Aprendizaje Automático - Parte I

Benjamín Palacios S.

28 de septiembre de 2024

1 Clase 1: Introducción al Aprendizaje Automático

1.1 Tipos de Aprendizaje

1.1.1 Aprendizaje Supervisado

- **Clasificación:** Predice etiquetas de clase.
- **Regresión:** Predice un valor continuo.

1.1.2 Aprendizaje No Supervisado

Analiza la estructura de los datos sin usar etiquetas, como en *clustering*.

1.1.3 Aprendizaje Semi-Supervisado

Combina ejemplos etiquetados y no etiquetados para la construcción del modelo.

1.1.4 Aprendizaje Reforzado

Un agente aprende a mejorar su rendimiento interactuando con un entorno.

1.2 Componentes del Aprendizaje Automático

- **Representación:** Árboles de decisión, redes neuronales, entre otros.
- **Evaluación:** Medida de la función de pérdida del modelo.
- **Optimización:** Algoritmos como gradiente descendente para mejorar el rendimiento del modelo.

1.3 Conceptos Clave

- **Generalización:** Capacidad del modelo de rendir bien en datos no vistos.
- **Overfitting (sobreajuste):** Ocurre cuando el modelo es demasiado complejo y ajusta el ruido de los datos.
- **Underfitting (subajuste):** Ocurre cuando el modelo es demasiado simple para capturar patrones en los datos.

1.4 Selección de Modelo

- Separación de datos en conjuntos de entrenamiento, validación y prueba.

1.5 Ingeniería de Características

- **Selección y Transformación:** Selección y transformación de características para mejorar el rendimiento del modelo, e.g., escalado, reducción de dimensionalidad.
- **Deep Learning:** Utiliza representaciones automáticas de los datos (embeddings).

1.6 Maldición de la Dimensionalidad

- A medida que aumentan las dimensiones, los datos se dispersan en el espacio, lo que afecta a los algoritmos basados en distancias (e.g., kNN, SVM).

1.7 Más Datos vs Algoritmo Inteligente

- Un mayor conjunto de datos puede reducir el overfitting y mitigar la maldición de la dimensionalidad.
- Los modelos no paramétricos como SVM o kNN aumentan su número de parámetros con más datos.
- Los modelos paramétricos como las redes neuronales también se benefician de más datos.

1.8 Preprocesamiento

- Técnicas como el escalado de características, codificación de datos, selección de características y reducción de dimensionalidad.

1.9 Evaluación y Predicción

- Comparar algoritmos y configuraciones de hiperparámetros en datos de prueba.
- El rendimiento del modelo se mide en un conjunto de prueba independiente.

2 Clase 2: k-Vecinos Más Cercanos (kNN)

2.1 Introducción

El algoritmo **k-Vecinos Más Cercanos (kNN)** es un método de aprendizaje supervisado que puede usarse tanto para clasificación como para regresión. Se basa en encontrar los k puntos más cercanos en el conjunto de entrenamiento al punto que se desea predecir.

2.2 Construcción del Modelo

El modelo kNN es un ejemplo de **aprendizaje basado en instancias** porque no construye un modelo explícito, sino que almacena todas las instancias del conjunto de entrenamiento.

2.3 Pseudocódigo de kNN

1. **Entrenamiento:** Almacenar el conjunto de entrenamiento completo.
2. **Predicción para un nuevo punto x:**
   (a) Calcular la distancia entre $\mathbf{x}$ y cada punto en el conjunto de entrenamiento.
   (b) Seleccionar los k vecinos más cercanos al punto $\mathbf{x}$.
   (c) **Clasificación:** Realizar una votación mayoritaria entre las clases de los vecinos.
   (d) **Regresión:** Calcular el promedio de los valores de los vecinos.

2.4 Función de Distancia

La distancia más comúnmente utilizada es la **distancia euclidiana**:

$$d(\mathbf{x_i}, \mathbf{x_j}) = \sqrt{\sum_{n=1}^{N} (x_{i,n} - x_{j,n})^2}$$

2.5 Ejemplo Práctico de Clasificación con kNN

Supongamos que tenemos el siguiente conjunto de datos:

Punto	(x_1, x_2)	**Clase**
A	(1, 2)	Rojo
B	(2, 3)	Rojo
C	(3, 3)	Azul
D	(5, 5)	Azul
E	(1, 5)	Rojo

Table 1: Conjunto de entrenamiento

Queremos predecir la clase del punto $\mathbf{x} = (3, 2)$ utilizando $k = 3$.

