Machine learning y deep learning

Rolando Gonzales Martinez, PhD

Fellow postdoctoral Marie Skłodowska-Curie

Universidad de Groningen (Países Bajos)

Investigador (researcher)

Iniciativa de Pobreza y Desarrollo Humano de la Universidad de Oxford (UK)

Contenido del curso

(4) Evaluación y Mejora de Modelos

- Métodos de validación cruzada.
- Técnicas de ajuste de hiper parámetros (Grid Search, Random Search).
- Laboratorio: validación y optimización de modelos de machine learning.

Laboratorios

- **AIMLDL_0401.ipynb:** Curva ROC y validación de los modelos ML para predicciones de fraudes de tarjetas de crédito.
- **AIMLDL_0402.ipynb:** Validación cruzada k-fold y validación cruzada k-fold estratificada de los modelos ML para datos desbalanceados
- **AIMLDL_0403.ipynb:** aprendizaje con datos desbalanceados: algoritmos de sobremuestreo, submuestreo, SMOTE y ADASYN
- **AIMLDL_0404.ipynb:** Validación cruzada e Hypertuning con algoritmos GridSearchCV, RandomizedSearchCV, BayesSearchCV: Clasificadores supervisados basados en Máquinas de Soporte Vectorial
- **AIMLDL_0401.R:** Hypertuning con optimización Bayesiana para clasificadores no supervisados basados en DBSCAN
- **AIMLDL_0405.ipynb:** Ensamblaje de modelos y algoritmos ML

Machine learning

Dado un espacio de entrada X (espacio de características) y un espacio de salida Y, machine learning es un problema de optimización en el que el objetivo es encontrar una función f: $X \rightarrow Y$ que predice la salida $y \in Y$ dada una entrada $x \in X$, siendo f^* óptima en términos de **generalización y regularización**:

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \qquad x_i \in \mathcal{X} \qquad y_i \in \mathcal{Y}$$

$$f^* = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim P} \left[L(f(x), y) \right]$$

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i)$$

$$f^* = \arg\min_{f \in \mathcal{F}} \left[\frac{1}{n} \sum_{i=1}^{n} L(f(x_i), y_i) + \lambda\, R(f) \right]$$

Ensamblaje y combinación de modelos

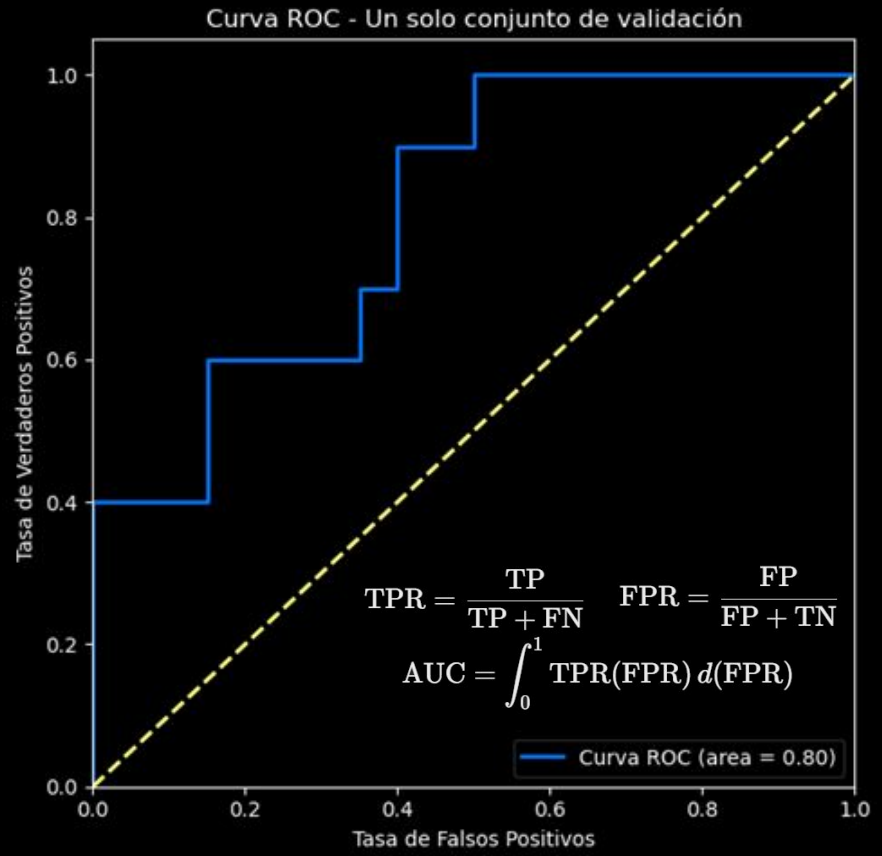
Los métodos de ensamblaje y combinación de modelos ML se utilizan para mejorar la precisión y robustez de las predicciones combinando varios modelos:

