Optimización Bayesiana para busqueda de hiperparámetros

Índice

1 Optimización Bayesiana para búsqueda de hiperparámetros **2**
- 1.1 Introducción 2
- 1.2 Búsqueda de Hiperparámetros 2
- 1.3 Optimización de Hiperparámetros 4
  - 1.3.1 Enfoques Automáticos 4
  - 1.3.2 Ventajas de la Optimización Bayesiana 4
- 1.4 Fundamentos Matemáticos de la Optimización Bayesiana 5
  - 1.4.1 Proceso Gaussiano (GP) 5
  - 1.4.2 Función de Adquisición 6
- 1.5 Implementación con NumPy y SciPy 8
- 1.6 Librerias para la Optimización Bayesiana 11
  - 1.6.1 Scikit-Optimize 11
  - 1.6.2 GPyOpt 12
- 1.7 Aplicación de Optimización Bayesiana en Aprendizaje Automático 14
  - 1.7.1 Diabetes 14
  - 1.7.2 California Housing Prices 18

2 Fuentes **20**

1 Optimización Bayesiana para búsqueda de hiperparámetros

1.1 Introducción

En el ámbito de la ciencia de datos y el aprendizaje automático, la selección de hiperparámetros es una tarea esencial para garantizar el desempeño óptimo de los modelos predictivos. Los hiperparámetros, a diferencia de los parámetros ajustados durante el entrenamiento, son configuraciones definidas previamente que afectan directamente la capacidad de generalización de los modelos, es decir, su habilidad para aprender patrones complejos y evitar problemas como el sobreajuste o el subajuste.

Técnicas como la búsqueda en cuadrícula (`GridSearch`) y la búsqueda aleatoria (`RandomSearch`) han sido ampliamente utilizadas para esta tarea. Sin embargo, ambas presentan limitaciones significativas: son computacionalmente costosas en espacios de búsqueda grandes y carecen de estrategias para priorizar configuraciones prometedoras.

La optimización bayesiana ha emergido como una solución eficiente y efectiva para abordar este tipo de desafíos. Esta técnica, combina principios probabilísticos y de optimización para modelar iterativamente el espacio de búsqueda. Al construir un modelo probabilístico del rendimiento del modelo en función de los hiperparámetros, permite identificar de manera estratégica configuraciones que optimizan métricas como la pérdida logarítmica o la precisión.

En este proyecto, exploramos la aplicación de la optimización bayesiana para la búsqueda de hiperparámetros en modelos de aprendizaje automático y profundo. A través de herramientas como procesos gaussianos (Gaussian Processes), estimadores Parzen estructurados en árbol (Tree-structured Parzen Estimators, TPE), y otras metodologías avanzadas, para evaluar su eficacia frente a enfoques tradicionales. Además, analizaremos sus beneficios, desafíos y limitaciones, proporcionando un marco claro para su implementación en proyectos prácticos.

1.2 Búsqueda de Hiperparámetros

La importancia de un ajuste adecuado de hiperparámetros, ha sido ampliamente reconocida en la literatura científica, ya que puede mejorar significativamente el rendimiento y la generalización de los modelos. Existen diversos enfoques descritos, desde métodos manuales hasta algoritmos automáticos basados en optimización global.

Limitaciones de Enfoques Tradicionales

1. **Búsqueda en cuadrícula (*Grid Search*):**

ste enfoque realiza una búsqueda exhaustiva, evaluando todas las combinaciones posibles dentro de un rango predefinido de hiperparámetros. Aunque garantiza la exploración completa del espacio, es ineficiente en términos computacionales, especialmente en espacios de alta dimensión.

2. **Búsqueda aleatoria (*Random Search*):**

Aquí, los hiperparámetros se seleccionan aleatoriamente dentro de los límites definidos. Aunque mejora la eficiencia respecto al Grid Search, carece de una estrategia para priorizar configuraciones prometedoras, lo que resulta en un rendimiento variable en problemas complejos.

3. **Optimización Bayesiana:**

La optimización bayesiana, aborda estas limitaciones al construir un modelo probabilístico del rendimiento del modelo, en función de los hiperparámetros. Este enfoque, guía la búsqueda hacia configuraciones óptimas mediante un equilibrio entre **exploración** (*probar regiones poco exploradas*) y **explotación** (*ajustar configuraciones prometedoras*).

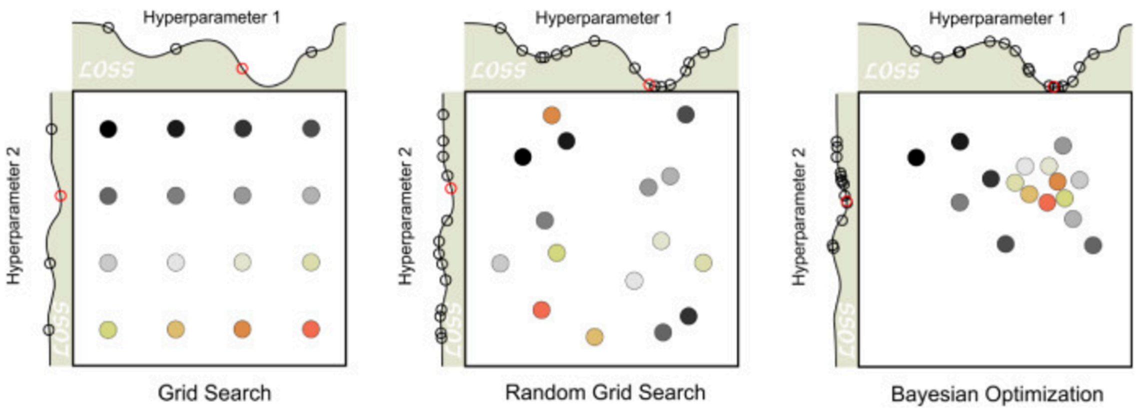


Figure 1: Comparativo de enfoques para búsqueda de hiperpárametros

Estudios destacados incluyen:

- En [1], se demostró que la optimización bayesiana, aplicada a modelos como bosques aleatorios y redes neuronales profundas (CNN, RNN), supera en eficiencia y precisión a los enfoques tradicionales.
- En [2], se destacó la capacidad del estimador TPE para ajustar hasta 32 hiperparámetros en modelos complejos como redes de creencias profundas (DBN).
- En [3], se validó la superioridad de la optimización bayesiana en términos de velocidad y rendimiento, especialmente en espacios de búsqueda de alta dimensión o recursos computacionales limitados.

1.3 Optimización de Hiperparámetros

El ajuste de hiperparámetros puede entenderse como un problema de optimización de funciones de caja negra (*BlackBox*), donde la función objetivo (por ejemplo, la métrica de evaluación del modelo) no tiene una expresión cerrada, ni derivadas accesibles. Esto dificulta el uso de métodos tradicionales como el descenso de gradiente.

1.3.1 Enfoques Automáticos

1. **Grid Search**:

Proporciona un marco exhaustivo, pero enfrenta la maldición de la dimensionalidad, limitando su aplicabilidad en problemas complejos.

2. **Random Search**:

Aunque más eficiente, carece de una estrategia para explotar información previa o configuraciones prometedoras.

