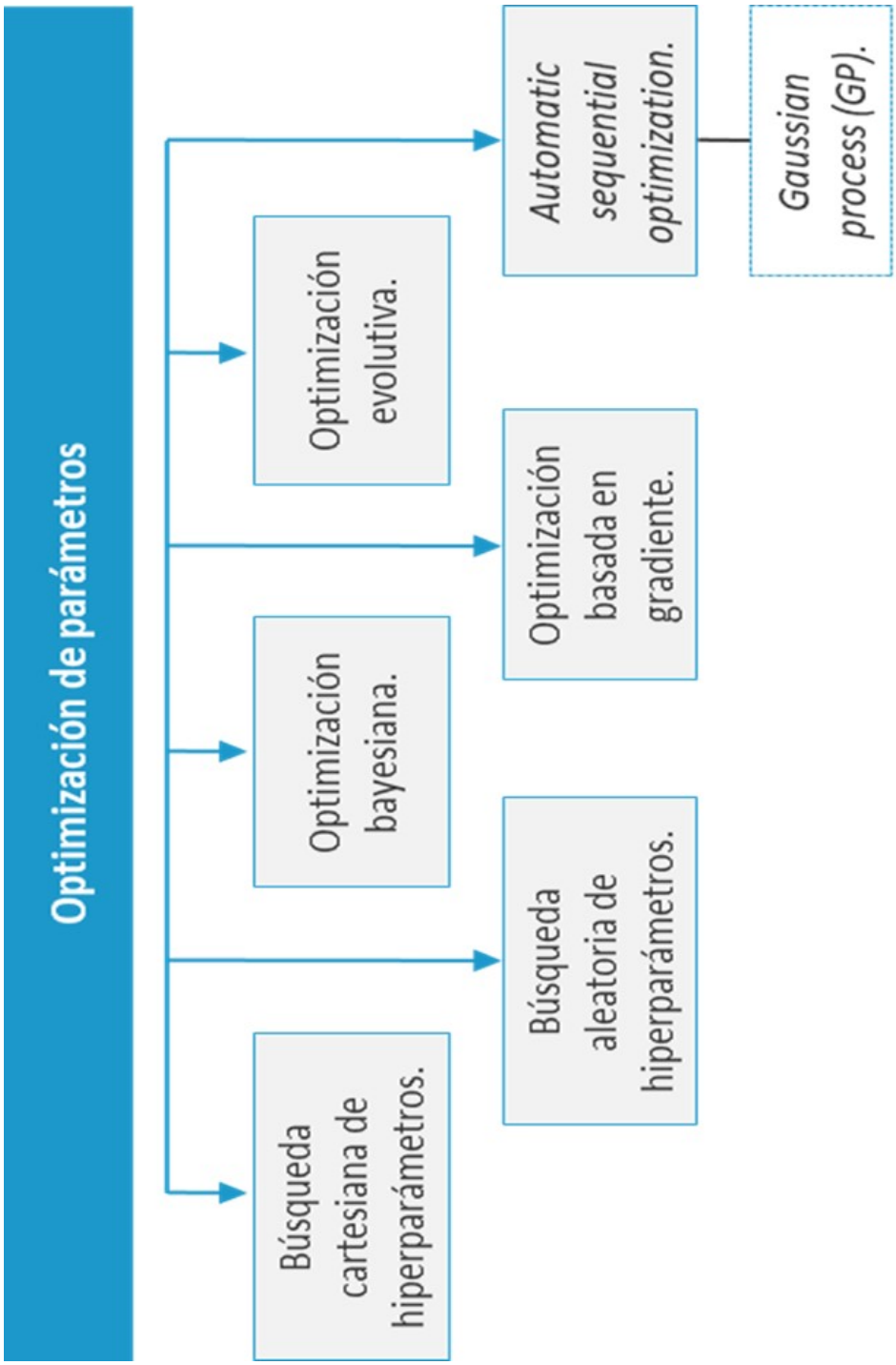
Técnicas de Aprendizaje Automático

Tema 11. Parametrización automática y optimización de algoritmos

Índice

Esquema



11.1. Introducción y objetivos

En este tema nos vamos a centrar en el **proceso de optimización de los hiperparámetros** de un modelo de aprendizaje supervisado.

La optimización de hiperparámetros es un proceso crucial en el desarrollo de modelos de aprendizaje supervisado. Este proceso busca encontrar la configuración óptima de parámetros que maximice el rendimiento predictivo del modelo. Los hiperparámetros son configuraciones que no se aprenden directamente del conjunto de datos, sino que se establecen antes del proceso de entrenamiento del modelo. Hasta ahora, se han presentado los diferentes algoritmos con los hiperparámetros más habituales en cada uno de ellos, y un proceso de prueba-error en el que se modificaba el valor de dichos hiperparámetros. En este tema, presentamos cómo hacer que ese proceso sea semiautomático. Algunos ejemplos comunes de hiperparámetros incluyen la tasa de aprendizaje en algoritmos de aprendizaje automático, el número de árboles en un bosque aleatorio o el tamaño del vecindario en un algoritmo de vecinos más cercanos.

