

# ALGORITMOS DE CLASIFICACIÓN

*Material de Estudio Avanzado - Nivel Maestría*

## REGRESIÓN LOGÍSTICA

*Logistic Regression - Modelo Lineal Generalizado*

---

### Fundamento Teórico

La regresión logística es un modelo discriminativo que estima directamente la probabilidad posterior  $P(Y|X)$  mediante una función sigmoide. A diferencia de la regresión lineal, utiliza la función logística para mapear cualquier valor real al intervalo  $[0,1]$ , haciéndola ideal para clasificación binaria. Se basa en el principio de máxima verosimilitud (Maximum Likelihood Estimation) para estimar los parámetros del modelo.

### Formulación Matemática

#### Función Sigmoide (Logística):

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z}$$

donde  $z = \beta_0 + \beta_1x_1 + \beta_2x_2 + \dots + \beta_nx_n = \boldsymbol{\beta}^T\mathbf{x}$

#### Probabilidad de Clasificación:

$$P(Y = 1|\mathbf{x}) = \sigma(\boldsymbol{\beta}^T\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\beta}^T\mathbf{x}}}$$

$$P(Y = 0|\mathbf{x}) = 1 - P(Y = 1|\mathbf{x}) = \frac{e^{-\boldsymbol{\beta}^T\mathbf{x}}}{1 + e^{-\boldsymbol{\beta}^T\mathbf{x}}}$$

### **Log-Odds (Logit):**

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right) = \boldsymbol{\beta}^T \mathbf{x}$$

La transformación logit convierte probabilidades en log-odds, permitiendo modelar la relación lineal.

### **Función de Pérdida (Log-Likelihood Negativa):**

$$\mathcal{L}(\boldsymbol{\beta}) = - \sum_{i=1}^n [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)]$$

donde  $\hat{p}_i = \sigma(\boldsymbol{\beta}^T \mathbf{x}_i)$  es la probabilidad predicha para la muestra  $i$ .

### **Gradiente de la Función de Pérdida:**

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}} = \sum_{i=1}^n (\hat{p}_i - y_i) \mathbf{x}_i = \mathbf{X}^T (\hat{\mathbf{p}} - \mathbf{y})$$

### **Actualización de Parámetros (Gradiente Descendente):**

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{\beta}}$$

donde  $\alpha$  es la tasa de aprendizaje (learning rate).

### **Regresión Logística con Regularización L2 (Ridge):**

$$\mathcal{L}_{reg}(\boldsymbol{\beta}) = - \sum_{i=1}^n [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)] + \lambda \sum_{j=1}^p \beta_j^2$$

### **Regresión Logística con Regularización L1 (Lasso):**

$$\mathcal{L}_{reg}(\boldsymbol{\beta}) = - \sum_{i=1}^n [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)] + \lambda \sum_{j=1}^p |\beta_j|$$

## Funcionamiento del Algoritmo

1. **Inicialización:** Los parámetros  $\boldsymbol{\beta}$  se inicializan (típicamente en ceros o valores aleatorios pequeños).
2. **Propagación hacia adelante:** Para cada muestra, se calcula  $z = \boldsymbol{\beta}^T \mathbf{x}$  y luego  $\hat{p} = \sigma(z)$ .
3. **Cálculo de pérdida:** Se evalúa la función de pérdida (cross-entropy) en todo el conjunto de entrenamiento.
4. **Cálculo del gradiente:** Se deriva la función de pérdida respecto a  $\boldsymbol{\beta}$  para obtener la dirección de mayor descenso.
5. **Actualización de parámetros:** Se ajustan los coeficientes  $\boldsymbol{\beta}$  en la dirección opuesta al gradiente, multiplicado por la tasa de aprendizaje.
6. **Iteración:** Los pasos 2-5 se repiten hasta convergencia (cuando el cambio en la pérdida es menor que un umbral  $\epsilon$ ).
7. **Predicción:** Para una nueva muestra, si  $P(Y = 1 | \mathbf{x}) \geq 0.5$ , se clasifica como clase 1; de lo contrario, clase 0.

### Nota Técnica: Newton-Raphson

En la práctica, métodos de segundo orden como Newton-Raphson o L-BFGS son preferidos para optimización, ya que convergen más rápido que el gradiente descendente estándar. Newton-Raphson utiliza la matriz Hessiana (segundas derivadas) para actualizar parámetros:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \mathbf{H}^{-1} \nabla \mathcal{L}$$

## Extensiones Multiclas

### Regresión Logística Multinomial (Softmax):

$$P(Y = k|\boldsymbol{x}) = \frac{e^{\beta_k^T \boldsymbol{x}}}{\sum_{j=1}^K e^{\beta_j^T \boldsymbol{x}}}$$

donde  $K$  es el número de clases. Para evitar indeterminación, se fija  $\beta_K = \mathbf{0}$  (clase de referencia).