1. **Calcular distancias:**
   - $d(\mathbf{x}, A) = \sqrt{(3-1)^2 + (2-2)^2} = 2$
   - $d(\mathbf{x}, B) = \sqrt{(3-2)^2 + (2-3)^2} = \sqrt{2} \approx 1.41$
   - $d(\mathbf{x}, C) = \sqrt{(3-3)^2 + (2-3)^2} = 1$
   - $d(\mathbf{x}, D) = \sqrt{(3-5)^2 + (2-5)^2} = \sqrt{13} \approx 3.61$
   - $d(\mathbf{x}, E) = \sqrt{(3-1)^2 + (2-5)^2} = \sqrt{13} \approx 3.61$

2. **Seleccionar los 3 vecinos más cercanos:** C, B, A
3. **Votación:**
   - Clase Rojo: A, B
   - Clase Azul: C
4. **Predicción:** La clase predicha es **Rojo**.

2.6 Visualización del Ejemplo

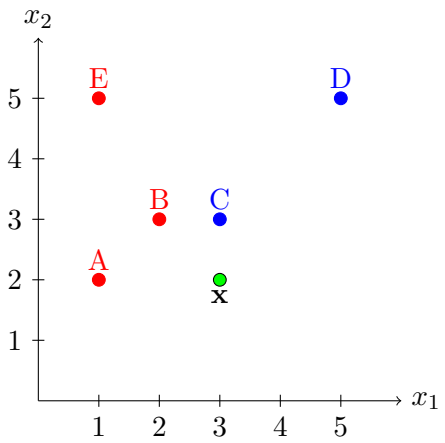

Figure 1: Visualización del ejemplo de kNN

2.7 Impacto del Valor de k

- **Valor pequeño de k:** Modelo más complejo, puede sobreajustarse al ruido.
- **Valor grande de k:** Modelo más simple, puede no capturar patrones importantes (subajuste).

2.8 Ventajas y Desventajas

Ventajas:

- Sencillo de implementar y entender.
- No hace suposiciones sobre la distribución de los datos.

Desventajas:

- Costoso computacionalmente durante la predicción.
- Sensible a la **maldición de la dimensionalidad**.

2.9 Consideraciones Prácticas

- **Normalización de Datos:** Es importante escalar las características, ya que las distancias pueden verse dominadas por características con mayor rango.
- **Selección de Distancia:** Además de la distancia euclidiana, pueden utilizarse otras métricas como la distancia Manhattan.

3 Clase 3: Árboles de Decisión y Métodos de Ensamble

3.1 Introducción

Los **árboles de decisión** son modelos de aprendizaje supervisado que pueden usarse tanto para clasificación como para regresión. Dividen el espacio de características en regiones homogéneas al realizar particiones sucesivas basadas en ciertas condiciones sobre las variables predictoras.

3.2 Árboles de Decisión

3.2.1 Estructura de un Árbol de Decisión

Un árbol de decisión consta de los siguientes elementos:

- **Nodo Raíz:** Es el nodo superior del árbol, donde se inicia la partición.
- **Nodos Internos:** Representan decisiones basadas en una característica.
- **Hojas (Nodos Terminales):** Representan una predicción o resultado final.

3.2.2 Construcción del Árbol

El proceso de construcción de un árbol de decisión implica:

1. Seleccionar la mejor característica para dividir los datos en cada nodo, utilizando un criterio de partición.
2. Repetir recursivamente el proceso para cada subárbol hasta cumplir una condición de parada (por ejemplo, profundidad máxima o número mínimo de muestras en una hoja).

3.2.3 Criterios de Partición

Los criterios más comunes para seleccionar la mejor división en clasificación son:

Entropía e Información La **entropía** mide la impureza de una partición:

$$Entropía(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

donde p_i es la proporción de muestras de la clase i en el conjunto S, y c es el número de clases.

La **ganancia de información** es la reducción en entropía al realizar una partición:

$$Ganancia(S, A) = Entropía(S) - \sum_{v \in Valores(A)} \frac{|S_v|}{|S|} Entropía(S_v)$$

donde A es la característica considerada, S_v es el subconjunto de S donde A toma el valor v.

Índice de Gini El **índice de Gini** es otra medida de impureza:

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2$$

Una división se elige para minimizar el índice de Gini.

3.2.4 Árboles de Regresión

En problemas de regresión, los criterios de partición suelen basarse en minimizar el **Error Cuadrático Medio (MSE)** en las particiones:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

donde $\bar{y}$ es el valor medio de la variable objetivo en la partición.