El proceso de optimización de hiperparámetros implica explorar un espacio de búsqueda de valores para encontrar la combinación óptima de hiperparámetros que minimice una función de pérdida o maximice una métrica de evaluación específica, como la precisión, el área bajo la curva ROC o la F1-score. Este proceso puede ser computacionalmente costoso y requiere técnicas eficientes para buscar en el espacio de hiperparámetros de manera efectiva.

Las estrategias comunes para la optimización de hiperparámetros incluyen la búsqueda aleatoria, la búsqueda en cuadrícula o en rejilla y la optimización bayesiana. La **búsqueda aleatoria** explora aleatoriamente el espacio de hiperparámetros, mientras que la **búsqueda en cuadrícula** evalúa sistemáticamente todas las combinaciones posibles dentro de un rango predefinido. Por otro lado, la

Ideas clave

optimización bayesiana utiliza el conocimiento acumulado de las iteraciones anteriores para enfocar la búsqueda en áreas prometedoras del espacio de hiperparámetros.

La optimización de hiperparámetros es un proceso esencial para mejorar el rendimiento de los modelos de aprendizaje supervisado, y la selección cuidadosa de los hiperparámetros puede tener un impacto significativo en la capacidad predictiva y generalización del modelo.

En este tema, en primer lugar, se va a realizar una definición del concepto de **hiperparámetros** y de su importancia. A continuación, se van a describir los métodos existentes para realizar la optimización de los hiperparámetros. Brevemente se va a comentar la optimización bayesiana, la optimización por gradiente y la optimización evolutiva. Posteriormente, con un mayor nivel de detalle se va a describir la optimización cartesiana basada en rejilla (*grid search*). Seguidamente, se detallará el proceso de optimización basada en la búsqueda aleatoria (*random search*). Finalmente, se describen algunas técnicas o métodos para mejorar las técnicas descritas, así como los escenarios en las cuales tienen utilidad.

Los **objetivos** del presente tema son los siguientes:

- Afianzar el concepto de hiperparámetros e identificar los principales hiperparámetros de los algoritmos de aprendizaje supervisado estudiados hasta ahora.
- Entender la búsqueda de hiperparámetros mediante la optimización bayesiana, la optimización por gradiente y la optimización evolutiva.
- Entender y aplicar los métodos de optimización basada en rejilla y optimización por búsqueda aleatoria.
- Conocer los métodos de optimización evolutiva.

11.2. Concepto de hiperparámetros

Prácticamente todos los algoritmos de aprendizaje automático tienen en mayor o menor medida **una serie de parámetros**. Estos parámetros que son específicos de cada modelo y que sirven para modificar diversos aspectos de cómo los algoritmos «aprenden» se conocen con el nombre de hiperparámetros (*hyperparamters* en inglés).

En la fase de entrenamiento de los modelos es común ir iterando con distintos valores de los hiperparámetros sobre los datos de entrenamiento e ir comparando los resultados de la aplicación de los modelos sobre el conjunto de prueba. Cuando estos resultados no son satisfactorios, una forma de mejorarlos es por medio de la modificación de los parámetros del modelo, los hiperparámetros. Este proceso de iteración sobre distintos valores de hiperparámetros se conoce como optimización de hiperparámetros (*hyperparameter tuning*). El principal objetivo de esta optimización es encontrar la configuración óptima de hiperparámetros que maximice el rendimiento del modelo en un conjunto de datos de prueba o validación. Sin embargo, aunque la optimización de hiperparámetros no se utiliza directamente para controlar el *overfitting* o el *underfitting* del modelo, es posible que, al encontrar una configuración óptima, podría indirectamente mejorar el control del *overfitting* o *underfitting* al mejorar el rendimiento general del modelo.

Esta **optimización de hiperparámetros** es una parte importante de la construcción de modelos de aprendizaje automático; hay personas que también la definen como una **fase de ensayo/error.**

Muchas veces, el problema radica en que existen muchos parámetros diferentes y es difícil encontrar el valor óptimo de todos ellos. El problema es precisamente elegir el **conjunto óptimo de hiperparámetros** para ese algoritmo de aprendizaje dado un conjunto de datos determinado.

Ideas clave

El método tradicional para seleccionar los valores de los hiperparámetros ha consistido en **entrenar de forma individual diferentes modelos**, cada uno con unos valores de los parámetros, y elegir el mejor modelo. A medida que el número de parámetros y los valores que se quieren probar crece, esto se vuelve poco práctico y muy difícil de gestionar.