1.3.2 Ventajas de la Optimización Bayesiana

Integra información previa sobre el espacio de búsqueda, para actualizar la distribución posterior iterativamente. Utiliza funciones de adquisición, para decidir las siguientes configuraciones a probar, optimizando recursos computacionales.Es decir, modela el rendimiento del modelo de manera probabilística, permitiendo una exploración más eficiente y efectiva del espacio de hiperparámetros.

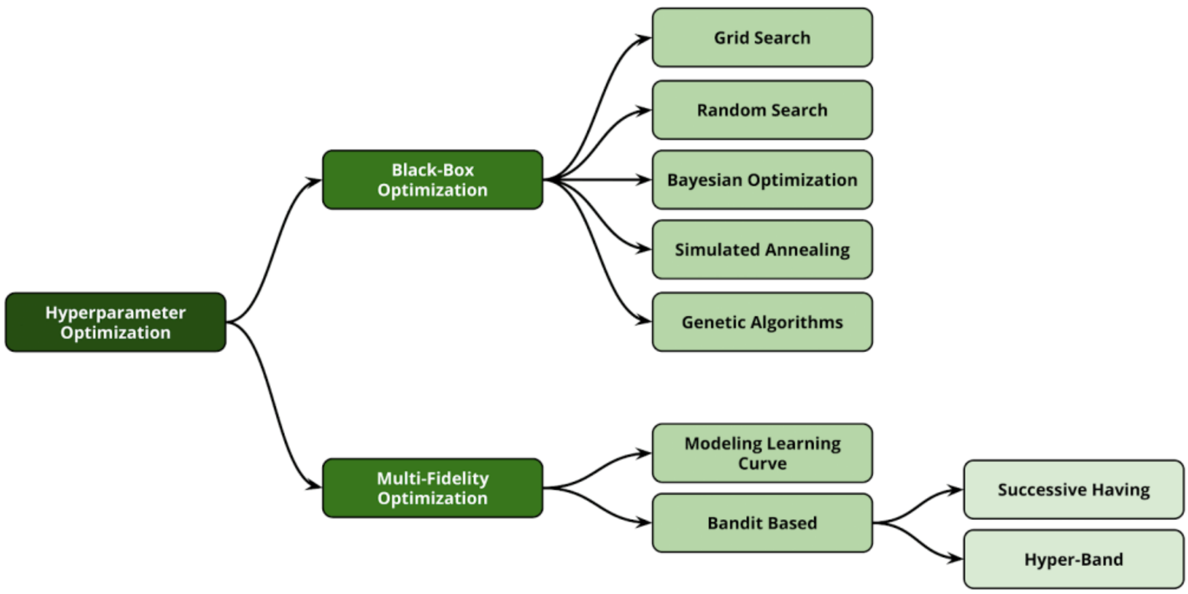


Figure 2: Esquema de enfoques en la búsqueda de hiperpárametros

1.4 Fundamentos Matemáticos de la Optimización Bayesiana

La optimización bayesiana busca maximizar una función objetivo $f(x)$ desconocida en un espacio de búsqueda A, utilizando el siguiente enfoque:

$$x^{+} = \underset{x \in A}{\text{arg max}}\, f(x) : \tag{1}$$

Se basa en el teorema de Bayes, el cual establece que, dado un conjunto de datos evidenciales E, la probabilidad posterior $P(M|E)$ de un modelo M, es proporcional a la probabilidad $P(E|M)$, de observar E, dado el modelo M, multiplicada por la probabilidad previa $P(M)$:

$$P(M|E) \propto P(E|M)P(M) : \tag{2}$$

La fórmula anterior, refleja la idea central de la optimización bayesiana. El principio consiste en combinar la distribución previa de la función $f(x)$ con la información de las muestras (evidencia) para obtener la distribución posterior de la función. Luego, se utiliza esta información posterior para determinar dónde se maximiza la función $f(x)$ según un criterio. En otras palabras,combina una distribución previa, con evidencia obtenida durante la búsqueda, para actualizar la distribución posterior de $f(x)$.

El criterio se representa mediante una función de utilidad u, también llamada **función de adquisición**. La función u se utiliza para determinar el siguiente punto de muestreo, con el fin de maximizar la utilidad esperada. Al explorar el área de muestreo, es necesario considerar tanto la exploración (muestreo en áreas de alta incertidumbre), como la explotación (muestreo en áreas con valores altos). Este equilibrio ayuda a reducir el número de muestras necesarias y mejora el rendimiento, incluso cuando la función tiene múltiples máximos locales.

1.4.1 Proceso Gaussiano (GP)

Es una técnica, basada en la teoría de probabilidad gaussiana y aprendizaje bayesiano. A diferencia de una distribución gaussiana, que describe variables escalares o vectores, un proceso gaussiano describe **propiedades de funciones**. Específicamente, un proceso gaussiano, asume que cualquier subconjunto finito de valores aleatorios sigue una distribución gaussiana multivariada.

Un GP se define por su función media $m(x)$ y su función de covarianza $k(x, x')$, representado como:

$$f(x) \sim GP(m(x), k(x, x'))$$

- **Función de Covarianza**

Una función común para $k(x_i, x_j)$, es la exponencial cuadrada:

$$k(x_i, x_j) = \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right)$$

donde x_i y x_j son puntos de muestreo. Si x_i y x_j están cerca, $k(x_i, x_j)$ se aproxima a 1; de lo contrario, tiende a 0. Esto representa la correlación y la influencia mutua entre los puntos.

- **Predicción y Varianza:**

Se calcula la media $\mu_{t+1}(x_{t+1})$ y varianza $\sigma^2_{t+1}(x_{t+1})$ para determinar el siguiente punto a evaluar.