3.3 Ejemplo Práctico de Construcción de un Árbol de Decisión

Consideremos un conjunto de datos donde queremos predecir si un cliente comprará un producto basado en dos características: *Edad* y *Ingresos*. Utilizamos el índice de Gini para seleccionar las particiones.

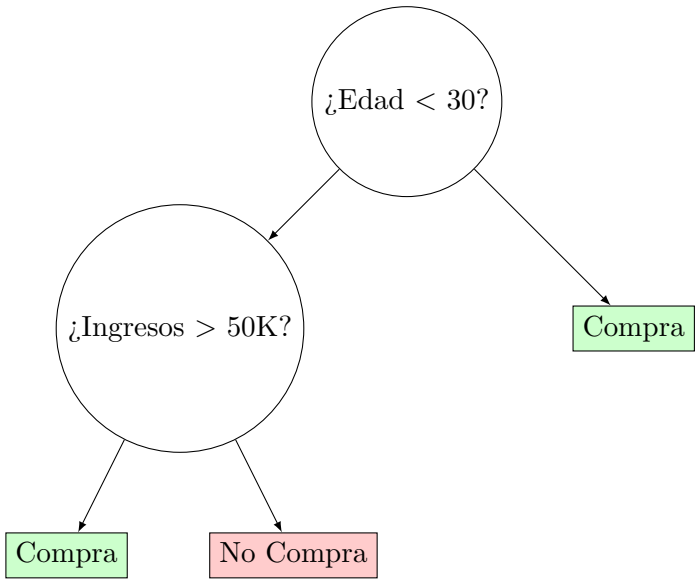

Figure 2: Ejemplo de árbol de decisión para predecir compras

3.4 Sobreajuste y Regularización

Los árboles de decisión tienden a sobreajustarse si se les permite crecer sin restricciones. Para evitarlo, se utilizan técnicas de regularización:

- **Poda del Árbol:** Remover ramas que aportan poco poder predictivo.
- **Limitación de la Profundidad:** Establecer una profundidad máxima para el árbol.
- **Número Mínimo de Muestras en una Hoja:** Evitar hojas con muy pocas muestras.

3.5 Métodos de Ensamble

Los **métodos de ensamble** combinan múltiples modelos base para mejorar el rendimiento general. Se basan en el principio de que una combinación de modelos puede producir resultados más robustos y precisos.

3.5.1 Bagging (Bootstrap Aggregating)

Bagging reduce la varianza al promediar múltiples modelos entrenados en subconjuntos de datos obtenidos mediante **bootstrap** (muestreo con reemplazo).

Algoritmo de Bagging

1. Para $m = 1$ hasta M (número de modelos base):
   (a) Generar un conjunto de entrenamiento D_m mediante bootstrap de los datos originales.
   (b) Entrenar un modelo base h_m en D_m.
2. Para predicciones:
   - **Clasificación:** Votación mayoritaria entre los modelos h_m.
   - **Regresión:** Promedio de las predicciones de los modelos h_m.

3.5.2 Random Forest

El **Random Forest** es una extensión del bagging que introduce aleatoriedad adicional al seleccionar aleatoriamente un subconjunto de características en cada división.

Características Clave

- Cada árbol se entrena con una muestra bootstrap del conjunto de datos.
- En cada nodo, se consideran solo un número aleatorio de características para la división.

3.5.3 Boosting

El **Boosting** construye modelos base secuencialmente, donde cada modelo intenta corregir los errores de los anteriores.

AdaBoost (Adaptive Boosting)

1. Inicializar los pesos de las muestras uniformemente.
2. Para $m = 1$ hasta M:
   (a) Entrenar un modelo base h_m usando los pesos actuales.
   (b) Calcular el error ponderado ϵ_m del modelo.
   (c) Calcular el peso del modelo $\alpha_m = \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$.
   (d) Actualizar los pesos de las muestras:

$$w_i \leftarrow w_i \times \exp\left(\alpha_m \times I(y_i \neq h_m(x_i))\right)$$

   donde I es la función indicadora.
   (e) Normalizar los pesos para que sumen 1.
3. La predicción final es:

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right)$$

Gradient Boosting El **Gradient Boosting** construye modelos secuencialmente optimizando una función de pérdida arbitraria mediante descenso de gradiente.