Existen **diferentes enfoques o métodos** para realizar esta optimización, como la búsqueda cartesiana o por rejilla, la búsqueda aleatoria, la optimización bayesiana, la optimización basada en gradiente o la optimización evolutiva. A continuación, veremos la búsqueda en rejilla y la búsqueda aleatoria con más detalle. En cuanto a las otras técnicas vamos a proporcionar una breve introducción:

- **Optimización bayesiana:** es una metodología para optimizar funciones de caja negra ruidosas. Consiste en desarrollar un modelo estadístico de la función de los valores de los hiperparámetros a la función objetivo del conjunto de validación. El método asume que existe una función suave pero ruidosa que mapea los hiperparámetros al objetivo. Se basa en la suposición de que existe una probabilidad *a priori* de las funciones con determinados valores de parámetros y sus salidas, dando lugar a una distribución sobre las funciones. El proceso iterativamente va estableciendo parámetros para observar su resultado.

- **Optimización basada en gradiente:** en algunos algoritmos de aprendizaje automático es posible calcular el gradiente respecto de los hiperparámetros y, por tanto, optimizar los valores de los hiperparámetros utilizando la técnica de descenso del gradiente.

- **Optimización evolutiva:** consiste en utilizar algoritmos evolutivos para buscar el espacio de los parámetros de un algoritmo determinado. Este proceso está inspirado en el concepto biológico de la evolución.

Ideas clave

Prácticamente todos los algoritmos de aprendizaje supervisado tienen algunos parámetros que afectan al proceso de aprendizaje del algoritmo sobre los datos, como, por ejemplo, el número de árboles de un *random forest*. El conjunto de las **combinaciones de todos estos valores** es lo que se conoce como hiperparámetros. En la mayoría de los algoritmos existen una serie de hiperparámetros por defecto, pero para obtener un mejor resultado es necesario optimizarlos.

Los **hiperparámetros** son una serie de parámetros que afectan al proceso de aprendizaje de los algoritmos supervisados sobre los datos. No se trata de parámetros que los algoritmos aprendan de los datos, sino de parámetros que es necesario establecer *a priori*. Un problema común consiste en encontrar los mejores valores de estos conjuntos de parámetros. A este proceso se le conoce con el nombre de **optimización de hiperparámetros**.

Una cuestión importante que tener en cuenta en la fase de optimización de hiperparámetros es **controlar el sobreajuste** u *overfitting*. El sobreajuste en los datos de entrenamiento implica que el algoritmo ha memorizado todos los detalles del conjunto de datos en lugar de obtener los patrones de los datos que también aplican a datos futuros sobre los cuales se desean realizar las predicciones.

RESUMEN DE ALGUNOS DE LOS HIPERPARÁMETROS POR ALGORITMO

REGRESIÓN LINEAL	• Ajuste de Intercepción (fit_intercept): booleano para indicar si se debe calcular la intercepción para este modelo. • Método de Optimización (solver): algoritmo utilizado para la optimización de los pesos. Ejemplos: 'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'.
SVM	• Tipo de Kernel (kernel): especifica el tipo de kernel a utilizar en el algoritmo. Ejemplos: 'linear', 'poly', 'rbf', 'sigmoid'. • Parámetro de Regularización (C): parámetro de penalización del término de error.
ÁRBOLES DE DECISIÓN	• Criterio de División (criterion): medida utilizada para evaluar la calidad de una división. Ejemplos: 'gini', 'entropy'. • Profundidad Máxima del Árbol (max_depth): profundidad máxima del árbol de decisión.
RANDOM FOREST	• Número de Árboles (n_estimators): número de árboles en el bosque. • Características a Considerar en Cada División (max_features): número de características a considerar al buscar la mejor división.
NAIVE BAYES	• Tipo de Distribución (distribution): tipo de distribución a suponer para las características. Ejemplos: 'gaussian' (para datos continuos), 'bernoulli' (para datos binarios), 'multinomial' (para datos discretos). • Suavizado (alpha): parámetro de suavizado para evitar la probabilidad cero.

Tabla 1. Visión general de los hiperparámetros más comunes para los algoritmos de machine learning indicados. Fuente: elaboración propia.

En la Tabla 1 se presenta una visión general de algunos de los hiperparámetros más comunes para los principales algoritmos de *machine learning* estudiados en los temas previos. Es importante tener en cuenta que pueden existir otros hiperparámetros específicos o adicionales dependiendo de la implementación de la biblioteca de *machine learning* utilizada.

11.3. Búsqueda cartesiana de hiperparámetros

Este método de búsqueda también es conocido con el nombre de **búsqueda de rejilla cartesiana** o bien como *cartesian grid search* o *grid search* en inglés. Se trata de un **tipo de optimización de hiperparámetros** de los modelos donde los ingenieros de *machine learning* especifican una serie de valores para cada uno de los parámetros del modelo que desean optimizar. A continuación, es necesario entrenar un modelo para cada una de las combinaciones de los valores de los hiperparámetros. Por ejemplo, en el caso de que tengamos tres hiperparámetros y se hayan especificado 5, 10 y 2 valores distintos para cada uno de ellos, será necesario crear 5*10*2=100 modelos diferentes con cada una de las combinaciones de los parámetros.