Cascading

- Descripción: En cascading, los modelos se aplican en una secuencia donde el resultado de un modelo se usa como entrada para el siguiente. Este enfoque es útil cuando cada modelo puede aportar un aspecto único de la predicción.
- Ejemplo: Un ejemplo podría ser una cadena de modelos donde el primer modelo realiza una detección de objetos, y el segundo modelo clasifica los objetos detectados.

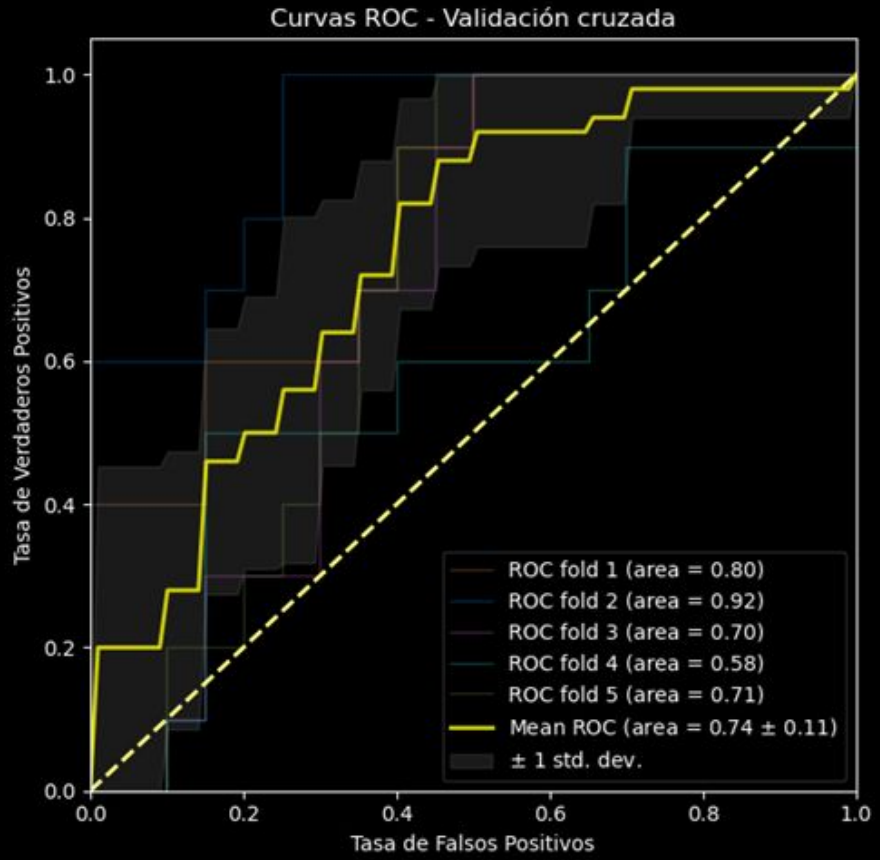
Curva ROC (Receiver Operating Characteristic) y validación de modelos ML

- ROC: representación gráfica de la capacidad de un modelo de clasificación para discriminar entre clases.
- Muestra la relación entre la Tasa de Verdaderos Positivos (TPR) y la Tasa de Falsos Positivos (FPR) para diferentes umbrales de corte de la probabilidad predicha:
  - Eje X (tasa de falsos positivos, FPR): Proporción de negativos que son incorrectamente clasificados como positivos.
  - Eje Y (tasa de verdaderos positivos, TPR o Sensibilidad): Proporción de positivos correctamente identificados por el modelo.