1. Inicializar el modelo con una constante:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. Para $m = 1$ hasta M:

(a) Calcular los residuos:

$$r_{i,m} = -\left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right]$$

(b) Entrenar un modelo base h_m para predecir los residuos $r_{i,m}$.

(c) Actualizar el modelo:

$$F_m(x) = F_{m-1}(x) + \nu h_m(x)$$

donde ν es la tasa de aprendizaje.

3.6 Comparación entre Bagging y Boosting

- **Bagging:**
  - Reduce la varianza.
  - Los modelos base son entrenados independientemente.
  - Bueno para modelos de alto sesgo.
- **Boosting:**
  - Reduce el sesgo.
  - Los modelos base se entrenan secuencialmente.
  - Bueno para modelos de baja varianza.

3.7 Ventajas y Desventajas

Árboles de Decisión:

- **Ventajas:**
  - Fácil de interpretar y visualizar.
  - Maneja tanto datos numéricos como categóricos.
  - Requiere poco preprocesamiento de datos.
- **Desventajas:**
  - Propensos al sobreajuste.
  - Sensibles a pequeñas variaciones en los datos.

Métodos de Ensamble:

- **Ventajas:**
  - Mejoran la precisión del modelo.
  - Reducen la varianza y/o el sesgo.
  - Son más robustos al sobreajuste que los modelos individuales.
- **Desventajas:**
  - Menos interpretables que los árboles individuales.
  - Mayor complejidad computacional.

3.8 Consideraciones Prácticas

- **Selección de Hiperparámetros:** Es importante ajustar parámetros como la profundidad de los árboles, el número de estimadores y la tasa de aprendizaje en boosting.

- **Balance entre Sesgo y Varianza:** Entender las características del problema para elegir entre bagging y boosting.
- **Escalabilidad:** Los métodos de ensamble pueden ser computacionalmente intensivos; considerar el uso de paralelización.

3.9 Resumen

Los árboles de decisión son modelos intuitivos y fáciles de interpretar, pero pueden sufrir de sobreajuste. Los métodos de ensamble como bagging y boosting combinan múltiples modelos para mejorar el rendimiento y reducir problemas de sesgo y varianza. La elección del método adecuado depende de las características del problema y los datos disponibles.

4 Clase 4: Máquinas de Vectores de Soporte (SVM)

4.1 Introducción

Las **Máquinas de Vectores de Soporte (SVM)** son modelos de aprendizaje supervisado utilizados para tareas de clasificación y regresión. El objetivo principal es encontrar el hiperplano que mejor separa las clases de datos con el **máximo margen** posible.

4.2 Conceptos Clave

Margen: Es la distancia entre el hiperplano de separación y los puntos más cercanos de cualquier clase.

Vectores de Soporte: Son los puntos de datos que están más cercanos al hiperplano y que definen la posición y orientación del mismo.

4.3 Formulación del Problema

Para datos separables linealmente, buscamos un hiperplano definido por $\mathbf{w}$ y b que satisfaga:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x_i} + b \geq +1 & \text{si } y_i = +1 \\ \mathbf{w} \cdot \mathbf{x_i} + b \leq -1 & \text{si } y_i = -1 \end{cases}$$

El margen se maximiza minimizando $\|\mathbf{w}\|$, lo que conduce al siguiente problema de optimización:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{sujeto a } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1$$

4.4 Función de Costo Hinge y Regularización

Para datos no separables, introducimos variables de holgura ξ_i y el parámetro de regularización C:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{sujeto a } y_i(\mathbf{w} \cdot \mathbf{x_i} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

La **función de costo hinge** es:

$$L = \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + b))$$

4.5 Ejemplo Visual de SVM Lineal

4.6 Kernelización

Para problemas no linealmente separables, utilizamos **kernels** para transformar los datos a un espacio de mayor dimensión donde sean separables.

4.6.1 Ejemplo del Kernel RBF

El **kernel de base radial (RBF)** se define como:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma\|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Donde γ es un parámetro que controla el alcance de la influencia de cada punto de entrenamiento.