Se trata de una **búsqueda exhaustiva de un subconjunto de hiperparámetros de un algoritmo de aprendizaje automático** que ha sido manualmente definido. Este algoritmo de búsqueda se debe guiar por alguna métrica de rendimiento normalmente con técnicas de validación cruzada sobre los datos de entrenamiento o sobre un conjunto independiente.

Como es posible que algunos parámetros admitan valores reales o sin límite, es **necesario establecer rangos** antes de poder aplicar este método de optimización.

Por ejemplo, en una máquina de vector de soporte con un *kernel* de base radial, es necesario especificar el valor de la constante de regularización C y el parámetro del *kernel* γ. Como ambos parámetros son continuos es necesario establecer una serie de valores para probar como, por ejemplo:

- $C \in \{10,100,1000\}$
- $\gamma \in \{0.1,0.2,0.6\}$

El método cartesiano entrena una SVM con cada par de valores (C,γ) y evalúa su rendimiento en un conjunto de test, o bien en validación cruzada. Finalmente, el algoritmo muestra la combinación que ha aportado un mejor resultado.

A modo de resumen, el proceso de búsqueda *gridsearch* se realiza de la siguiente manera y puede verse resumido en la Figura 1:

- **Definición de la cuadrícula o rejilla de hiperparámetros:** se especifican los hiperparámetros que se desean ajustar y se define un conjunto de valores posibles para cada uno de ellos. Por ejemplo, para un modelo de árbol de decisión, se podrían definir los valores posibles para la profundidad máxima del árbol y el criterio de división.

- **Generación de las combinaciones de hiperparámetros:** se genera una cuadrícula con todas las posibles combinaciones de valores de hiperparámetros. Esto implica tomar todos los valores posibles de cada hiperparámetro y combinarlos entre sí para formar todas las combinaciones posibles.

- **Entrenamiento y evaluación de modelos:** para cada combinación de hiperparámetros se entrena un modelo utilizando esos valores y se evalúa su rendimiento utilizando una métrica de evaluación, como precisión, exactitud o F1-score. Esto se realiza mediante validación cruzada o dividiendo el conjunto de datos en un conjunto de entrenamiento y un conjunto de validación.

- **Selección del mejor modelo:** después de evaluar todas las combinaciones de hiperparámetros, se selecciona la combinación que produjo el mejor rendimiento según la métrica de evaluación especificada.

La búsqueda *gridsearch* es exhaustiva y garantiza que se explore todo el espacio de hiperparámetros definido. Sin embargo, puede ser computacionalmente costosa, especialmente cuando se tienen muchos hiperparámetros o valores posibles para cada uno.

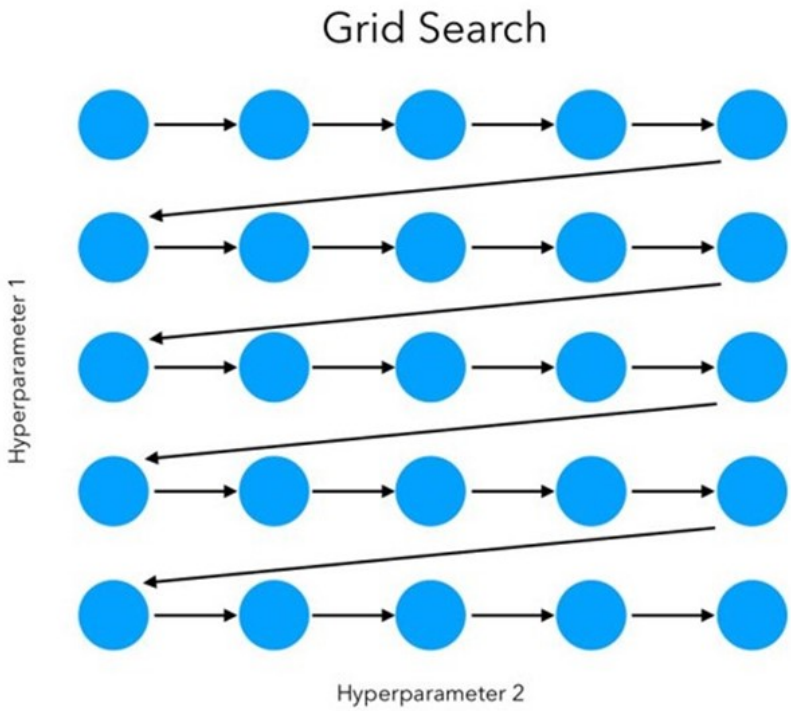


Figura 1. Diagrama de una búsqueda cartesiana en rejilla bidimensional entre dos hiperparámetros.

Fuente: Maël Fabien, s.f.

El método de búsqueda *gridsearch* sufre el problema de *curse of dimensionality* o **maldición de la dimensión**. Pero, por otro lado, se trata de un método que se puede paralelizar muy bien pues la evaluación de cada uno de los conjuntos de parámetros es independiente.

