Ficha Técnica: Ajuste de Hiperparámetros (Tuning)

Jordi Pozo
I.E.S. Ribera de Castilla

Curso 2025/26

1 Concepto: Parámetros vs. Hiperparámetros

En el aprendizaje automático, es crucial distinguir entre dos tipos de variables que definen el comportamiento del modelo:

- **Parámetros del Modelo:** Son aprendidos automáticamente por el algoritmo durante el entrenamiento a partir de los datos. El programador no los fija manualmente.
  - Ejemplos: Los pesos (w) en una red neuronal, los coeficientes en una regresión lineal.
- **Hiperparámetros:** Son configuraciones externas al modelo que el programador debe definir **antes** del entrenamiento. Controlan la complejidad y la forma en que el algoritmo aprende.
  - Ejemplos: La profundidad máxima (`max_depth`) en un árbol, la `C` en SVM, el número de vecinos (`k`) en KNN.

El **Ajuste de Hiperparámetros (Hyperparameter Tuning)** es el proceso de buscar la combinación óptima de estos valores para maximizar el rendimiento del modelo (generalmente medido por precisión, F1-score o error cuadrático).

2 Estrategia de Validación: Cross-Validation

Cuando ajustamos hiperparámetros, corremos un gran riesgo de sobreajuste si solo usamos un conjunto de "Train" y "Test". Podríamos acabar eligiendo hiperparámetros que funcionan muy bien en el Test por pura suerte.

Para evitarlo, utilizamos la **Validación Cruzada (k-Fold Cross-Validation)**: 1. Dividimos los datos de entrenamiento en k partes (folds). 2. Entrenamos k veces, usando cada vez $k-1$ partes para entrenar y 1 parte para validar. 3. El rendimiento final es el promedio de las k iteraciones.

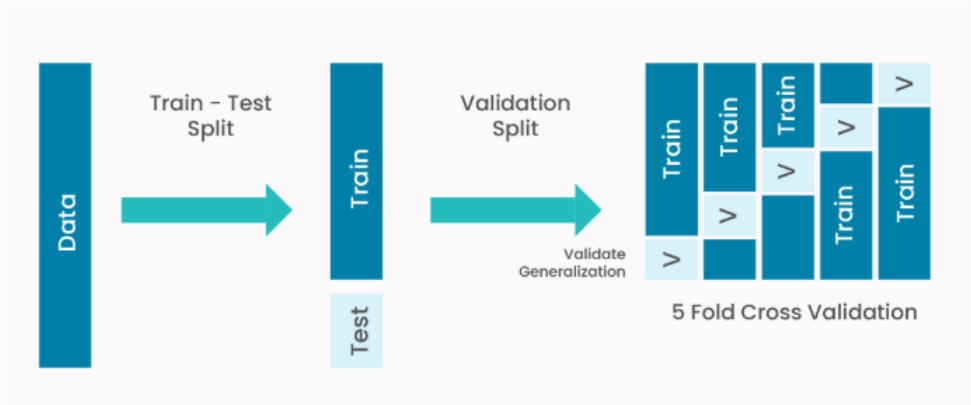

Figure 1: Esquema validación cruzada (Cross Validation).

3 Métodos de Búsqueda

3.1 1. Búsqueda en Rejilla (Grid Search)

Es el método de fuerza bruta.

- **Funcionamiento:** Se define una lista de valores para cada hiperparámetro. El algoritmo prueba **todas** las combinaciones posibles.
- **Ventaja:** Garantiza encontrar la mejor combinación dentro de la rejilla definida.
- **Desventaja:** Computacionalmente muy costoso si hay muchos parámetros.

3.2 2. Búsqueda Aleatoria (Random Search)

- **Funcionamiento:** En lugar de probar todas las combinaciones, selecciona combinaciones **aleatorias** un número fijo de veces (definido por `n_iter`).
- **Ventaja:** Mucho más rápido y eficiente. A menudo encuentra una solución casi tan buena como Grid Search en una fracción del tiempo.
- **Desventaja:** No garantiza encontrar el óptimo global exacto.

4 Implementación en `scikit-learn`

La librería ofrece dos clases principales en el módulo `model_selection`:

- `GridSearchCV`: Para búsqueda en rejilla con validación cruzada.
- `RandomizedSearchCV`: Para búsqueda aleatoria con validación cruzada.

Principales argumentos:

- `estimator`: El modelo a ajustar (ej. `SVC()`).
- `param_grid` (en Grid): Diccionario con las listas de valores a probar.
- `param_distributions` (en Random): Diccionario con valores o distribuciones estadísticas.
- `cv`: Número de "folds" para la validación cruzada (ej. 5).
- `scoring`: Métrica a optimizar (ej. 'accuracy', 'f1', 'neg_mean_squared_error').
- `n_jobs`: Número de núcleos de CPU a usar (-1 usa todos).

5 Ejemplos Prácticos

5.1 Ejemplo 1: GridSearchCV con SVM

Buscamos el mejor kernel, C y gamma para clasificar el dataset Iris.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC

# 1. Cargar datos
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(
iris.data, iris.target, test_size=0.3, random_state=42
)

# 2. Definir el modelo base
svm = SVC()

# 3. Definir la rejilla de parametros (param_grid)
param_grid = {
  'C': [0.1, 1, 10, 100],
  'gamma': [1, 0.1, 0.01, 0.001],
  'kernel': ['rbf', 'linear']
}

# 4. Configurar GridSearchCV
# cv=5 significa validacion cruzada de 5 pliegues
grid = GridSearchCV(estimator=svm,
param_grid=param_grid,
refit=True, # Re-entrena el mejor modelo al final
verbose=2,
cv=5)

# 5. Ejecutar la busqueda
grid.fit(X_train, y_train)

# 6. Resultados
print(f"Mejores parametros: {grid.best_params_}")
print(f"Mejor score (CV): {grid.best_score_:.4f}")

# 7. Usar el mejor modelo para predecir
# Gracias a refit=True, 'grid' se comporta como el mejor modelo
predicciones = grid.predict(X_test)

```

Listing 1: Uso de GridSearchCV para optimizar un SVM.

5.2 Ejemplo 2: RandomizedSearchCV con Random Forest

Buscamos la profundidad y el número de árboles. Usamos Random Search porque el espacio de búsqueda puede ser muy grande.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# 1. Modelo base
rf = RandomForestClassifier(random_state=42)

# 2. Distribucion de parametros
# Usamos rangos o listas mas amplias
param_dist = {
```

```
    'n_estimators': [50, 100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# 3. Configurar RandomizedSearchCV
# n_iter=20 probara solo 20 combinaciones aleatorias
random_search = RandomizedSearchCV(estimator=rf,
param_distributions=param_dist,
n_iter=20,
cv=3,
verbose=1,
random_state=42,
n_jobs=-1)

# 4. Ajustar
random_search.fit(X_train, y_train)

# 5. Resultados
print(f"Mejores parametros: {random_search.best_params_}")

```

Listing 2: Uso de RandomizedSearchCV.