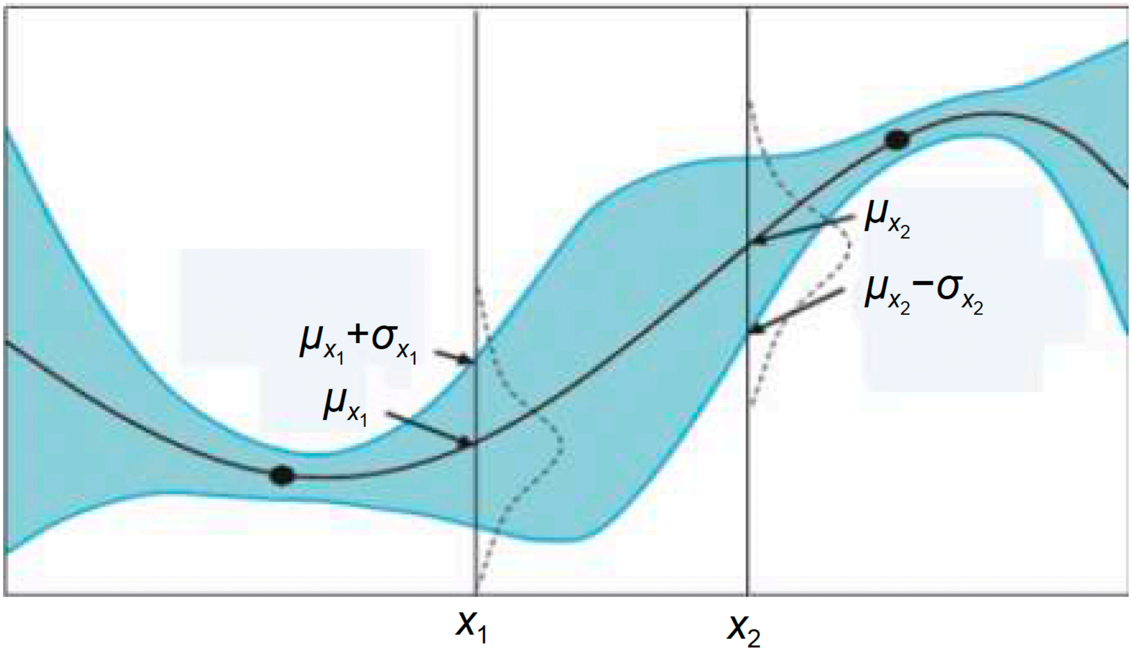


Figure 3: Fuente (3) Proceso Gaussiano

1.4.2 Función de Adquisición

La función de adquisición o función de utilidad $u(x)$, determina el próximo punto a muestrear, equilibrando exploración y explotación con el fin de maximizar la utilidad esperada. Al explorar el área de muestreo, es necesario considerar tanto la exploración (muestreo en áreas de alta incertidumbre), como la explotación (muestreo en áreas con valores altos). Este equilibrio ayuda a reducir el número de muestras necesarias y mejora el rendimiento, incluso cuando la función tiene múltiples máximos locales. Ejemplos:

1. **Probabilidad de Mejora (PI)**

La función PI, busca puntos donde el valor de $f(x)$ sea mayor al valor óptimo actual $f(x^+)$. La probabilidad de mejora se expresa como:

$$PI(x) = P(f(x) \geq f(x^+)) = \Phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right),$$

donde $\Phi(\cdot)$ es la función de distribución acumulativa de la normal estándar. Una versión extendida de PI introduce un parámetro ϵ para garantizar que el nuevo punto muestreado supere al óptimo actual por al menos ϵ:

$$PI(x) = \Phi\left(\frac{\mu(x) - f(x^+) - \epsilon}{\sigma(x)}\right).$$

2. **Mejora Esperada (EI)**

La función EI, calcula la expectativa del grado de mejora que puede lograrse al explorar un punto cerca del óptimo actual. La mejora esperada se define como:

$$I(x) = \text{max}\{0, f_{t+1}(x) - f(x^+)\}.$$

Maximizamos la EI respecto al valor óptimo actual $f(x^+)$:

$$x = \text{arg}\max_x E[I(x)].$$

La expresión para EI es:

$$E(I) = \sigma(x)\left[Z\Phi(Z) + \phi(Z)\right],$$

donde

$Z = \frac{\mu(x) - f(x^+)}{\sigma(x)}$, $\Phi(\cdot)$ es la función acumulativa y

$\phi(\cdot)$ es la densidad de probabilidad de la normal estándar.

3. **Límite Superior de Confianza (GP-UCB)**

La función GP-UCB, decide si el próximo punto a muestrear debe explotar el valor óptimo actual (zona de alta $\mu(x)$) o explorar zonas de alta incertidumbre $(\sigma(x))$. El equilibrio se controla mediante el parámetro κ:

$$UCB(x) = \mu(x) + \kappa\sigma(x).$$

En la siguiente imagen se compara el rendimiento de las funciones PI, EI y GP-UCB en el proceso de optimización. Las líneas azules representan la media posterior, mientras que las áreas sombreadas muestran la incertidumbre ($\pm\sigma(x)$). EI y GP-UCB logran encontrar el valor global óptimo, mientras que PI tiende a quedarse atrapada en óptimos locales debido a su naturaleza codiciosa.

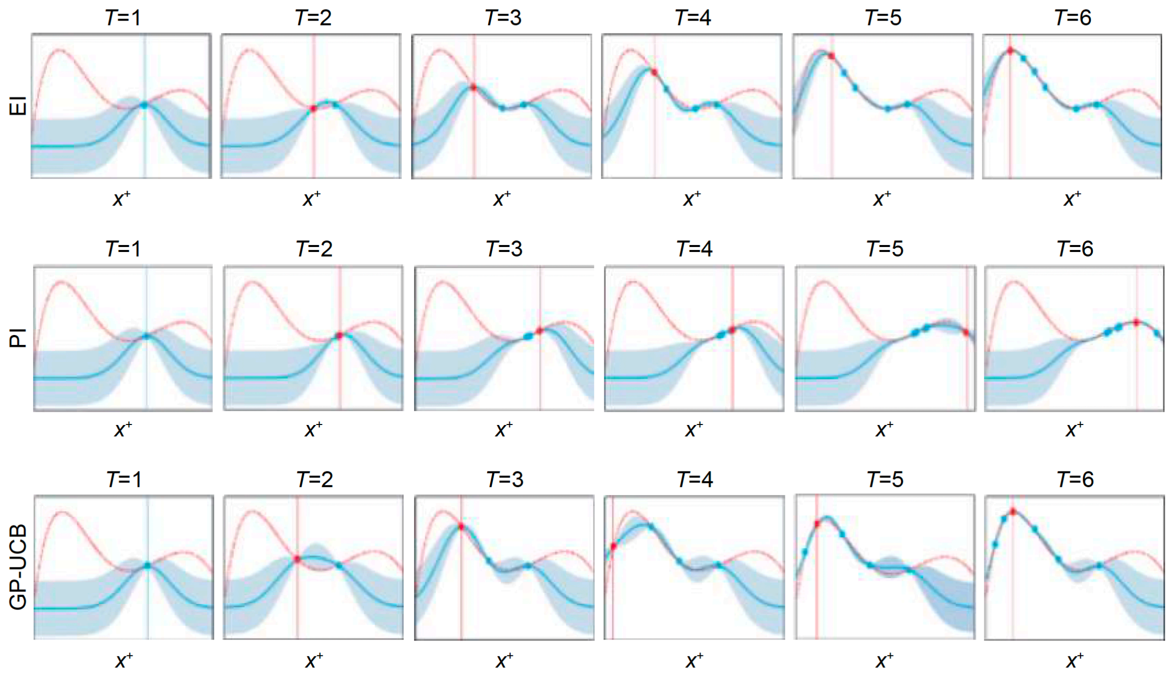


Figure 4: Fuente (3) Comparación de funciones de adquisición)

Según los resultados experimentales, la función EI es ideal para optimizar hiperparámetros por su simplicidad y rendimiento superior en comparación con PI y GP-UCB. Sin embargo, la elección de la función de adquisición depende del problema y la función objetivo, por lo que es recomendable probar varias funciones para determinar la más adecuada.