11.4. Búsqueda aleatoria de parámetros

Debido a que el método de búsqueda cartesiana es exhaustivo y costoso computacionalmente, han surgido alternativas como la búsqueda aleatoria. En el caso de la **búsqueda aleatoria de hiperparámetros** o *random search* también es necesario establecer un rango para aquellos parámetros que se quieran optimizar. La diferencia radica en que, en lugar de buscar todas las posibles combinaciones, el proceso realiza un muestreo uniforme sobre todas las posibles combinaciones de parámetros. En la Figura 2 se presenta un ejemplo de este tipo de búsqueda para dos hiperparámetros.

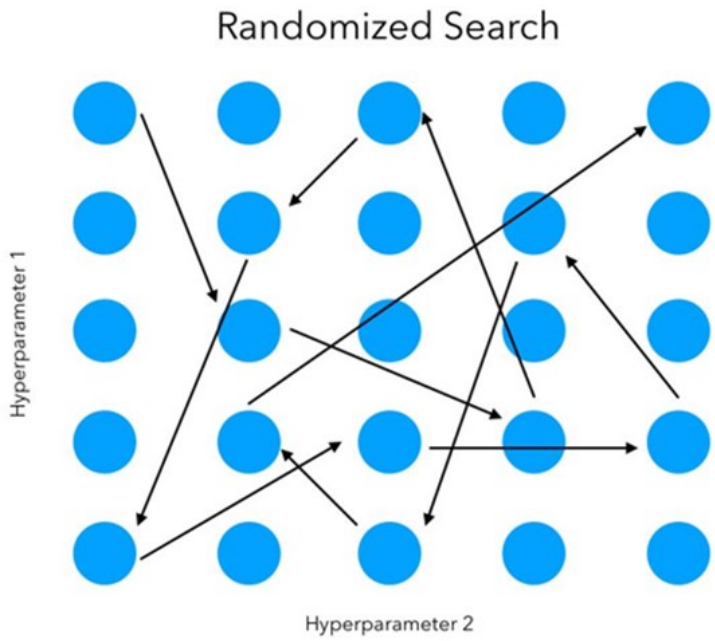


Figura 2. Diagrama de una búsqueda aleatoria bidimensional entre dos hiperparámetros. Fuente: Maël Fabien, s.f.

Además, este proceso suele tener también un **criterio de parada** donde el ingeniero de *machine learning* establece con qué umbral se puede detener el proceso. Por ejemplo, el ingeniero puede especificar un número máximo de modelos o un número

máximo de tiempo permitido para la búsqueda. O bien, se puede especificar un criterio de parada basado en una métrica de rendimiento (el AUC, por ejemplo) que haría que el proceso termine si se mejora el valor en un porcentaje determinado.

Dado un conjunto de tiempo finito, hacer elecciones aleatorias dentro de unos valores generalmente proporciona mejores resultados que una búsqueda cartesiana exhaustiva.

Este proceso **muestrea los parámetros un número de veces determinado** y se ha demostrado que es más efectivo en escenarios con altas dimensiones que en la búsqueda cartesiana. Este motivo se debe a que la mayoría de las veces los parámetros no afectan demasiado a la función de pérdida. Por tanto, valores dispersados aleatoriamente proporcionan una mejor composición que buscar de forma exhaustiva todos los parámetros. Bergstra y Bengio en el artículo *Random Search for HyperParameter Optimization* escribieron: «Compared with neural networks configured by a pure grid search, we find that random search over the same domain is able to find models that are as good or better within a fraction of the computation time».

La cuestión es que para la mayoría de los conjuntos de datos **solo algunos pocos parámetros son relevantes**. Este hecho hace que la búsqueda cartesiana sea una opción poco eficiente para la optimización de algoritmos.

Con la **búsqueda aleatoria** se hace una pequeña reducción de eficiencia en aquellos algoritmos con pocos parámetros y una gran mejora de eficiencia en aquellos con un gran número de parámetros.

Una vez que se ha hecho una búsqueda aleatoria se puede hacer zoom sobre aquellas regiones del espacio de hiperparámetros que parezcan prometedoras. Esto se puede llevar a cabo haciendo una nueva búsqueda aleatoria, con el método cartesiano o bien de forma manual.

Por ejemplo, si se ha realizado una primera búsqueda aleatoria con los valores de 0,0, 0,25, 0,5, 0,75 y 1,0 y los valores del medio parecen prometedores, se puede hacer una segunda búsqueda más fina sobre 0,3, 0,4, 0,5, 0,6, 0,7.

Criterios de parada

Es habitual establecer algún criterio de parada a la hora de buscar los parámetros óptimos. Por lo general, el criterio de *early-stopping* de una métrica determinada combinada con el **tiempo máximo** de ejecución es el mejor criterio. El número de modelos que son necesarios para converger a un óptimo global puede variar bastante, y el criterio de parada basado en la métrica de evaluación tiene en consideración este aspecto de forma que se detiene cuando la gráfica de error se estabiliza.