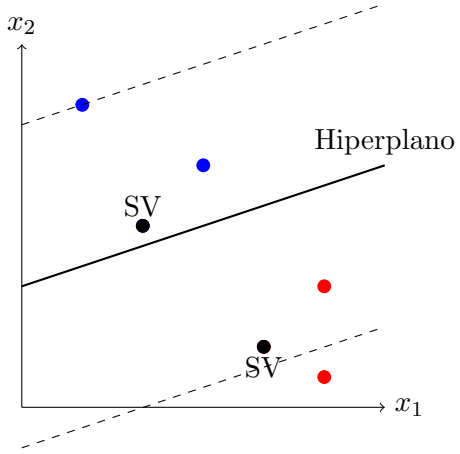

Figure 3: Máxima separación con SVM lineal

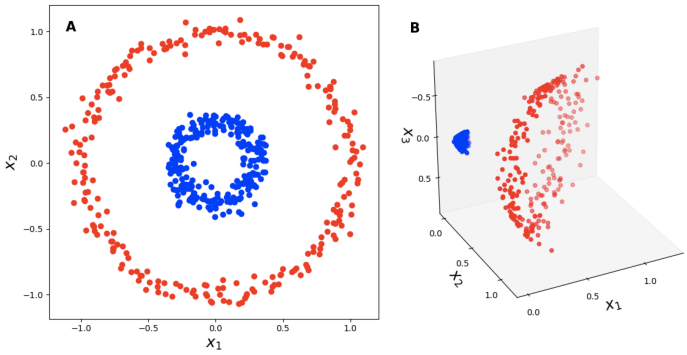

Figure 4: Transformación de datos con kernel RBF

En este ejemplo, los datos que no son separables linealmente en el espacio original se transforman a un espacio de mayor dimensión donde sí lo son.

4.7 Selección de Hiperparámetros

- **Parámetro C:** Controla la penalización por errores de clasificación.
  - C grande: Menos tolerante a errores, margen más pequeño.
  - C pequeño: Más tolerante a errores, margen más amplio.
- **Parámetro γ (en kernel RBF):** Controla la influencia de los puntos de datos individuales.
  - γ grande: Alcance corto, puede llevar a sobreajuste.
  - γ pequeño: Alcance amplio, puede llevar a subajuste.

4.8 Ejemplo Práctico con SVM

Supongamos un conjunto de datos donde las clases no son separables linealmente. Al aplicar un kernel RBF, podemos transformar los datos a un espacio de mayor dimensión donde una SVM puede encontrar un hiperplano que las separe.

4.9 Ventajas y Desventajas

Ventajas:

- Efectivo en espacios de alta dimensionalidad.
- Funciona bien cuando el número de dimensiones es mayor que el número de muestras.

- Versátil gracias al uso de diferentes kernels.

Desventajas:

- No es eficiente en conjuntos de datos muy grandes.
- Requiere preprocesamiento y ajuste cuidadoso de hiperparámetros.
- Difícil de interpretar comparado con modelos más simples.

4.10 Consideraciones Prácticas

- **Escalado de Características:** Es esencial escalar las características antes de entrenar una SVM.
- **Ajuste de Hiperparámetros:** Utilizar técnicas como la validación cruzada para encontrar los mejores valores de C y γ.

5 Clase 5: The Bias-Variance Tradeoff

5.1 Descomposición del Error de Test

El error de predicción puede descomponerse en tres componentes principales:

$$\text{Error} = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

5.1.1 Sesgo (*Bias*)

El **sesgo** mide la diferencia sistemática entre la predicción promedio del modelo y el valor verdadero:

$$\text{Bias}^2(x) = \left(\mathbb{E}[\hat{f}(x)] - f(x)\right)^2$$

El sesgo es el principal contribuyente al error en modelos subajustados (*underfitting*).

5.1.2 Varianza (*Variance*)

La **varianza** mide cuánto varían las predicciones para un punto de datos dado debido a diferentes realizaciones del modelo:

$$\text{Variance}(x) = \mathbb{E}\left[\left(\hat{f}(x) - \mathbb{E}[\hat{f}(x)]\right)^2\right]$$

Los modelos sobreajustados (*overfitting*) tienden a tener errores altos debido a la varianza.

5.2 Ejemplos del Tradeoff Sesgo-Varianza

5.2.1 LOESS Regression

En la regresión **LOESS**, el tradeoff sesgo-varianza se controla mediante el parámetro de suavización.

5.2.2 K-Nearest Neighbors (KNN)

En **KNN**, el parámetro K controla el tradeoff. Valores pequeños de K implican alta varianza, mientras que valores grandes conllevan alto sesgo.

6 Clase 6: Selección de Modelos

6.1 Evaluación

- Para saber si podemos *confiar* en nuestro método o sistema, necesitamos evaluarlo.
- Si no podemos medirlo, no podemos mejorarlo.
- Selección de modelo: elegir entre diferentes modelos usando un enfoque basado en datos.
- Convencer a otros de que el trabajo de uno es significativo.