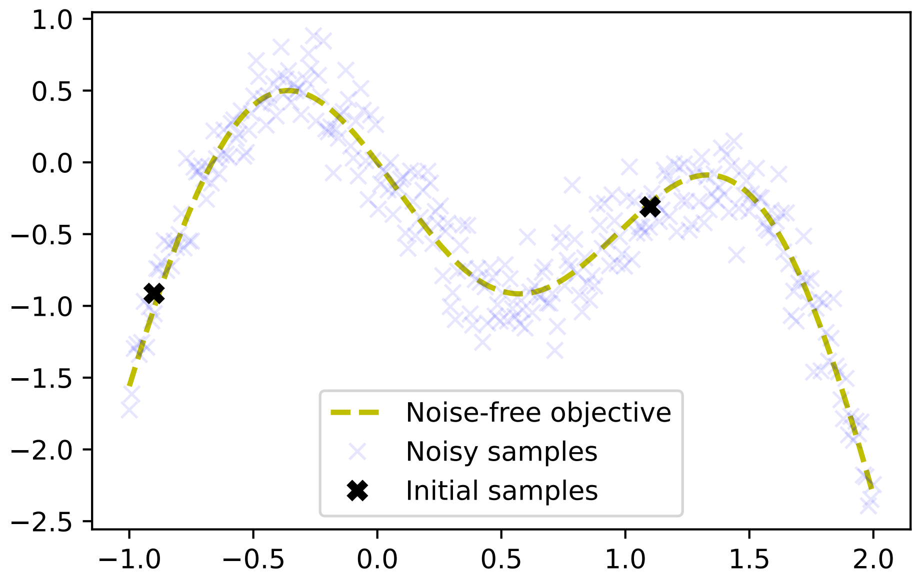
1.5 Implementación con NumPy y SciPy

En esta sección, implementaremos la función de adquisición y su optimización en NumPy y SciPy, así mismo usaremos scikit-learn para la el proceso gaussiano. Aunque tenemos una expresión analítica del objetivo de optimización `f` en el siguiente ejemplo, lo tratamos como una `Black-Box` y lo aproximamos iterativamente con un proceso gaussiano durante la optimización bayesiana. Además, las muestras extraídas de la función objetivo son ruidosas y el nivel de ruido está dado por la variable `noise`. La optimización se realiza dentro de los `límites` dados. También asumimos que existen dos muestras iniciales en `X_init` e `Y_init`.

Función:

$$f(X) = -\sin(3X) - X^2 + 0.7X + \text{noise} * \text{randn}$$

La siguiente gráfica muestra: - La función objetivo libre de ruido - La cantidad de ruido al representar gráficamente una gran cantidad de muestras y - Las dos muestras iniciales



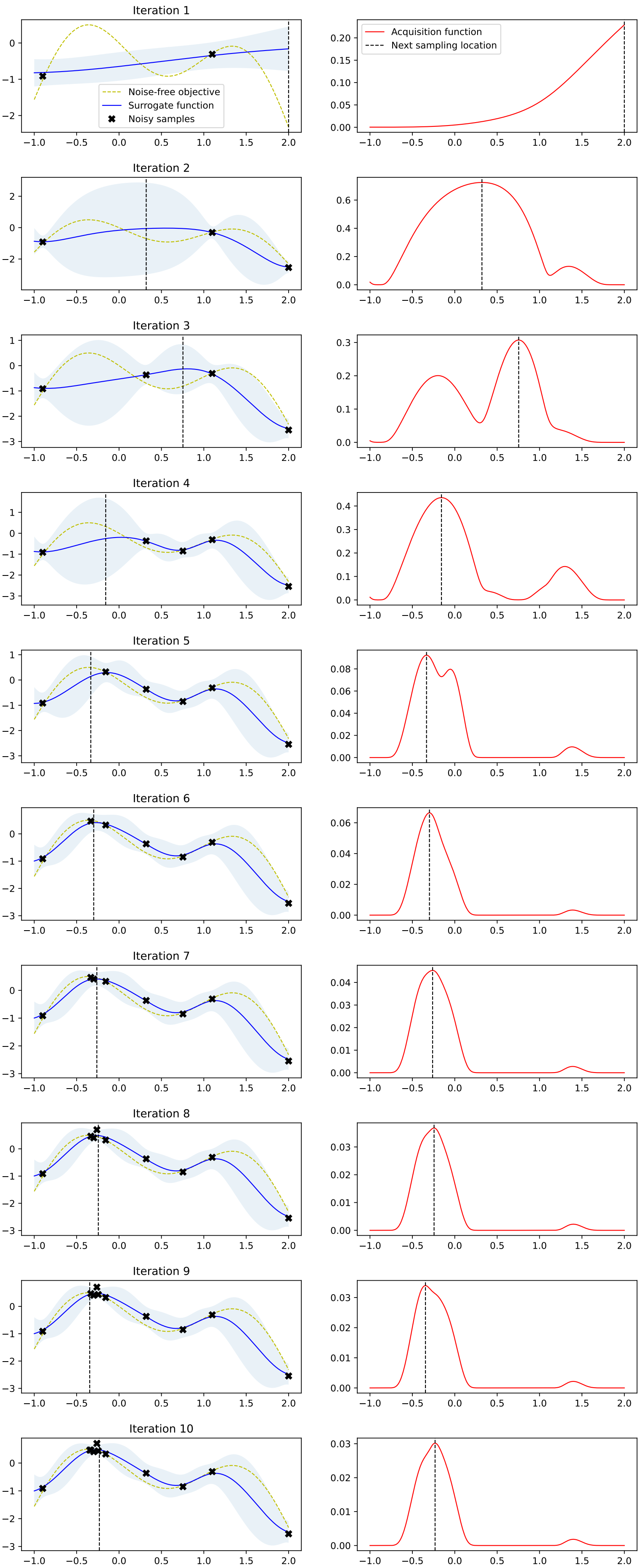
El objetivo, **es encontrar el óptimo global a la izquierda**, en una pequeña cantidad de pasos. El siguiente paso es implementar la función de adquisición definida como función EI (`expected_improvement`).

También necesitamos una función que proponga el siguiente punto de muestreo, calculando la ubicación del máximo de la función de adquisición. La optimización se reinicia `n_restarts` veces para evitar óptimos locales.

Ahora tenemos todos los componentes necesarios para ejecutar la optimización bayesiana con el algoritmo descrito anteriormente. El proceso gaussiano del siguiente ejemplo está configurado con un Matérn kernel. El nivel de ruido conocido se configura con el parámetro `alpha`.

La optimización bayesiana, se ejecuta durante 10 iteraciones. En cada iteración, produce una fila con dos gráficos.

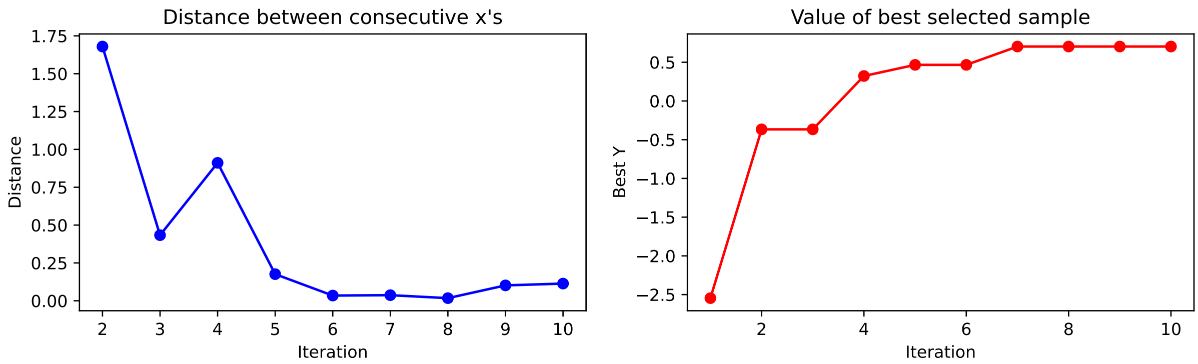
- El gráfico de la izquierda muestra la función objetivo sin ruido, **la función sustituta**, que es la media predictiva posterior de Gaussian Process, el intervalo de confianza es de 95% de la media y las muestras ruidosas obtenidas de la función objetivo hasta el momento.
- El gráfico de la derecha muestra la función de adquisición.
- La línea discontinua vertical en ambos gráficos muestra el punto de muestreo propuesto para la siguiente iteración, que corresponde al máximo de la función de adquisición.