Curva ROC (Receiver Operating Characteristic) y validación de modelos ML

- El rendimiento de un modelo ML depende de la partición del conjunto de datos utilizado para el entrenamiento.
- Técnicas de validación cruzada y hypertuning permiten evaluar y controlar la estabilidad y generalización del modelo para dar una estimación robusta de la capacidad discriminatoria/predictiva de los modelos ML



Validación cruzada k-fold (k-fold cross-validation)

- Se particiona la muestra en k partes (folds)
- Valores convencionales: k = 5, k = 10, k =20. Depende del tamaño de los datos y el costo computacional. Normalmente k =10
- En muestras pequeñas, k = n (partición LOO)

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

$$\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$$

$\mathcal{D}_i \subset \mathcal{D}$ para todo i y $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset$ para $i \neq j$.

Para cada $i = 1, 2, \ldots, k$:

$$\mathcal{D}_{-i} = \mathcal{D} \setminus \mathcal{D}_i = \bigcup_{j \neq i} \mathcal{D}_j.$$

$h(\cdot; \theta_i)$ usando $\mathcal{D}_{-i}$

$$E_i = \frac{1}{|\mathcal{D}_i|} \sum_{(x_j, y_j) \in \mathcal{D}_i} \ell(h(x_j; \theta_i), y_j)$$

$$E_{\text{CV}} = \frac{1}{k} \sum_{i=1}^{k} E_i$$

Validación cruzada k-fold estratificada

- Extiende la validación cruzada k-fold para que cada pliegue (fold) tenga aproximadamente la misma proporción de clases c = 1,2,...,C del target que el conjunto de datos completo.
- Implica un proceso de estratificación (cada categoria c es un estrato)
- Útil en datos con un target desbalanceado

$$D = \{(x_i, y_i)\}_{i=1}^{n} \qquad c \in C$$

$$n_c = \sum_{i=1}^{n} \mathbb{I}(y_i = c) \qquad p(c) = \frac{n_c}{n}$$

$$S_i = \varnothing, \quad T_i = \varnothing \quad \text{para } i = 1, \ldots, k$$

$$D_c = \{(x_i, y_i) \mid y_i = c, i = 1, \ldots, n\}$$

$$\tilde{D}_c = \{\tilde{D}_{c,1}, \tilde{D}_{c,2}, \ldots, \tilde{D}_{c,k}\}$$

$$T_i = T_i \cup \tilde{D}_{c,i}$$

$$S_i = S_i \cup \left(\bigcup_{j \neq i} \tilde{D}_{c,j} \right)$$

$$\{(S_i, T_i)\}_{i=1}^{k}$$

Aprendizaje desbalanceado: imbalanced learning

- Los algoritmos/modelos ML tradicionales optimizan la precisión, lo que puede llevar a que favorezcan la predicción de la clase mayoritaria en detrimento de la minoritaria en situaciones de desequilibrio.
- EL aprendizaje desbalanceado se aplica cuando una clase (la clase mayoritaria) tiene muchos más ejemplos o instancias que otra clase (la clase minoritaria).

