cas d'un processus gaussien. Supposons que nous voulions maximiser $f(x)$, et que la meilleure solution que nous ayons jusqu'à présent soit $x^\star$. Nous pouvons alors définir l'*Improvement*, $I(x)$, comme suit :

$$I(x) = \max\left(f(x) - f\left(x^\star\right), 0\right) \tag{9}$$

Par conséquent, si le nouveau x que nous examinons a une valeur associée $f(x)$ qui est inférieure à $f\left(x^\star\right)$, alors $f(x) - f\left(x^\star\right)$ est négatif. Nous ne progressons donc pas du tout, et la formule ci-dessus renvoie 0 , puisque le nombre maximum entre un nombre négatif et zéro, est zéro . Au contraire, si la nouvelle valeur $f(x)$ est supérieure à notre meilleure estimation actuelle, alors $f(x) - f\left(x^\star\right)$ est positif. Dans ce cas, $I(x)$ renvoie la différence, c'est-à-dire l'amélioration de notre meilleure solution actuelle si nous évaluons f au nouveau point $\boldsymbol{x}$.

Pour PI, nous attribuons à chaque candidat x la probabilité que $I(x) > 0$, c'est-à-dire que $f(x)$ soit plus grand que notre meilleur résultat actuel $f\left(x^\star\right)$. Rappelons que dans un processus gaussien, chaque point est associé à une distribution gaussienne. Par conséquent, au point x, la valeur de la fonction $f(x)$ est échantillonnée à partir d'une distribution normale de moyenne $\mu(x)$ et de variance $\sigma^2(x)$:

$$f(x) \sim \mathcal{N}\left(\mu(x), \sigma^2(x)\right)$$

Utilisons maintenant une reparamétrisation. Si $z \sim \mathcal{N}(0,1)$, alors $f(x) = \mu(x) + \sigma(x)z$ est une distribution normale avec une moyenne $\mu(x)$ et une variance $\sigma^2(x)$. Par conséquent, nous pouvons réécrire l'*Improvement*, $I(x)$, comme suit :

$$I(x) = \max\left(f(x) - f\left(x^\star\right), 0\right) = \max\left(\mu(x) + \sigma(x)z - f\left(x^\star\right), 0\right) z \sim \mathcal{N}(0,1) \tag{10}$$

Ce qui se passe ici est que, $\boldsymbol{x}$ est un point dont nous voulons vérifier s'il vaut la peine d'y évaluer f. Nous lui attribuons donc une valeur $I(x)$. Cependant, la valeur de $I(x)$ est échantillonnée à partir d'une distribution normale $\mathcal{N}\left(\mu(x), \sigma^2(x)\right)$. Voici donc comment nous calculons :

$$\mathrm{PI}(x) = \Pr(I(x) > 0) \Leftrightarrow \Pr\left(f(x) > f\left(x^\star\right)\right)$$

Si vous re-regardez la figure 6, il est clair que PI est la zone grisée sous la gaussinée pour $z > z_0$. De fait,

$$\mathrm{PI}(x) = 1 - \Phi\left(z_0\right) = \Phi\left(-z_0\right) = \Phi\left(\frac{\mu(x) - f\left(x^\star\right)}{\sigma(x)}\right) \tag{11}$$

où $\Phi(z) \equiv \mathrm{FDR}(z)$ et $z_0 = \frac{f(x^*) - \mu(x)}{\sigma(x)}$ où FDR est la fonction de répartition.

3.2.3 Expected Improvement (EI)

Nous allons également analyser cette fonction dans le cadre du processus gaussien. Son optimisation pour TPE a été détaillée en 3.1.2. L'*EI* tient compte de l'amélioration attendue en choisissant un nouveau set d'hyperparamètres. C'est là que la fonction est différente. Au lieu de se concentrer sur l'amélioration I(x), qui est une variable aléatoire, nous allons plutôt calculer l'"Amélioration Attendue", qui est la valeur attendue de I(x) :