Iteration 1
Noise-free objective
Surrogate function
Noisy samples
Acquisition function
Next sampling location
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10

Se puede observar, cómo las dos muestras iniciales dirigen la búsqueda hacia la dirección del máximo local en el lado derecho, pero la exploración, permite que el algoritmo escape de ese óptimo local y encuentre el óptimo global en el lado izquierdo.

Así mismo se observa también cómo las propuestas de puntos de muestreo, a menudo caen dentro de regiones de alta incertidumbre (exploración) y no solo están impulsadas por los valores más altos de la función sustituta (explotación).

Un gráfico de convergencia revela cuántas iteraciones se necesitan para encontrar un máximo y si las propuestas de puntos de muestreo se mantienen alrededor de ese máximo, es decir, convergen a pequeñas diferencias entre pasos consecutivos.

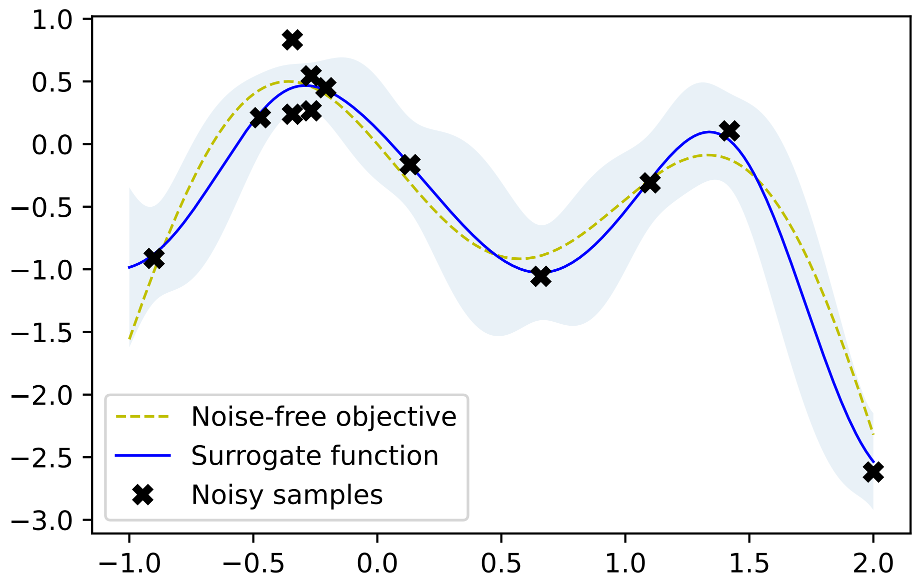


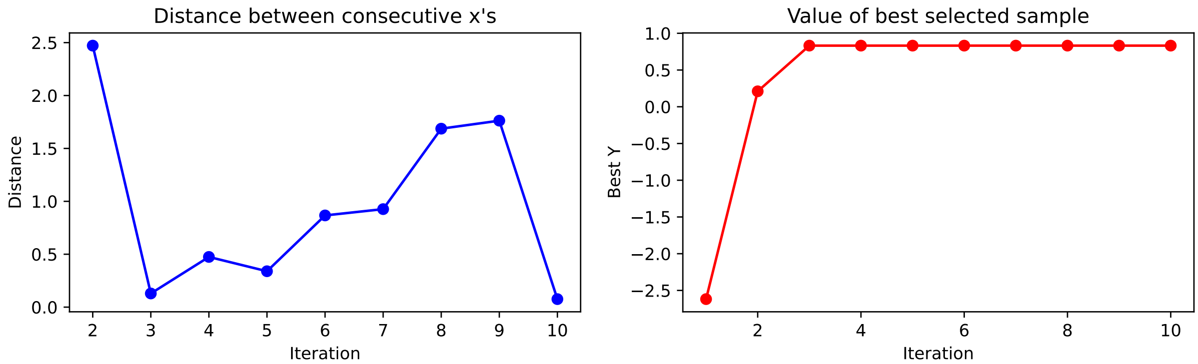
1.6 Librerias para la Optimización Bayesiana

Existen numerosas librerias de optimización bayesiana y el objetivo de este análisis no es brindar una descripción general de estas, sino, de ejemplificar dos y mostrar la configuración mínima necesaria para ejecutar el ejemplo anterior.

1.6.1 Scikit-Optimize

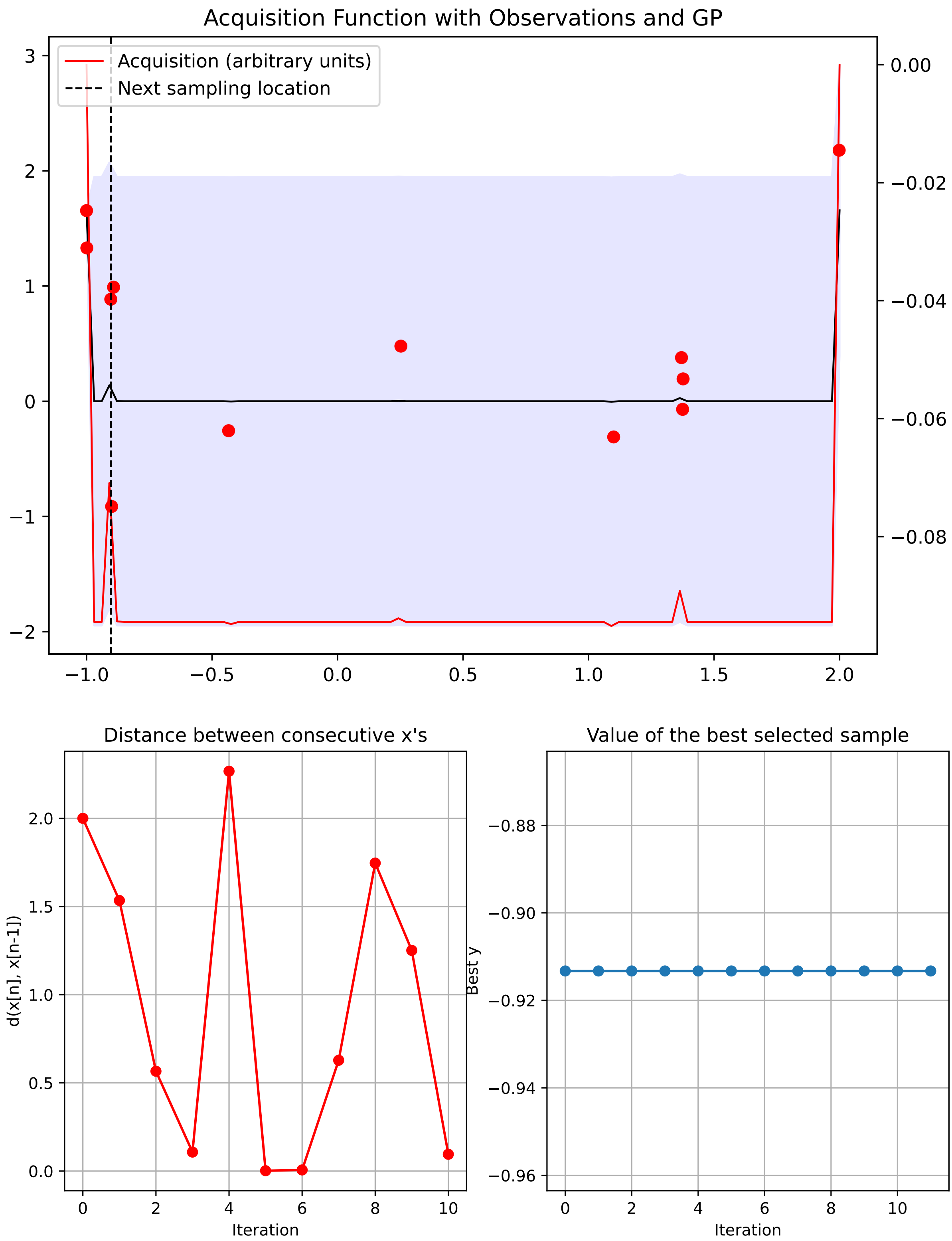
Scikit-optimize es una biblioteca para la optimización basada en modelos secuenciales que se basa en NumPy, SciPy y Scikit-Learn. También admite la optimización bayesiana mediante procesos gaussianos. La API está diseñada entorno a la minimización, por lo tanto, tenemos que proporcionar valores de función objetivo negativos. Los resultados obtenidos aquí difieren ligeramente de los resultados anteriores debido al comportamiento de optimización no determinista y a las diferentes muestras ruidosas extraídas de la función objetivo.