Opciones:

- Sobremuestreo y submuestreo aleatorio simple
- SMOTE (Synthetic Minority Over-sampling Technique): es una técnica de sobremuestreo de minorías sintéticas que genera nuevos datos similares pero no iguales a los originales, mediante interpolacion, en base a k-vecinos cercanos
- ADASYN (Adaptive Synthetic Sampling) es una técnica adaptativa de sobremuestreo de minorías sintéticas similar a SMOTE pero considerando puntos con dificultad de clasificación

GridSearchCV: Validación cruzada con búsqueda de cuadrícula

- La validación cruzada con búsqueda de cuadrícula (GridSearchCV) se utiliza para optimizar los hiperparámetros de un modelo de machine learning
- GridSearchCV realiza una búsqueda exhaustiva a través de un conjunto especificado de valores de hiperparámetros para encontrar la mejor combinación que maximice la capacidad predictiva del modelo

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Validación cruzada con búsqueda de cuadrícula

- S es el conjunto de valores de los hiperparámetros (el conjunto de todas las posibles combinaciones de los hiperparámetros que se quiere evaluar)
- S representa la cuadrícula donde cada dimensión corresponde a un hiperparámetro diferente y se obtiene como el producto cartesiano de los S-conjuntos de los *m*-hiperparametros que se quiere evaluar.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Validación cruzada y hypertuning con búsqueda de cuadrícula

- La validación cruzada divide el conjunto de datos en *k* subconjuntos (folds). Luego, se entrena el modelo en *k-1* de estos subconjuntos y se valida en el subconjunto restante.
- Este proceso se repite *k* veces, cambiando el subconjunto de validación cada vez, para asegurar que cada observación se use tanto para entrenamiento como para validación.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Validación cruzada y hypertuning con búsqueda de cuadrícula

- Se estima el modelo M en cada partición de entrenamiento de la muestra y se realizan predicciones en la muestra test
- Se obtienen métricas de desempeño (scores) del modelo en las muestras test y se promedian para obtener el score de validación cruzada promedio en los *k*-folds.
- El algoritmo devuelve los mejores hiperparámetros y el modelo óptimo entrenado en todo el conjunto de datos D.

$$D = \{(x_i, y_i)\}_{i=1}^{n}$$

$$M(\theta) \qquad \theta = (\theta_1, \theta_2, \ldots, \theta_m)$$

$$\mathcal{L}(M, D)$$

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \cdots \times \mathcal{S}_m$$

$$\mathcal{S} = \{\theta_1^{(1)}, \theta_1^{(2)}, \ldots\} \times \{\theta_2^{(1)}, \theta_2^{(2)}, \ldots\} \times \cdots \times \{\theta_m^{(1)}, \theta_m^{(2)}, \ldots\}$$

$$D_1, D_2, \ldots, D_k$$

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \in \mathcal{S}$$

$$\text{CV\_score}(\theta) = 0$$

$$D_{\text{train}}^{(i)} = D \setminus D_i \qquad D_{\text{val}}^{(i)} = D_i$$

$$\text{score}_i(\theta) = \mathcal{L}(M(\theta), D_{\text{val}}^{(i)})$$

$$\text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k}\text{score}_i(\theta)$$

$$\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta)$$

GridSearchCV: Validación cruzada y hypertuning con búsqueda de cuadrícula

El algoritmo devuelve los mejores hiperparámetros y el modelo óptimo entrenado en todo el conjunto de datos D.

$$
\begin{aligned}
&\text{Inicializar: } \mathcal{S},\ k,\ D = \{D_1, \ldots, D_k\}, \\
&\qquad \text{CV\_score}(\theta) = 0 \text{ para cada } \theta \in \mathcal{S} \\
&\text{Para cada combinación de hiperparámetros } \theta \in \mathcal{S}: \\
&\quad \text{Para cada fold } i = 1, \ldots, k: \\
&\qquad M(\theta) \text{ se entrena en } D^{(i)}_{\text{train}} = D \setminus D_i \\
&\qquad \text{score}_i(\theta) = \mathcal{L}(M(\theta), D^{(i)}_{\text{val}}) \\
&\qquad \text{CV\_score}(\theta) \mathrel{+}= \frac{1}{k} \text{score}_i(\theta) \\
&\theta^* = \arg\min_{\theta \in \mathcal{S}} \text{CV\_score}(\theta) \\
&\text{Retornar } \theta^* \text{ y el modelo entrenado } M(\theta^*)
\end{aligned}
$$