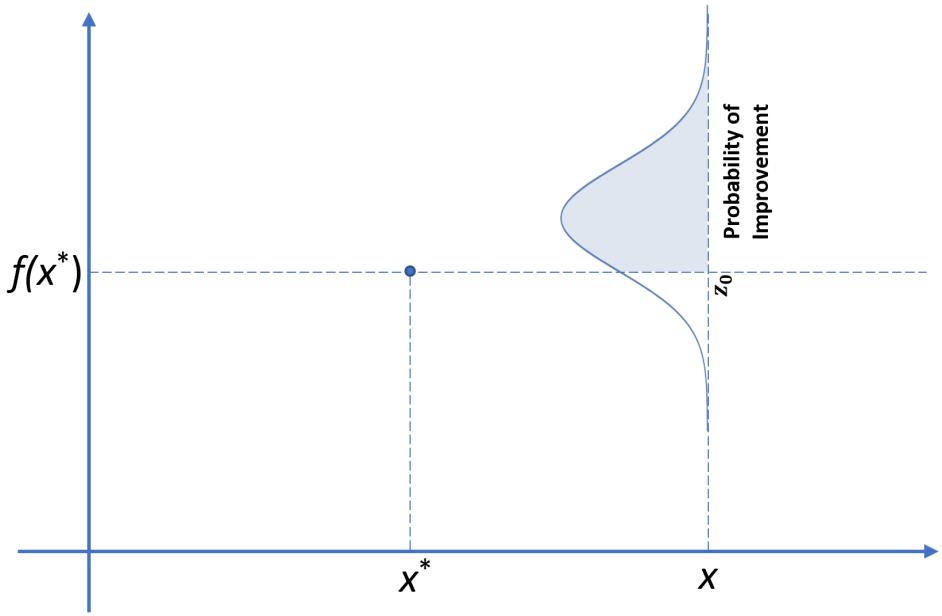


FIGURE 6 – PI Source : Ekamperi Github [5]

$$EI(x) \equiv E[I(x)] = \int_{-\infty}^{\infty} I(x)\phi(z)\,dz$$

Où $\phi(z)$ est la fonction de densité de probabilité de la distribution normale N(0,1), c'est-à-dire $\phi(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$.

$$EI(x) = \int_{-\infty}^{\infty} I(x)\phi(z)\,dz = \int_{-\infty}^{\infty} \max(f(x) - f(x^*), 0) \cdot \phi(z)\,dz$$

Pour calculer l'intégrale, nous allons la diviser en deux composantes, l'une où $f(x) - f(x^*)$ est positif et l'autre où il est négatif. Le point où le changement se produit est donné par :

$$f(x) = f(x^*) \Rightarrow \mu + \sigma z = f(x^*) \Rightarrow z = \frac{f(x^*) - \mu}{\sigma}$$

Appelons ce point $z_0 = \frac{f(x^*)-\mu}{\sigma}$, et divisons l'intégrale comme suit :

$$EI(x) = \int_{-\infty}^{z_0} I(x)\phi(z)\,dz + \int_{z_0}^{\infty} I(x)\phi(z)\,dz$$

On peut alors détailler ce calcul la :

$$
\begin{aligned}
\mathrm{EI}(x) &= \int_{z_0}^{\infty} \max\left(f(x) - f\left(x^\star\right), 0\right) \varphi(z)\mathrm{d}z = \int_{z_0}^{\infty} \left(\mu + \sigma z - f\left(x^\star\right)\right) \varphi(z)\mathrm{d}z \\
&= \int_{z_0}^{\infty} \left(\mu - f\left(x^\star\right)\right) \varphi(z)\mathrm{d}z + \int_{z_0}^{\infty} \sigma z \frac{1}{\sqrt{2\pi}} e^{-z^2/2}\, \mathrm{d}z \\
&= \left(\mu - f\left(x^\star\right)\right) \underbrace{\int_{z_0}^{\infty} \varphi(z)\mathrm{d}z}_{1-\Phi(z_0)\equiv 1-\mathrm{CDF}(z_0)} + \frac{\sigma}{\sqrt{2\pi}} \int_{z_0}^{\infty} z e^{-z^2/2}\, \mathrm{d}z \\
&= \left(\mu - f\left(x^\star\right)\right)\left(1 - \Phi\left(z_0\right)\right) - \frac{\sigma}{\sqrt{2\pi}} \int_{z_0}^{\infty} \left(e^{-z^2/2}\right)' \mathrm{d}z \\
&= \left(\mu - f\left(x^\star\right)\right)\left(1 - \Phi\left(z_0\right)\right) - \frac{\sigma}{\sqrt{2\pi}} \left[e^{-z^2/2}\right]_{z_0}^{\infty} \\
&= \left(\mu - f\left(x^\star\right)\right) \underbrace{\left(1 - \Phi\left(z_0\right)\right)}_{\Phi(-z_0)} + \sigma\varphi\left(z_0\right) \\
&= \left(\mu - f\left(x^\star\right)\right) \Phi\left(\frac{\mu - f\left(x^\star\right)}{\sigma}\right) + \sigma\varphi\left(\frac{\mu - f\left(x^\star\right)}{\sigma}\right)
\end{aligned}
\tag{12}
$$

Quand est-ce qu'EI(x) prend des valeurs élevées ? Quand $\mu > f(x^*)$. C'est-à-dire, la valeur moyenne du processus gaussien est élevée à x. L'amélioration attendue est également accrue lorsqu'il y a beaucoup d'incertitude, donc lorsque $\sigma > 1$. D'ailleurs, la formule ci-dessus fonctionne pour $\sigma(x) > 0$, sinon, si $\sigma(x) = 0$, on a EI(x) = 0.