1.6.2 GPyOpt

GPyOpt es una biblioteca de optimización bayesiana basada en GPy. El nivel de abstracción de la API, es comparable al de scikit-optimize. La API `BayesianOptimization` proporciona un parámetro `maximize` para configurar si la función objetivo se maximizará o minimizará (predeterminado). En la versión 1.2.1, esto parece ignorarse al proporcionar muestras iniciales, por lo que tenemos que negar sus valores objetivo manualmente en el siguiente ejemplo. Además, los métodos integrados `plot_acquisition` y `plot_convergence` muestran el resultado de la minimización en cualquier caso. Nuevamente, los resultados obtenidos aquí difieren ligeramente de los resultados anteriores debido al comportamiento de optimización no determinista y a las diferentes muestras ruidosas extraídas de la función objetivo.


Acquisition Function with Observations and GP
Acquisition (arbitrary units)
Next sampling location
3
2
1
0
−1
−2
0.00
−0.02
−0.04
−0.06
−0.08
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
Distance between consecutive x's
d(x[n], x[n-1])
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
Iteration
Value of the best selected sample
Best y
−0.88
−0.90
−0.92
−0.94
−0.96
0
2
4
6
8
10
Iteration

1.7 Aplicación de Optimización Bayesiana en Aprendizaje Automático

1.7.1 Diabetes

En esta sección, aplicaremos la optimización bayesiana a un problema de regresión. La regresión se realiza en un pequeño conjunto de datos sobre diabetes que es parte de scikit-learn.

El conjunto de datos, contiene 10 variables basales como son: edad, sexo, índice de masa corporal, presión arterial media y 6 mediciones de suero sanguíneo para cada uno de los 442 pacientes diabéticos, así como la respuesta del paciente, una medida cuantitativa de la progresión de la enfermedad, un año después de la toma inicial.

Atributo	Descripción
age	age in years
sex	sex
bmi	body mass index
bp	average blood pressure
s1 tc	total serum cholesterol
s2 ldl	low-density lipoproteins
s3 hdl	high-density lipoproteins
s4 tch	total cholesterol / HDL
s5 ltg	possibly log of serum triglycerides level
s6 glu	blood sugar level
y	quantitative measure of disease progression one year after baseline

	AGE	SEX	BMI	BP	··· Serum Measurements ···						Response
Patient	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	y
1	59	2	32.1	101	157	93.2	38	4	4.9	87	151
2	48	1	21.6	87	183	103.2	70	3	3.9	69	75
3	72	2	30.5	93	156	93.6	41	4	4.7	85	141
4	24	1	25.3	84	198	131.4	40	5	4.9	89	206
5	50	1	23.0	101	192	125.4	52	4	4.3	80	135
6	23	1	22.6	89	139	64.8	61	2	4.2	68	97
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
441	36	1	30.0	95	201	125.2	42	5	5.1	85	220
442	36	1	19.6	71	250	133.2	97	3	4.6	92	57

Fuente Diabetes Study

1. Definición del problema El objetivo, es predecir la progresión de la enfermedad un año después del estudio, basándose en múltiples características explicativas proporcionadas en el conjunto de datos, utilizando un `XGBRegressor` con optimización bayesiana para la búsqueda de hiperparámetros. En esta sección demostraremos cómo optimizar los hiperparámetros de un `XGBRegressor` con GPyOpt y skopt y cómo se compara el rendimiento de la optimización bayesiana `BayesianOptimization` con la búsqueda aleatoria `RandomSearch`.

Matemáticamente, esto se modela como:

$$Y = f(X; \Theta) + \epsilon$$

Donde: - $Y \in \mathbb{R}$ representa la variable objetivo (medida de progreso de la enfermedad). - $X \in \mathbb{R}^{n \times d}$ es la matriz de entrada con n muestras y d características. - Θ representa los hiperparámetros del modelo que afectan su capacidad de generalización. - ϵ es un término de error (ruido) que modela incertidumbre o variabilidad no explicada por $f(X; \Theta)$.

El modelo objetivo $f(X; \Theta)$ es un `XGBRegressor` (modelo de boosting basado en árboles) que tiene múltiples hiperparámetros a optimizar.

2. Función objetivo para la optimización

El problema de ajuste de hiperparámetros se puede expresar como un problema de minimización, donde se busca el conjunto de hiperparámetros Θ^* que minimice el error cuadrático medio (MSE) en validación cruzada:

$$\Theta^* = \underset{\Theta}{\text{argmin}} \frac{1}{k} \sum_{i=1}^{k} \left(\frac{1}{n_i} \sum_{j=1}^{n_i} \left(y_{ij} - \hat{y}_{ij}(\Theta) \right)^2 \right)$$

Donde: - k es el número de pliegues en la validación cruzada. - n_i es el número de muestras en el pliegue i. - y_{ij} es el valor real de la muestra j en el pliegue i. - $\hat{y}_{ij}(\Theta)$ es la predicción realizada por el modelo con hiperparámetros Θ.

3. Hiperparámetros optimizados

Los hiperparámetros ajustados (Θ) incluyen: 1. **Tasa de aprendizaje (η)**: Controla la contribución de cada árbol en el modelo final. - Rango: $[0.01, 1.0]$ 2. **Profundidad máxima de los árboles (d)**: Controla la complejidad de los árboles. - Rango: $[1, 50]$ 3. **Número de estimadores (n_{trees})**: Número total de árboles en el modelo. - Rango: $[1, 300]$ 4. **Peso mínimo de las hojas (w_{min})**: Peso mínimo requerido para dividir un nodo. - Rango: $[1, 10]$ 5. **Parámetro γ**: Penalización por complejidad en la división de nodos. - Rango: $[0, 5]$

El espacio de búsqueda para los hiperparámetros es definido explícitamente en la optimización bayesiana.