6.1.1 Diseño de sistemas de machine learning

- Simplemente correr el algoritmo favorito de uno, generalmente no es una buena manera de comenzar.
- Considere el problema en general:
  - ¿Queremos entender los fenómenos o realizar un modelo caja negra?
  - ¿Cómo definir y medir el éxito? ¿Hay costos involucrados?
  - ¿Se tienen los datos correctos? ¿Cómo puedes mejorarlo?
- Construir prototipos desde el principio para evaluar los cambios a lo largo del tiempo.
- Analizar los errores del modelo que uno está construyendo (entrenando):
  - ¿Debería recopilar más datos, o datos adicionales?
  - ¿Debería reformularse la tarea?
- Deuda técnica: compensación de creación-mantenimiento:
  - Los sistemas de aprendizaje automático muy complejos son difíciles/imposibles de poner en práctica.
  - Revisar *Machine Learning: The High Interest Credit Card of Technical Debt* `https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43146.pdf`

6.1.2 Evaluaciones en el mundo real

- Evaluar predicciones, pero también cómo mejoraron los resultados *gracias a ellas*.
- Feedback loops: las predicciones se introducen en las entradas, como datos nuevos.
- La señal que encontró su modelo puede ser solo un artefacto de sus datos sesgados.
  - Cuando sea posible, intente *interpretar* lo que su modelo ha aprendido.
  - Revisar *Why Should I Trust You?* `https://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf` por Marco Ribeiro et al.

6.1.3 Técnicas de estimación de desempeño

- No tenemos acceso a futuras observaciones.
- Evaluar modelos *como si estuvieran prediciendo el futuro*.
- Reserve datos para una evaluación objetiva.

6.2 The holdout (partición simple train-test)

Ya hemos visto la forma más básica de evaluación.

6.2.1 Cross-validation

- ¿Qué pasa si una división aleatoria produce diferentes modelos (y scores) que otra?
- Reduzca el sesgo probando en cada punto exactamente una vez.
- *Validación cruzada de k-dobladas* o *k-fold cross-validation* (CV): particionar (split) aleatoriamente en k partes del mismo tamaño, llamadas *folds*. Supongamos $k = 5$:
  - Primero, el fold 1 es el conjunto de prueba, y los folds 2-5 comprenden el conjunto de entrenamiento.
  - Luego, el fold 2 es el conjunto de prueba, los folds 1,3,4,5 comprenden el conjunto de entrenamiento.
  - Calcule las k puntuaciones de evaluación, agregue después (por ejemplo, tome la media).

6.2.2 Beneficios y desventajas de cross-validation

- Más robusto: cada ejemplo de entrenamiento estará en un conjunto de prueba exactamente una vez.
  - El modelo se evalúa en todas las muestras, debe funcionar bien en todas.
  - En una partición train-test, podemos tener:
    - ‘suerte’: todos los ejemplos sencillos están en el conjunto de prueba.
    - ‘desafortunado’: todos los ejemplos difíciles están en el conjunto de prueba.
- Muestra cuán *sensible* es el modelo al conjunto de entrenamiento exacto.
- Mejor estimación del rendimiento real.

6.3 Matriz de confusión y métricas de clasificación

La **matriz de confusión** es una herramienta útil para evaluar el rendimiento de un clasificador. Para una clasificación binaria, la matriz de confusión se organiza de la siguiente manera:

	Predicción Positiva	Predicción Negativa
Clase Positiva	Verdaderos Positivos (TP)	Falsos Negativos (FN)
Clase Negativa	Falsos Positivos (FP)	Verdaderos Negativos (TN)

A partir de esta matriz, podemos calcular diversas métricas importantes:

6.3.1 Precisión (Accuracy)

La **precisión** es la proporción de predicciones correctas entre el total de predicciones realizadas.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

6.3.2 Precision

La **precision** se utiliza para medir la calidad de las predicciones positivas. Indica cuántas de las predicciones positivas fueron correctas.

$$\text{Precision} = \frac{TP}{TP + FP}$$

6.3.3 Recall

El **recall** o *sensibilidad* mide cuántas de las muestras positivas reales fueron correctamente identificadas.

$$\text{Recall} = \frac{TP}{TP + FN}$$

6.3.4 F1-Score

El **F1-score** es la media armónica entre la precision y el recall. Es útil cuando buscamos un equilibrio entre ambos.

