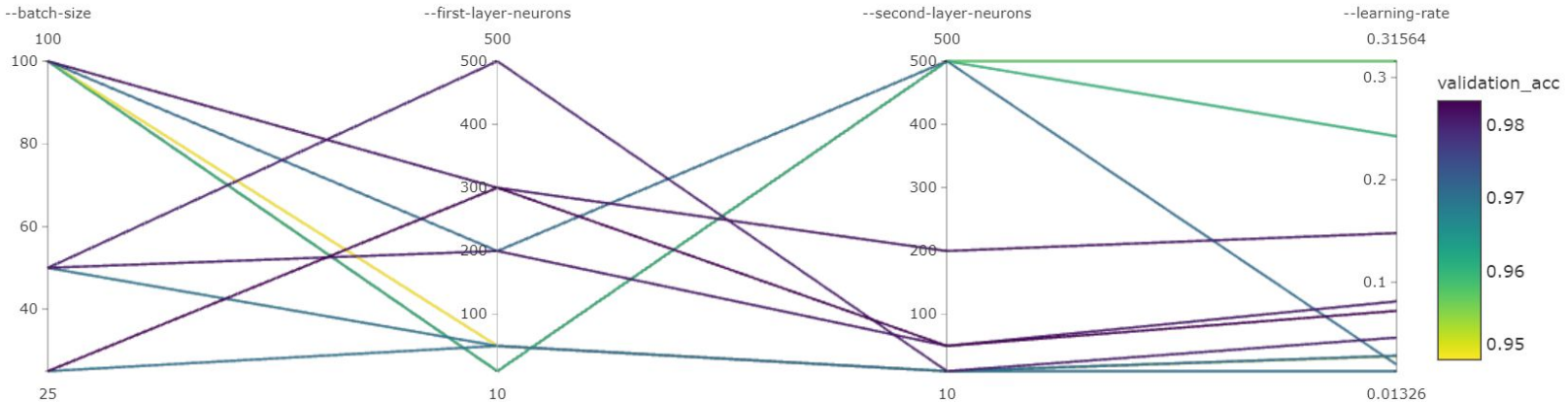
Ajuste de hiperparámetros

Curso de inteligencia artificial-Intermedio

Objetivo

Estudiar una técnica para optimizar nuestro modelo.



Ajuste de hiperparámetros Teoría

Introducción

1. ¿Qué son los hiperparámetros?

En un modelo de machine learning, los hiperparámetros son parámetros que se establecen **antes** del proceso de entrenamiento y no se aprenden directamente de los datos. Son diferentes a los **parámetros** del modelo (como los coeficientes en una regresión lineal) que se ajustan durante el entrenamiento. Los hiperparámetros afectan el rendimiento y la capacidad del modelo de generalizar correctamente.

Ejemplos comunes de hiperparámetros:

- **Tasa de aprendizaje** en algoritmos de optimización.
- **Número de vecinos** en K-NN.
- **Profundidad máxima de un árbol de decisión**.
- **Cantidad de unidades en capas ocultas** en redes neuronales.

Introducción

¿Por qué es importante ajustar los hiperparámetros?

El ajuste adecuado de los hiperparámetros puede mejorar el rendimiento de un modelo significativamente, ya que ayudan a controlar el **sesgo** y la **varianza** del modelo:

- **Subajuste (underfitting)**: Cuando el modelo es demasiado simple, tiene un alto sesgo y bajo rendimiento tanto en los datos de entrenamiento como en los de prueba.
- **Sobreajuste (overfitting)**: Cuando el modelo es demasiado complejo, ajusta demasiado los datos de entrenamiento y tiene un alto error en los datos de prueba debido a la alta varianza.

Métodos comunes de ajuste de hiperparámetros

a) Búsqueda en rejilla (Grid Search)

Este es uno de los métodos más sencillos y populares. Implica definir un conjunto de valores posibles para cada hiperparámetro y probar todas las combinaciones posibles para encontrar la que mejor funcione. El modelo se entrena y evalúa con cada combinación para seleccionar la mejor.

Ventajas:

- Sencillo de implementar.

Desventajas:

- Puede ser muy costoso computacionalmente cuando hay muchos hiperparámetros o si el espacio de búsqueda es grande.

Métodos comunes de ajuste de hiperparámetros

b) Búsqueda aleatoria (Random Search)

En lugar de probar todas las combinaciones posibles, la búsqueda aleatoria selecciona valores de los hiperparámetros de manera aleatoria de un espacio predefinido. Es más eficiente en términos computacionales, ya que puede explorar el espacio de búsqueda de forma más amplia en menos tiempo.

Ventajas:

- Más eficiente que Grid Search en muchos casos.
- Puede descubrir combinaciones efectivas rápidamente.

Desventajas:

- No garantiza encontrar la mejor combinación.

Métodos comunes de ajuste de hiperparámetros

c) Optimización bayesiana

Este es un enfoque más avanzado. En lugar de buscar al azar o exhaustivamente, la optimización bayesiana trata de construir un modelo probabilístico del rendimiento del modelo en función de los hiperparámetros. Usa esta información para decidir qué combinaciones probar a continuación, explorando las que tienen más potencial.