RandomizedSearchCV: Validación cruzada y hypertuning con búsqueda no exhaustiva

Dado un espacio cartesiano *H* formado por los valores de los *n*-hiperparametros *h*:

- Se selecciona aleatoriamente j = 1,2,...,m-muestras (m-iteraciones) de valores de los hiperparametros y se forman tuplas
- Se estiman modelos ML con las tuplas en una muestra train en particiones k (cv)
- Se calculan j-métricas de desempeño ℓ en la muestra de validación
- Se seleccionan los valores de h que maximizan (o minimizan) ℓ
- Se ajusta el modelo con la muestra completa (refit)

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{m}$$

$$\mathcal{H} = \mathcal{H}_1 \times \mathcal{H}_2 \times \cdots \times \mathcal{H}_n$$

$$h^{(j)} = (h_1^{(j)}, h_2^{(j)}, \ldots, h_n^{(j)})$$

$$M(h^{(j)}) \qquad \mathcal{D}_{\text{train}} \subseteq \mathcal{D}$$

$$\ell^{(j)} = \mathcal{L}(M(h^{(j)}), \mathcal{D}_{\text{val}})$$

$$h^* = \arg \max_{j \in \{1,2,\ldots,k\}} \ell^{(j)}$$

$$M(h^*)$$

BayesSearchCV: Validación cruzada y hypertuning con procesos Gaussianos

Dado un espacio *H* **discreto o continuo** formado por los valores de los *n*-hiperparametros *h*:

- Se calcula una función (surrogada) **probabilística** para aproximar la función de desempeño objetivo
- Se escoge valores de h que optimizan la función surrogada e iterativamente maximizan la función de adquisición $\alpha(\mathbf{h})$
- La función de adquisición equilibra:
  - La exploración de regiones del espacio de hiperparámetros.
  - La explotación (concentracion) en regiones con los mejores hiperparámetros (donde la diferencia entre surrogados es positiva)

$$f(\mathbf{h}) \qquad \{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{n_0}\}$$

$$\mathbf{h} = (h_1, h_2, \ldots, h_n)$$

$$\mathbf{h}^* = \arg\min_{\mathbf{h} \in \mathcal{H}} f(\mathbf{h})$$

$$f(\mathbf{h}^*)$$

$$\alpha(\mathbf{h}) = \mathbb{E}\left[\max(0, f(\mathbf{h}) - f(\mathbf{h}_{\text{best}}))\right]$$

$$\mathcal{GP}(\mu(\mathbf{h}), k(\mathbf{h}, \mathbf{h}'))$$

$$\mu(\mathbf{h}) = \mathbb{E}[f(\mathbf{h})]$$

$$k(\mathbf{h}, \mathbf{h}') = \text{Cov}(f(\mathbf{h}), f(\mathbf{h}'))$$

$$f(\mathbf{h}) \sim \mathcal{N}(\mu(\mathbf{h}), \sigma^2(\mathbf{h}))$$

DBSCAN y métrica de desempeño de silueta

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de clustering que agrupa puntos en alta densidad y puede identificar puntos aislados como ruido.

No requiere especificar el número de clusters pero es sensible a hiperparámetros.