4. Enfoque de optimización

Se utiliza la optimización bayesiana para encontrar Θ^*. Este enfoque construye un modelo probabilístico (Proceso Gaussiano) para aproximar la relación entre los hiperparámetros Θ y la función objetivo (MSE). La técnica, guía la búsqueda hacia regiones prometedoras del espacio de hiperparámetros.

1. **Modelo probabilístico**: El modelo probabilístico del MSE se denota como:

$$f(\Theta) \sim \mathcal{GP}(\mu(\Theta), k(\Theta, \Theta'))$$

   Donde:

   - $\mu(\Theta)$: Media predictiva del MSE.
   - $k(\Theta, \Theta')$: Función de covarianza que mide la similitud entre puntos Θ y Θ'.

2. **Función de adquisición**: La función de adquisición (Expected Improvement, EI) guía la selección del próximo conjunto de hiperparámetros a evaluar:

$$\text{EI}(\Theta) = \mathbb{E}\left[\text{max}(0, f(\Theta^*) - f(\Theta))\right]$$

   Donde:

   - $f(\Theta^*)$ es el mejor valor conocido de la función objetivo.
   - $f(\Theta)$ es el valor predicho por el modelo probabilístico.

3. **Actualización iterativa**: En cada iteración:

   - Se evalúa EI(Θ) en el espacio de búsqueda.
   - Se selecciona el conjunto de hiperparámetros Θ_{next} que maximiza EI(Θ).
   - Se entrena el modelo con Θ_{next} y se actualiza el modelo probabilístico.

1.7.1.1 Ajuste de hiperparámetros con búsqueda aleatoria

Para el ajuste de hiperparámetros con búsqueda aleatoria, utilizamos `RandomSearchCV` de scikit-learn y calculamos una puntuación de validación cruzada para cada punto seleccionado aleatoriamente en el espacio de hiperparámetros.

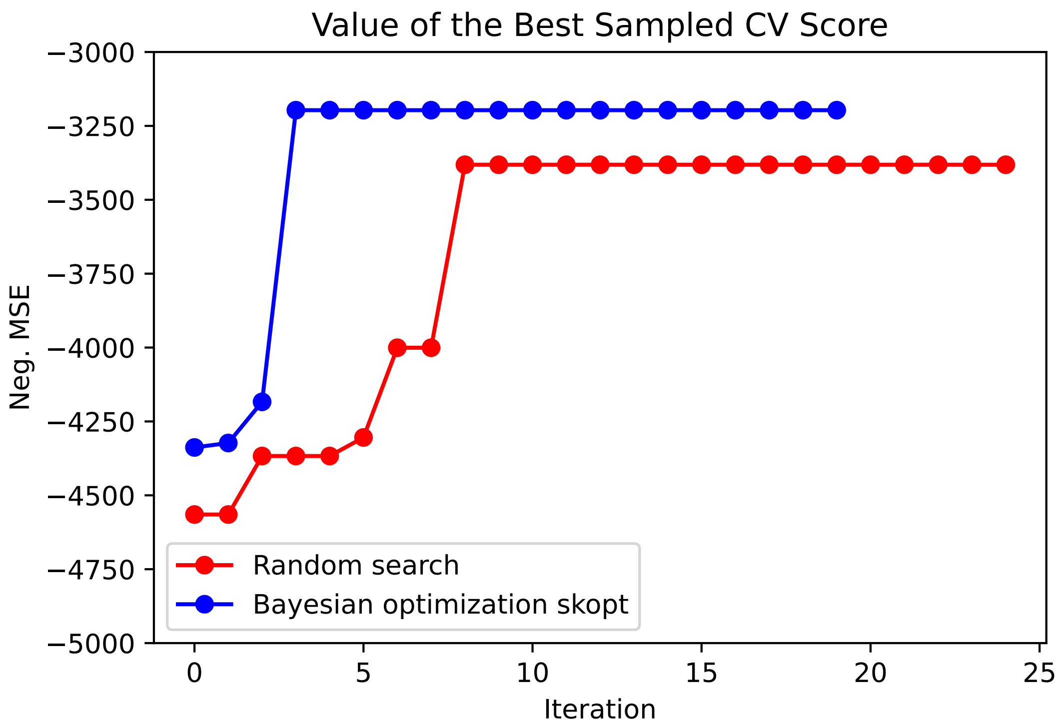
1.7.1.2 Ajuste de hiperparámetros con optimización bayesiana

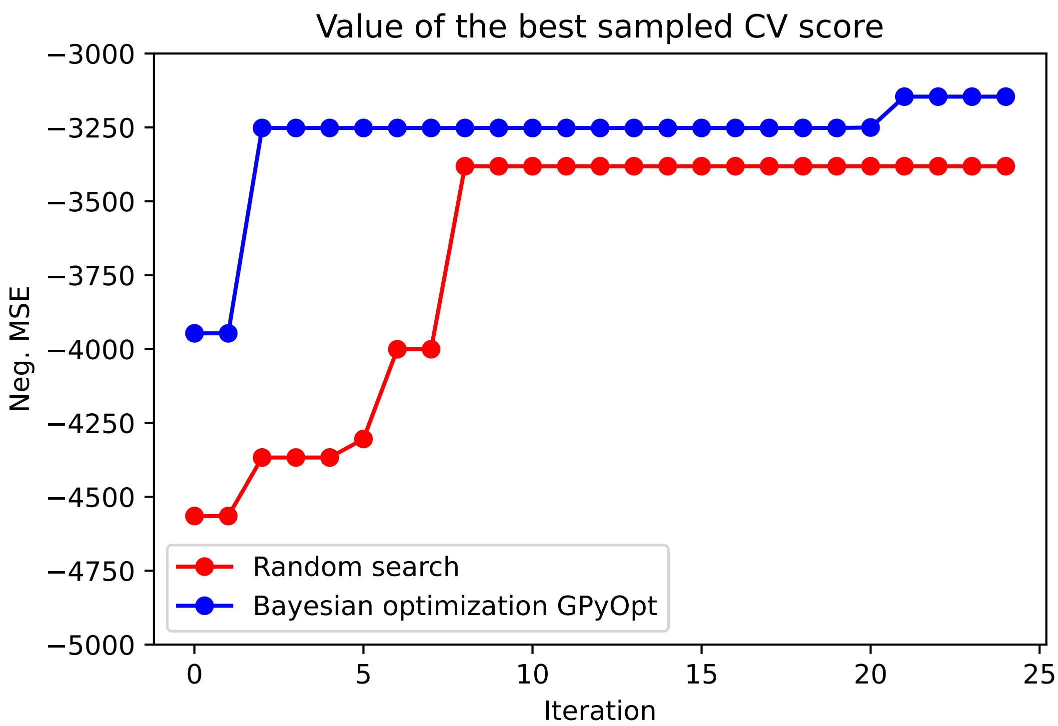
Para ajustar los hiperparámetros con optimización bayesiana, implementamos una función objetivo `cv_score` que toma los hiperparámetros como entrada y devuelve una puntuación de validación cruzada. Aquí, asumimos que la validación cruzada en un punto determinado en el espacio de hiperparámetros es determinista y, por lo tanto, establecemos el parámetro `exact_feval` de `BayesianOptimization` en `True`. Dependiendo del ajuste del modelo y los detalles de la validación cruzada, esto podría no ser el caso, pero lo ignoraremos aquí.