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

6.3.5 Área bajo la curva ROC (AUC)

El **AUC** es una métrica basada en la curva ROC que mide la capacidad del modelo para separar correctamente las clases. Es especialmente útil en conjuntos de datos desbalanceados, ya que es insensible al desbalanceo de clases.

6.4 Métricas para Regresión

En los problemas de regresión, las métricas comunes incluyen:

6.4.1 Error cuadrático medio (MSE)

El **Mean Squared Error (MSE)** mide el promedio de los errores al cuadrado entre las predicciones y los valores verdaderos.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

6.4.2 Error absoluto medio (MAE)

El **Mean Absolute Error (MAE)** mide el promedio de los errores absolutos entre las predicciones y los valores verdaderos, menos sensible a los valores atípicos.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

6.4.3 Coeficiente de determinación (R^2)

El **R-squared** mide la proporción de la varianza explicada por el modelo en comparación con la varianza total. Tiene un valor entre 0 y 1, y puede ser negativo si el modelo es peor que predecir la media.

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

6.5 Resumen I

6.5.1 k-fold Cross-validation

- Elija k dependiendo de la cantidad de datos que tenga.
  - k grande es más lento, pero permite más datos de entrenamiento.
  - 10-fold, 5-fold, 5x2-fold son las opciones más populares.
- Utilice siempre la estratificación para la clasificación (especialmente en los casos desbalanceados).
- División Train-test y Shuffle-split: útil para grandes conjuntos de datos.
- Utilice grouping cuando se quiera generalizar sobre grupos.

6.5.2 Selección de modelo

- No agregue sobre los puntajes de test: aquellos que han visto los datos del test set.
- Utilice conjuntos de validación para elegir primero algoritmos/hiperparámetros.

6.5.3 Optimización

- **Grid Search**: exhaustivo pero simple.
- **Random Search**: escala mejor.

6.6 Resumen II

- Los datos del mundo real a menudo están desbalanceados.
- Los falsos positivos pueden ser mucho peores que los falsos negativos (o viceversa).
- Clasificación binaria:
  - Seleccione métricas que puedan distinguir diferentes tipos de errores (precision, recall, F1-score, AUC,...).
  - Calibre los umbrales de decisión para la tarea en cuestión.
  - Curvas de Precision-Recall y ROC: elija el mejor umbral o tome el área debajo de la curva.
- Clasificación multiclase:
  - Promedio Macro/Micro/weighted de puntajes por clase (one-vs-all).
- Regresión:
  - (Root) mean squared/absolute error desde 0..Inf.
  - R^2 más fácil de interpretar.
- Todas las medidas se pueden utilizar en cross-validation o grid/random search.
- Clasificación Cost-sensitive: optimizar para cualquier matriz de costos o función de costos.

7 Comparativas

En esta sección, se comparan los diferentes algoritmos estudiados en las clases anteriores, evaluando sus principales diferencias, fortalezas, debilidades y casos de uso recomendados. A continuación, se presentan tablas comparativas que resumen esta información.

Característica	**kNN**	**Árboles de Decisión**	**SVM**
Tipo de Modelo	No paramétrico	Paramétrico	Paramétrico
Aprendizaje	Instancia	Basado en reglas	Margen máximo
Uso	Clasificación/Regresión	Clasificación/Regresión	Clasificación/Regresión
Interpretabilidad	Baja	Alta	Baja
Escalabilidad	Baja	Media	Baja
Manejo de Datos	Datos pequeños	Datos medianos	Datos pequeños
Sensibilidad a Dimensionalidad	Alta	Media	Media
Resistencia al Overfitting	Baja (k pequeño)	Baja (árbol profundo)	Alta
Principales Hiperparámetros	k, distancia	Profundidad, criterios de división	C, kernel, γ
Fortalezas	Fácil de implementar	Fácil de interpretar	Maneja datos no lineales
Debilidades	Lento en predicción	Sobreajuste si no se poda	Requiere ajuste de hiperparámetros

Table 2: Comparativa de otros algoritmos.