Hiperparámetros:

- Epsilon: distancia máxima entre dos puntos para que se considere en el vecindario del otro
- min_samples: número mínimo de muestras en un vecindario para que un punto sea considerado como el punto central de un cluster

$$D = \{x_1, x_2, \ldots, x_n\}$$

$\epsilon > 0$: radio máximo de la vecindad

$\text{min\_samples} \in \mathbb{Z}^+$

$$N_\epsilon(p) = \{q \in D \mid \text{dist}(p, q) \leq \epsilon\}$$

Si $|N_\epsilon(p)| < \text{min\_samples}$, marca p como ruido

$$|N_\epsilon(p)| \geq \text{min\_samples}$$

$$S = \frac{1}{n}\sum_{i=1}^{n} s(i) \qquad s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$a(i) = \frac{1}{|C(i)| - 1} \sum_{j \in C(i), j \neq i} \text{dist}(i, j)$$

$$b(i) = \min_{k \neq C(i)} \frac{1}{|C_k|} \sum_{j \in C_k} \text{dist}(i, j)$$

DBSCAN y métrica de desempeño de silueta

El desempeño de algoritmos no supervisados como DBSCAN puede evaluarse con métricas de silueta

- El score de silueta global (S) es el promedio de los scores individuales para cada punto s(i), y $-1 \leq S \leq 1$
- Los scores individuales s(i), $-1 \leq s(i) \leq 1$, dependen de la cohesión a(i) y la separación b(i)
  - a(i): distancia media entre el punto i y todos los demás puntos en el mismo clúster (cercanía entre puntos en el mismo cluster)
  - b(i): distancia media entre el punto i y todos los demás puntos en el clúster más cercano (separación entre clusters)

$$D = \{x_1, x_2, \ldots, x_n\}$$

$\epsilon > 0$: radio máximo de la vecindad

$\text{min\_samples} \in \mathbb{Z}^+$

$$N_\epsilon(p) = \{q \in D \mid \text{dist}(p, q) \leq \epsilon\}$$

Si $|N_\epsilon(p)| < \text{min\_samples}$, marca p como ruido

$$|N_\epsilon(p)| \geq \text{min\_samples}$$

$$S = \frac{1}{n}\sum_{i=1}^{n} s(i) \qquad s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

$$a(i) = \frac{1}{|C(i)| - 1} \sum_{j \in C(i), j \neq i} \text{dist}(i, j)$$

$$b(i) = \min_{k \neq C(i)} \frac{1}{|C_k|} \sum_{j \in C_k} \text{dist}(i, j)$$

Ensamblaje y combinación de modelos

Los métodos de ensamblaje y combinación de modelos ML se utilizan para mejorar la precisión y robustez de las predicciones combinando varios modelos:

Bagging (Bootstrap Aggregating)

- Implica entrenar múltiples modelos sobre diferentes subconjuntos de los datos de entrenamiento generados mediante remuestreo con reemplazamiento.
- Cada modelo es entrenado de manera independiente y sus predicciones se combinan (por lo general mediante un promedio en el caso de regresión o votación en el caso de clasificación).
- Ejemplos: Bosques aleatorios

Ensamblaje y combinación de modelos

Los métodos de ensamblaje y combinación de modelos ML se utilizan para mejorar la precisión y robustez de las predicciones combinando varios modelos:

Boosting

- En boosting se entrena modelos secuencialmente, y cada nuevo modelo intenta corregir los errores cometidos por los modelos anteriores. Los modelos se combinan de manera ponderada, dando más peso a aquellos que son más precisos.
- Ejemplo: AdaBoost, Gradient Boosting Machines (GBM), y XGBoost

Ensamblaje y combinación de modelos

Los métodos de ensamblaje y combinación de modelos ML se utilizan para mejorar la precisión y robustez de las predicciones combinando varios modelos:

Stacking y blending

- Stacking combina predicciones de múltiples modelos (denominados "modelos base") mediante un modelo "meta". Las predicciones de los modelos base se utilizan como entradas para el modelo meta, que aprende a optimizar la combinación de estas predicciones.
- Blending es una variante de stacking donde en el que las predicciones de los modelos base en un subconjunto de validación se utilizan para entrenar el modelo meta.
- Ejemplo: Usar modelos como regresión logística, árboles de decisión, y redes neuronales como modelos base, y una regresión lineal como el modelo meta.