	Hyperparameter	Value
0	Learning Rate	0.055993
1	Gamma	2.213313
2	Max Depth	2.000000
3	N Estimators	88.000000
4	Min Child Weight	3.000000
5	Best Neg. MSE	3196.807177

1.7.1.3 Resultados

En promedio, la optimización bayesiana encuentra un óptimo mejor en una menor cantidad de pasos que la búsqueda aleatoria y supera la línea base en casi todas las ejecuciones. Esta tendencia se vuelve aún más prominente en espacios de búsqueda de dimensiones superiores. Aquí, el espacio de búsqueda es de cinco dimensiones, lo cual es bastante bajo para obtener un beneficio sustancial de la optimización bayesiana. Una ventaja de la búsqueda aleatoria es que es fácil de paralelizar. La paralelización de la optimización bayesiana es mucho más difícil y está sujeta a investigación.





	Method	Neg. MSE
0	Baseline	4000.175
1	Random Search	-3381.370
2	Bayesian Optimization (skopt)	-3196.807
3	Bayesian Optimization (GPyOpt)	-3145.947

1.7.2 California Housing Prices

En este ejemplo, aplicaremos la optimización bayesiana a un problema de regresión de precios de vivienda en California. El conjunto de datos se puede descargar de Kaggle y contiene información sobre la población, los ingresos medios, los precios de las viviendas, etc. El objetivo, es predecir los precios de las viviendas en función de las características proporcionadas.

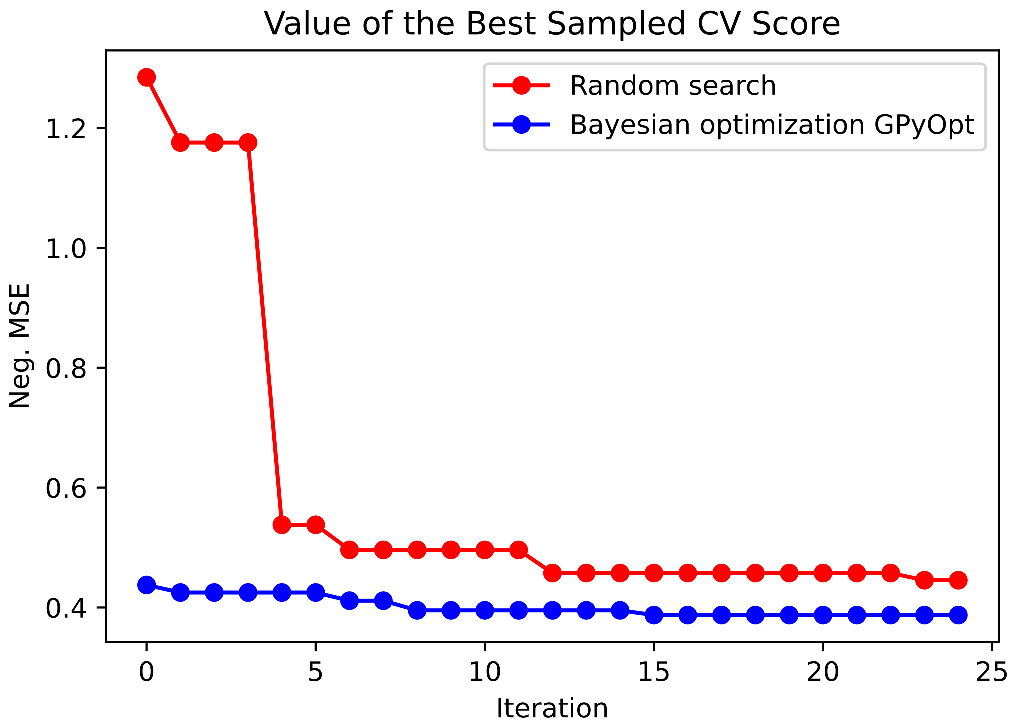
1.7.2.1 Ajuste de hiperparámetros con búsqueda aleatoria

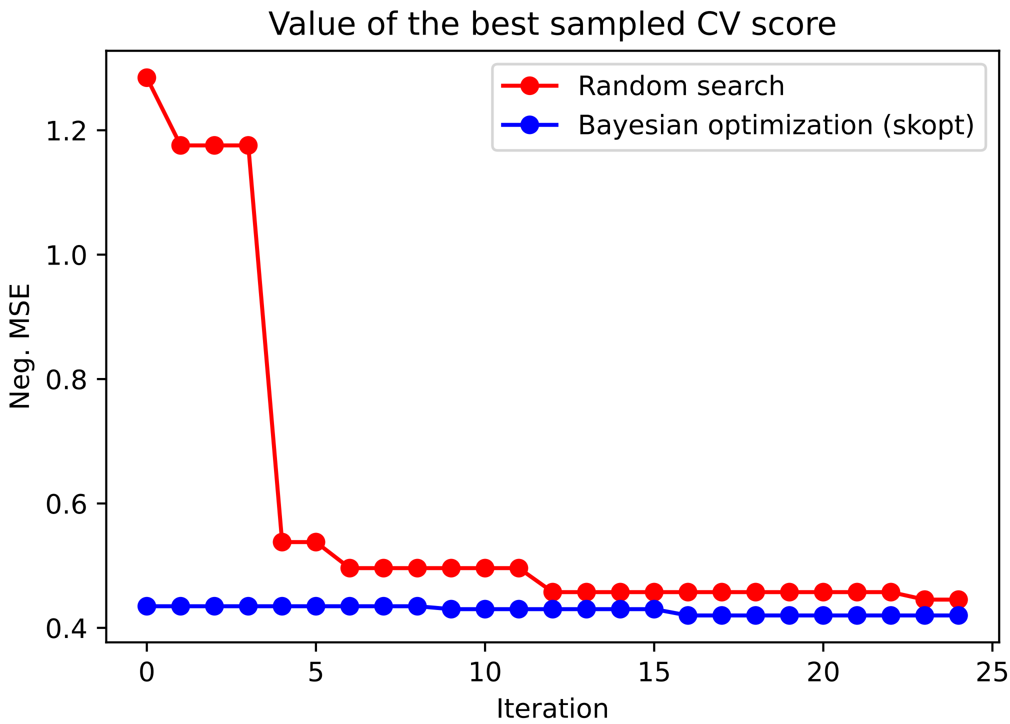
Para el ajuste de hiperparámetros con búsqueda aleatoria, utilizamos `RandomSearchCV` de scikit-learn y calculamos una puntuación de validación cruzada para cada punto seleccionado aleatoriamente en el espacio de hiperparámetros. El modelo utilizado fue `GradientBoostingRegressor` de scikit-learn.

1.7.2.2 Ajuste de hiperparámetros con optimización bayesiana

Para ajustar los hiperparámetros con optimización bayesiana, implementamos una función objetivo `cv_score` que toma los hiperparámetros como entrada y devuelve una puntuación de validación cruzada. El modelo, empleado fue `GradientBoostingRegressor` de scikit-learn.

```
Baseline neg. MSE = 0.45815
Random search neg. MSE = 0.44542
Bayesian optimization GPyOpt neg. MSE = 0.38734
```





	Method	Neg. MSE
0	Baseline	0.45815
1	Random Search	0.44542
2	Bayesian Optimization (skopt)	0.41997
3	Bayesian Optimization (GPyOpt)	0.38734

2 Fuentes

- [#] Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statistics (with discussion), 407-499. (https://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)