En injectant un (hyper)paramètre ξ dans la formule pour EI(x), on peutajuster finement la quantité d'exploitation par rapport à la quantité d'exploration que l'algorithme effectuera. Ainsi, la formule complète est :

$$
EI(x;\xi) = (\mu - f(x^*) - \xi)\Phi\left(\frac{\mu - f(x^*) - \xi}{\sigma}\right) + \ \sigma\phi\left(\frac{\mu - f(x^*) - \xi}{\sigma}\right)
$$

Pour $\xi = 0$, nous obtenons simplement la formule précédente. Cependant, pour de grandes valeurs de ξ, vous pouvez le considérer comme si nous prétendions avoir une valeur actuelle meilleure plus grande que nous ne le faisons réellement. Par conséquent, cela oriente l'algorithme vers une exploration plus poussée.

4 Conclusion et Prochaines Etapes

En conclusion, les méthodes d'optimisation basées sur un modèle bayésien construisent un modèle de probabilité de la fonction objectif afin de proposer des choix plus intelligents pour le prochain ensemble d'hyperparamètres à évaluer. Les étapes de l'algorithme que nous avons détaillé durant ce papier sont une formalisation de l'optimisation bayésienne qui est plus efficace pour trouver les meilleurs hyperparamètres pour un modèle d'apprentissage automatique que le GridSearch ou le RandomSearch.

Les méthodes d'optimisation bayésienne diffèrent dans la façon dont elles construisent le *surrogate* et l'*Acquisition Function* qu'elles utilisent, mais elles s'appuient toutes sur les informations des essais précédents pour proposer de meilleurs hyperparamètres pour l'évaluation suivante.

À un niveau élevé, les méthodes d'optimisation bayésienne sont efficaces parce qu'elles choisissent les hyperparamètres suivants en connaissance de cause, tout comme le ferait un humain : On passe un peu plus de temps à sélectionner les hyperparamètres suivants afin de faire moins d'appels à la fonction objectif. Dans la pratique, le temps passé à sélectionner les hyperparamètres suivants est sans importance par rapport au temps passé dans la fonction objectif, d'où leur pertinence.

Heureusement pour nous, il existe aujourd'hui un certain nombre de bibliothèques qui permettent de faire de l'optimisation bayésienne en Python. **Spearmint** et **MOE** utilisent un processus gaussien pour le surrogate, **Hyperopt** utilise le *TPE* et **SMAC** utilise une régression Random Forest, méthode que nous n'avons pas évoqué dans ce papier.. Ces bibliothèques utilisent toutes le l'*EI* pour sélectionner les hyperparamètres suivants du modèle de substitution. La librairie **Optuna** est également en grande expansion car elle permet de choisir quel type d'optimisation on désire de faire et propose une simplicité d'utilisation impressionnante.

Les prochaines étapes de notre projet seront de justement appliquer une de ces librairies dans un problème concret de Machine Learning pour comparer **GridSearch** vs **RandomSearch** vs **OptimBayes**. Nous utiliserons un challenge Kaggle pour ce faire afin de pouvoir accéder à des problèmes connus et des datasets utilisés et ainsi tester en pratique la théorie que nous venons de détailler.

Bibliographie

[1] AIXplained. *Automated Machine Learning - Tree Parzen Estimator (TPE)*. URL : https://www.youtube.com/watch?v=bcy6A57jAwI.

[2] James Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In : (2011).

[3] Christopher K. I. Williams Carl Edward Rasmussen. "Gaussian process for machine learning". In : (2005).

[4] Zhiting Hu Chao-Ming Yen Biwei Huang. *Gaussian Process and Deep Kernel Learning*. URL : https://www.cs.cmu.edu/~epxing/Class/10708-17/notes-17/10708-scribe-lecture24.pdf.

[5] Stathis Kamperis. "Acquisition functions in Bayesian Optimization". In : (). URL : https://ekamperi.github.io/machine%20learning/2021/06/11/acquisition-functions.html.

[6] Will Koehrsen. *A Conceptual Explanation of Bayesian Hyperparameter Optimization*. URL : https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f.

[7] F Hutter M Feurer. "Hyperparameter optimization". In : (2019).

[8] Aston Zhang Zack C. Lipton Mu Li Alex J. Smola. *Dive into Deep Learning - introduction to gaussian process*. URL : https://d2l.ai/chapter_gaussian-processes/gp-intro.html.

[9] Frank Hutter Lars Kotthoff Joaquin Vanschoren. "Automated machine learning". In : (2019).