Número de modelos

El número de modelos que se requiere para converger depende de varios factores, pero principalmente en la «forma» de la función de error del espacio de hiperparámetros. Mientras que la mayoría de los algoritmos se comportan razonablemente bien en una gran región del espacio de hiperparámetros, en la mayoría de los conjuntos de datos algunas combinaciones de conjuntos de datos y algoritmos son muy sensibles: tienen unas funciones de error con muchos picos.

11.5. Mejora de la búsqueda de hiperparámetros

Una forma de mejorar la búsqueda aleatoria de parámetros es **elegir conjuntos de parámetros que cubran el espacio de forma más eficiente** que si se eligen de forma aleatoria. Bersgtra y Bengio realizaron esta prueba y encontraron una mejora potencial, pero únicamente cuando se hacían búsquedas de entre 100 y 500 modelos. Esto se debe principalmente a que el número de parámetros que suelen ser importantes para un conjunto de datos particular suele ser pequeño (1-4) y el proceso de búsqueda aleatoria lo cubre bastante bien, lo cual se representa gráficamente en la Figura 3.

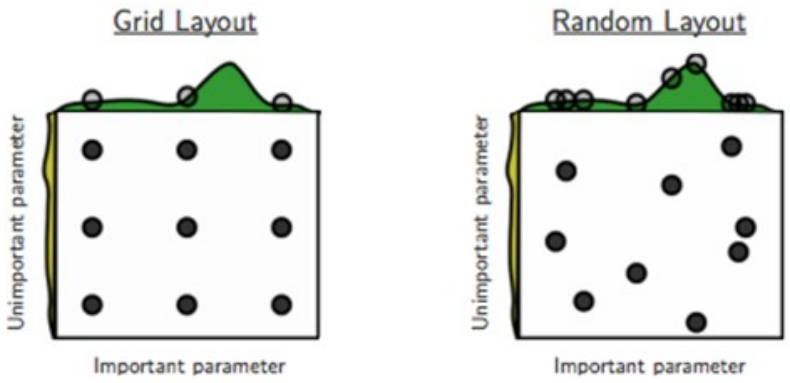


Figura 3. Búsqueda de hiperparámetros con el método cartesiano y el método aleatorio utilizando nueve ejecuciones. Fuente: Bergstra y Bengio, 2012.

En la imagen, observamos que el método aleatorio es capaz de encontrar el punto óptimo mientras que el método por rejilla no lo consigue.

Sin embargo, para algunos algoritmos complejos, como las *deep belief networks*, la búsqueda aleatoria de hiperparámetros puede ser insuficiente. Para estos algoritmos se utiliza una técnica denominada *automatic sequential optimization*, que consiste en **construir un modelo del espacio de hiperparámetros** y utilizarlo para guiar el proceso de búsqueda. La técnica más conocida de este tipo de métodos son los

modelos de procesos gaussianos (*gaussian process models*).

Los modelos de **optimización bayesiana** son una clase de métodos de optimización que se basan en el teorema de Bayes para encontrar la configuración óptima de los hiperparámetros de un modelo de aprendizaje automático. Estos modelos utilizan técnicas probabilísticas para modelar la relación entre los hiperparámetros y la función objetivo que se desea optimizar.

El proceso de optimización bayesiana comienza con la especificación de una distribución inicial sobre los hiperparámetros, llamada «prior». Luego, a medida que se observan los resultados de evaluación del modelo, esta distribución se actualiza para reflejar la información recopilada, lo que se conoce como «posterior». Utilizando esta distribución posterior, se puede tomar una decisión informada sobre cuáles hiperparámetros explorar en la siguiente iteración del proceso de optimización.

Los modelos de optimización bayesiana pueden ser especialmente útiles cuando la función objetivo es costosa de evaluar, ya que intentan minimizar la cantidad de evaluaciones necesarias para encontrar la configuración óptima de hiperparámetros. Al incorporar información previa y actualizarla con cada evaluación del modelo, estos modelos pueden realizar un muestreo eficiente del espacio de hiperparámetros y converger rápidamente hacia la solución óptima.

Algunos algoritmos comunes utilizados en la optimización bayesiana incluyen el Proceso Gaussiano (Gaussian Process) (Williams y Rasmussen, 2006), el Proceso Gaussiano de Matern (*Matern Gaussian Process*) (Borovitskiy, Azangulov, Terenin, Mostowsky, Deisenroth y Durrande, 2021), y métodos basados en árboles como el Proceso de Árboles Probabilísticos (*Probabilistic Tree Process*) y Bosques Aleatorios Bayesianos (*Bayesian Random Forests*) (Wu, Chen, Zhang, Xiong, Lei y Deng, 2019).

Ideas clave

Para finalizar, podemos decir que los modelos de optimización bayesiana son una eficaz herramienta para la optimización de hiperparámetros en el aprendizaje automático, que utilizan inferencia probabilística para explorar de manera eficiente el espacio de hiperparámetros y encontrar configuraciones óptimas en menos evaluaciones.