Característica	**Bagging**	**Random Forest**	**AdaBoost**	**Gradient Boosting**
Tipo de Modelo	Ensamble	Ensamble	Ensamble	Ensamble
Aprendizaje	Basado en ensamble	Basado en ensamble	Basado en ensamble	Basado en ensamble
Uso	Clasificación/Regresión	Clasificación/Regresión	Clasificación	Clasificación/Regresión
Interpretabilidad	Media	Media	Baja	Baja
Escalabilidad	Baja	Media	Baja	Baja
Manejo de Datos	Datos medianos	Datos grandes	Datos medianos	Datos medianos
Sensibilidad a Dimensionalidad	Media	Media	Media	Media
Resistencia al Overfitting	Alta	Alta	Media	Media
Principales Hiperparámetros	Número de modelos	Número de árboles, profundidad	Número de modelos, tasa de aprendizaje	Número de modelos, tasa de aprendizaje
Fortalezas	Reduce varianza	Reduce varianza	Mejora precisión	Mejora precisión
Debilidades	Requiere recursos	Complejo	Sensible al ruido	Sensible al ruido

Table 3: Comparativa de algoritmos de ensamble.

7.1 Casos de Uso y Selección de Algoritmo

En la siguiente tabla se resumen las situaciones en las que es más adecuado utilizar cada algoritmo, según las características del problema y los datos disponibles.

7.2 Principales Diferencias entre los Algoritmos

- **kNN vs. Árboles de Decisión:** kNN es un método basado en instancias que requiere almacenar todo el conjunto de datos y es sensible a la dimensionalidad, mientras que los árboles de decisión generan un modelo interpretable y manejan mejor características irrelevantes.
- **Ensamble (Bagging, Random Forest) vs. Modelos Simples:** Los ensambles combinan múltiples modelos para reducir la varianza y mejorar la precisión, a costa de una mayor complejidad y menor interpretabilidad.
- **Boosting (AdaBoost, Gradient Boosting) vs. Bagging:** Mientras que Bagging busca reducir la varianza mediante la combinación de modelos independientes, Boosting reduce el sesgo construyendo secuencialmente modelos que corrigen los errores de los anteriores.
- **SVM vs. Otros Algoritmos:** Las SVM son eficaces en espacios de alta dimensionalidad y pueden manejar relaciones no lineales mediante kernels, pero requieren un ajuste cuidadoso de hiperparámetros y pueden ser computacionalmente costosas con grandes conjuntos de datos.

Algoritmo	Casos de Uso Recomendados
kNN	- Problemas con conjuntos de datos pequeños. - Cuando se requiere un modelo rápido de entrenar. - Cuando no se necesita interpretabilidad.
Árboles de Decisión	- Cuando se requiere interpretabilidad. - Datos con relaciones no lineales. - Problemas de clasificación y regresión.
Bagging	- Reducir varianza de modelos inestables. - Cuando se dispone de recursos computacionales. - Mejorar precisión en datos medianos.
Random Forest	- Conjuntos de datos grandes. - Problemas con muchas características. - Necesidad de reducir varianza y manejar overfitting.
AdaBoost	- Cuando se necesita mejorar el rendimiento de clasificadores débiles. - Datos con ruido moderado. - Problemas de clasificación binaria.
Gradient Boosting	- Alta precisión en clasificación y regresión. - Datos con patrones complejos. - Cuando se pueden ajustar hiperparámetros cuidadosamente.
SVM	- Datos de alta dimensionalidad. - Problemas de clasificación con fronteras complejas. - Cuando el número de muestras es menor que el de características.

Table 4: Recomendaciones para la selección de algoritmos según el caso de uso.

7.3 Evaluación para la Selección de Algoritmo

Al seleccionar un algoritmo, se deben considerar los siguientes aspectos:

- **Tamaño y Dimensionalidad de los Datos:** Algoritmos como kNN y SVM pueden no escalar bien con conjuntos de datos muy grandes.
- **Interpretabilidad:** Si es crucial entender las decisiones del modelo, los árboles de decisión son preferibles.
- **Recursos Computacionales:** Ensambles y SVM pueden requerir más tiempo y recursos para entrenar y predecir.
- **Rendimiento y Precisión:** Ensambles como Random Forest y Gradient Boosting suelen ofrecer mayor precisión a costa de interpretabilidad.
- **Presencia de Ruido y Outliers:** Algoritmos como AdaBoost pueden ser sensibles al ruido, mientras que Random Forest es más robusto.
- **Necesidad de Ajuste de Hiperparámetros:** Modelos como SVM y Gradient Boosting requieren un ajuste cuidadoso para obtener un buen rendimiento.

7.4 Conclusión

La elección del algoritmo adecuado depende de múltiples factores, incluyendo las características del conjunto de datos, los recursos disponibles y los objetivos específicos del problema. Las tablas y comparativas presentadas sirven como guía para orientar esta selección, pero es recomendable realizar pruebas y validaciones cruzadas para determinar el mejor modelo para cada caso particular.