Ventajas:

- Menos evaluaciones del modelo en comparación con Grid y Random Search.
- Más eficiente para hiperparámetros continuos.

Desventajas:

- Más complicado de implementar.

Métodos comunes de ajuste de hiperparámetros

- **d) Algoritmos evolutivos y otros enfoques avanzados**

  Existen métodos aún más avanzados, como el uso de algoritmos evolutivos o técnicas de optimización como el **simulated annealing**. Estos métodos tienden a ser útiles para hiperparámetros complejos y en problemas que requieren una gran exploración del espacio de búsqueda

Ajuste de hiperparámetros y validación cruzada

¿Cómo elegimos la mejor combinación de hiperparámetros?

Se puede hacer evaluando por cada selección nuestro modelo usando alguna métrica, sin embargo,

Ajuste de hiperparámetros y validación cruzada

¿Cómo elegimos la mejor combinación de hiperparámetros?

Se puede hacer evaluando por cada selección nuestro modelo usando alguna métrica, sin embargo, el **método más común para validar la selección de hiperparámetros es la validación cruzada (cross-validation)**.

Ajuste de hiperparámetros y validación cruzada

¿Cómo elegimos la mejor combinación de hiperparámetros?

Se puede hacer evaluando por cada selección nuestro modelo usando alguna métrica, sin embargo, el **método más común para validar la selección de hiperparámetros es la validación cruzada (cross-validation)**.

Una vez se prueban todas las combinaciones (en Grid Search, Random Search) o hasta que la optimización Bayesiana converja, la mejor combinación de hiperparámetros se selecciona en función del rendimiento promedio en la validación cruzada.

Implementación en Python

Nuevas librerías

```
import numpy as np from sklearn.model_selection
import GridSearchCV, RandomizedSearchCV from sklearn.ensemble
import RandomForestClassifier from sklearn.datasets
import load_iris from sklearn.model_selection
import train_test_split from sklearn.metrics import
accuracy_score
```

Importaciones:

- `GridSearchCV` para usar el ajuste de hiperparámetros por búsqueda de rejilla.
- `RandomizedSearchCV` para usar el ajuste de hiperparámetros por búsqueda aleatoria.

Código rejilla de parámetros

```
param_grid = { 'n_estimators': [50, 100, 200], 'max_depth': [None, 10, 20, 30],
'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4] }
```

Este diccionario `param_grid` define los **hiperparámetros** y sus respectivos valores que se van a explorar en la búsqueda:

- **`n_estimators`**: Número de árboles en el bosque aleatorio (50, 100 o 200).
- **`max_depth`**: Profundidad máxima de los árboles (sin límite o hasta una profundidad de 10, 20 o 30).
- **`min_samples_split`**: Número mínimo de muestras necesarias para dividir un nodo.
- **`min_samples_leaf`**: Número mínimo de muestras que debe tener un nodo hoja.

En una **búsqueda en rejilla (Grid Search)**, se prueban todas las combinaciones posibles de estos valores.
En una **búsqueda aleatoria (Random Search)**, se selecciona un subconjunto aleatorio de combinaciones.

Código Grid Search

```
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, verbose=1,
n_jobs=-1)

grid_search.fit(X_train, y_train)
```

Aquí, se realiza una búsqueda exhaustiva sobre todas las combinaciones de los hiperparámetros en `param_grid`.

- **`GridSearchCV`**: Es un método que realiza una búsqueda en rejilla. Para cada combinación de hiperparámetros, entrena un modelo y valida su rendimiento.
- **`cv=5`**: Utiliza validación cruzada con 5 particiones. Esto significa que los datos de entrenamiento se dividen en 5 subconjuntos, se entrena el modelo en 4 de ellos y se valida en el quinto, repitiendo el proceso 5 veces (cada vez usando un subconjunto diferente como validación).
- **`n_jobs=-1`**: Ejecuta las tareas en paralelo utilizando todos los núcleos de CPU disponibles, lo que mejora la eficiencia del proceso.
- **`verbose=1`**: Muestra información detallada sobre el proceso de búsqueda.

Tras realizar la búsqueda, `grid_search.fit` entrena el modelo con las diferentes combinaciones y selecciona los mejores hiperparámetros basados en los resultados de validación cruzada.

Código Random Search

```
random_search = RandomizedSearchCV(estimator=rf, param_distributions=param_grid,
n_iter=10, cv=5, verbose=1, n_jobs=-1, random_state=42)

random_search.fit(X_train, y_train)
```

En lugar de probar todas las combinaciones de hiperparámetros como en la búsqueda en rejilla, **RandomizedSearchCV** selecciona un subconjunto aleatorio de combinaciones.

- **`n_iter=10`**: Aquí se especifica el número de combinaciones aleatorias a probar. En este caso, se probarán 10 combinaciones distintas.
- **`param_distributions`**: Igual que en `GridSearchCV`, define los posibles valores de hiperparámetros.
- **`random_state=42`**: Asegura que los resultados de la búsqueda sean reproducibles (la selección aleatoria de combinaciones será la misma cada vez que se ejecute).

Al igual que en Grid Search, `RandomizedSearchCV` entrena y evalúa las combinaciones seleccionadas mediante validación cruzada.