11.6. Cuaderno de ejercicios

- Realiza una búsqueda en rejilla de hiperparámetros para un modelo de Random Forest, haciendo uso del dataset de scikit learn breast cancer. Los hiperparámetros deben tener al menos tres valores cada uno.

SOLUCIÓN

```
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer(as_frame=True)

df_features = pd.DataFrame(data.data, columns = data.feature_names)

df_target = pd.DataFrame(data.target, columns=['target'])

df_target['target'].value_counts()from sklearn.ensemble import

df = pd.concat([df_features, df_target], axis=1)

X = df.drop(['target'], axis = 1)

y = df['target']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3,
random_state = 0)

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

# Hyperparameter Optimization

parameters = {'n_estimators': [4, 6],#, 9, 10, 15],

              'max_features': ['log2'],#, 'sqrt','auto'],
```

```
    'criterion': ['entropy', 'gini'],
    'max_depth': [2, 3, 5, 10],
    'min_samples_split': [2, 3, 5],
    'min_samples_leaf': [1, 5, 8]
}

# Run the grid search
grid_obj = GridSearchCV(rf, parameters)
grid_obj = grid_obj.fit(X_train, y_train)
# Set the rf to the best combination of parameters
rf = grid_obj.best_estimator_
```

- ¿Cuáles son los mejores hiperparámetros para el modelo?

SOLUCIÓN

Solución para la combinación de valores indicada en el código propuesto: RandomForestClassifier(criterion='entropy', max_depth=10, max_features='log2', min_samples_split=3, n_estimators=6)

- Realiza la misma búsqueda en rejilla que en el ejercicio 1, pero esta vez haciendo uso de un modelo SVM.

```
from sklearn import svm
svc = svm.SVC()
```

```
# Hyperparameter Optimization

parameters = [

  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},

  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},

]



# Run the grid search

grid_obj = GridSearchCV(svc, parameters, verbose=5)

grid_obj = grid_obj.fit(X_train, y_train)

# Set the svc to the best combination of parameters

svc = grid_obj.best_estimator_
```

- ¿Cuáles son los mejores hiperparámetros para el modelo?

SOLUCIÓN

Solución para la combinación de valores indicada en el código propuesto: VC(C=100, gamma=0.001) #kernel rbf

Con los dos modelos entrenados previamente (Random Forest y SVM), realiza una comparativa sobre cuál obtiene mejores resultados de predicción, utiliza la métrica de accuracy para llevar a cabo la comparativa. ¿Cuál obtiene mejor puntuación?

SOLUCIÓN:

```
# Train the model using the training sets

rf.fit(X_train,y_train)

acc_rf = round( metrics.accuracy_score(y_test, y_pred) * 100 , 2 )

print( 'Accuracy of Random Forest model : ', acc_rf )

# Train the model using the training sets

svc.fit(X_train,y_train)

acc_svm = round( metrics.accuracy_score(y_test, y_pred) * 100, 2 )

models = pd.DataFrame({

    'Model': [ 'Random Forest', 'Support Vector Machines'],

    'Score': [ acc_rf, acc_svm]})

models.sort_values(by='Score', ascending=False)

Model    Score

4    Support Vector Machines    98.25

3    Random Forest    93.57
```

11.7. Referencias bibliográficas

Bergstra, J. y Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research, 13*, 281-305.

Bergstra, J., Bengio, Y., Bardenet, R. y Kegel, B. (2011). Algorithms for Hyper-parameter Optimization. *Advances in Neural Information Processing Systems, 24*, 1-9. https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf

Borovitskiy, V., Azangulov, I., Terenin, A., Mostowsky, P., Deisenroth, M. y Durrande, N. (2021). *Matérn Gaussian Processes on Graphs*. Proceedings of Machine Learning Research 130, 2593-2601 https://proceedings.mlr.press/v130/borovitskiy21a.html

Fabien, M. (s.f.). *A Guide to Hyperparameter Optimization (HPO): Parameters and Model Optimization*. https://maelfabien.github.io/machinelearning/Explorium4/#

Williams, C. K. y Rasmussen, C. E. (2006). Gaussian processes for machine learning. En Dietterich, T. (ed.), *Adaptive Computation and Machine Learning,* 2(3). MIT press.

Wu, J., Chen, X. Y., Zhang, H., Xiong, L. D., Lei, H. y Deng, S. H. (2019). Hyperparameter optimization for machine learning models based on Bayesian optimization. *Journal of Electronic Science and Technology, 17*(1), 26-40.

Optimización de hiperparámetros con sci-kit learn

Scikit learn. (s.f.). *3.2. Tuning the hyper-parameters of an estimator*. https://scikit-learn.org/stable/modules/grid_search.html

En la guía de usuario de la librería scikit learn se presenta la implementación de los métodos de optimización existentes en esta librería. En particular, se presentan los algoritmos la búsqueda en rejilla y la búsqueda aleatoria.

Optimizar los hiperparámetros del modelo en Azure

Microsoft Azure. (2023). *Optimizar los hiperparámetros del modelo*. Microsoft Learn. https://learn.microsoft.com/es-es/azure/machine-learning/component-reference/tune-model-hyperparameters?view=azureml-api-2

En esta entrada de la documentación de Azure, se presenta una explicación sobre los hiperparámetros de los modelos de *Machine Learning* junto con la explicación de cómo configurarlo en el entorno cloud de Azure.

AutoML

He, X., Zhao, K. y Chu, X. (2021). AutoML: A Survey of the State-of-the-Art. *Knowledge-Based Systems, 212*, 1-37. https://www.sciencedirect.com/science/article/abs/pii/S0950705120307516

En el presente artículo se presenta una revisión de los principales métodos de AutoML, o Aprendizaje Automático Automatizado. AutoML se refiere a la automatización de tareas relacionadas con el diseño, entrenamiento y evaluación de modelos de aprendizaje automático. El objetivo principal del AutoML es hacer que el proceso de construcción de modelos de aprendizaje automático sea más accesible, eficiente y menos dependiente de la experiencia humana. Se recomienda la lectura del artículo para entender cuáles son las principales técnicas y herramientas.

¿Búsqueda en rejilla o búsqueda aleatoria?

Torino, B. (2020). *GridSearchCV or RandomSearchCV?* Medium. https://towardsdatascience.com/gridsearchcv-or-randomsearchcv-5aa4acf5348c

En esta entrada de Towards Data Science se presenta una comparativa de los dos principales métodos de optimización de hiperparámetros y se exponen las ventajas e inconvenientes de cada uno de ellos, así como algunas pistas para la elección de uno u otro método.

Test

1. Señala la afirmación correcta sobre los hiperparámetros:

   A. Son específicos de cada modelo.

   B. Son generales para todos los modelos.

   C. Se utilizan para mejorar el rendimiento en el conjunto de test.

   D. Los hiperparámetros no afectan el rendimiento del modelo de aprendizaje.

2. Señala la afirmación correcta sobre los hiperparámetros:

   A. Sirven para optimizar los modelos.

   B. Su optimización puede generar sobreajuste a los datos de entrenamiento.

   C. Su optimización nunca genera sobreajuste, pues no modifican los datos de entrenamiento.

   D. Son iguales para todos los modelos.

3. Señala la afirmación verdadera sobre el método cartesian grid search:

   A. Consiste en una búsqueda aleatoria de los parámetros.

   B. Consiste en una búsqueda exhaustiva de los parámetros.

   C. Consiste en una búsqueda estocástica de los parámetros.

   D. Consiste en una búsqueda exhaustiva acotada de los parámetros.

4. La optimización de hiperparámetros:

   A. Permite controlar y resolver el overfitting.

   B. Permite controlar y resolver el underfitting.

   C. La optimización de hiperparámetros no tiene impacto en el rendimiento del modelo.

   D. Ninguna de las anteriores es correcta.

5. Señala la afirmación válida sobre el método de cartesian grid search:

A. No se puede paralelizar.

B. Se puede paralelizar.

C. Se puede calcular de forma distribuida.

D. Requiere entrenamiento del modelo en múltiples secuencias.

6. Señala la afirmación correcta sobre el método de optimización evolutiva:

A. Consiste en ejecutar algoritmos evolutivos para encontrar el mejor conjunto de parámetros.

B. Es una evolución del método cartesiano.

C. Es una evolución del método bayesiano.

D. La optimización evolutiva de hiperparámetros garantiza resultados precisos rápidamente.

7. Señala la afirmación válida. Para la mayoría de los conjuntos de datos:

A. Todos los hiperparámetros son relevantes.

B. Solo algunos de los hiperparámetros son relevantes.

C. La relevancia o no de los hiperparámetros depende del volumen de los datos.

D. Existe el mismo conjunto de hiperparámetros relevante.

8. Señala la afirmación correcta. El número de modelos necesarios para converger a un óptimo global:

A. Por lo general es siempre el mismo.

B. Puede variar bastante y el criterio de parada lo debe considerar.

C. No es algo que debe ser tenido en cuenta.

D. Debe estar predefinido por el ingeniero de machine learning.

9. Los modelos de procesos gaussianos:

A. Son un método de optimización de hiperparámetros basado en búsqueda cartesiana.

B. Se utilizan para optimizar parámetros de modelos complejos.

C. Son un tipo de automatic sequential optimization.

D. Son un método de optimización que ralentiza la búsqueda de hiperparámetros.

10. La búsqueda aleatoria de hiperparámetros:

A. Siempre es suficiente.

B. En algunos modelos complejos puede ser insuficiente.

C. Que sea suficiente o insuficiente depende de los datos de entrenamiento.

D. Es eficiente porque siempre encuentra la configuración óptima de hiperparámetros.