### Función de Pérdida Cross-Entropy Multiclasificación:

$$\mathcal{L}(\boldsymbol{\beta}) = - \sum_{i=1}^n \sum_{k=1}^K y_{ik} \log(\hat{p}_{ik})$$

donde  $y_{ik}$  es un indicador one-hot (1 si la muestra  $i$  pertenece a la clase  $k$ , o de lo contrario).

## Aplicaciones Principales

### Medicina y Salud

Predicción de riesgo de enfermedad, diagnóstico médico, análisis de supervivencia, predicción de readmisión hospitalaria

### Finanzas

Credit scoring, detección de fraude, predicción de default, aprobación de préstamos, análisis de riesgo crediticio

### Marketing

Predicción de churn, propensión de compra, segmentación de clientes, respuesta a campañas, A/B testing

### Ciencias Sociales

Modelos de elección binaria, análisis de encuestas, predicción de comportamiento electoral, estudios epidemiológicos

## Métricas de Rendimiento

Métrica	Fórmula	Interpretación
<b>Accuracy</b>	$\frac{TP+TN}{TP+TN+FP+FN}$	Proporción de predicciones correctas
<b>Precision</b>	$\frac{TP}{TP+FP}$	Proporción de positivos predichos correctamente

<b>Recall (Sensitivity)</b>	$\frac{TP}{TP+FN}$	Proporción de positivos reales identificados
<b>F1-Score</b>	$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$	Media armónica de precision y recall
<b>AUC-ROC</b>	$\int_0^1 TPR(FPR^{-1}(x))dx$	Área bajo la curva ROC (0.5 = aleatorio, 1.0 = perfecto)
<b>Log-Loss</b>	$-\frac{1}{n} \sum [y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})]$	Penaliza predicciones confiadas pero incorrectas

## Ventajas y Limitaciones

### Ventajas:

- Interpretable: los coeficientes  $\beta_j$  representan el cambio en log-odds por unidad de  $x_j$
- Eficiente computacionalmente, escalable a grandes datasets
- Salida probabilística bien calibrada
- No requiere supuestos de normalidad de las características
- Robusto con regularización L1/L2
- Extensible a clasificación multiclas

### Limitaciones:

- Asume linealidad en el espacio logit (relación lineal entre log-odds y características)
- Sensible a multicolinealidad entre características
- Requiere características relativamente independientes
- Puede presentar underfitting en relaciones complejas no lineales
- Sensible a outliers en el espacio de características
- Requiere más datos cuando se aumenta la dimensionalidad

## Complejidad Computacional

- **Entrenamiento (Gradiente Descendente):**  $O(n \cdot m \cdot k)$  donde  $n$  = número de muestras,  $m$  = número de características,  $k$  = número de iteraciones
- **Entrenamiento (Newton-Raphson):**  $O(n \cdot m^2 + m^3)$  por iteración (cálculo de Hessiana e inversión)
- **Predicción:**  $O(m)$  por muestra (simple producto punto y función sigmoide)
- **Espacio:**  $O(m)$  para almacenar parámetros  $\beta$

## SUPPORT VECTOR MACHINES (SVM)

*Máquinas de Vectores de Soporte - Maximización del Margen*

---

### Fundamento Teórico

Las SVM son clasificadores discriminativos que buscan el hiperplano óptimo que maximiza el margen entre clases. Se basan en la teoría de aprendizaje estadístico de Vapnik-Chervonenkis. El principio fundamental es encontrar la frontera de decisión con máxima distancia mínima a los puntos de entrenamiento más cercanos (vectores de soporte). Este enfoque proporciona robustez y buena generalización. El truco del kernel (kernel trick) permite proyectar datos no linealmente separables a espacios de mayor dimensión donde sí lo son.

### Formulación Matemática - SVM Lineal de Margen Duro

#### Hiperplano de Separación:

$$\mathbf{w}^T \mathbf{x} + b = 0$$

donde  $\mathbf{w}$  es el vector de pesos normal al hiperplano y  $b$  es el término de sesgo.

#### Función de Decisión:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + b)$$

Clasifica en  $+1$  si  $\mathbf{w}^T \mathbf{x} + b \geq 0$ , y en  $-1$  en caso contrario.

### Margen Geométrico:

$$\gamma = \frac{2}{\|\mathbf{w}\|}$$

El margen es la distancia entre el hiperplano y los vectores de soporte más cercanos.

### Problema de Optimización Primal (Margen Duro):

$$\min_{w,b} \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{sujeto a: } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \quad \forall i = 1, \dots, n$$

Minimizar  $\|\mathbf{w}\|$  equivale a maximizar el margen  $\gamma$ . Las restricciones garantizan clasificación correcta con margen mínimo de 1.

## SVM de Margen Suave (Soft Margin)

### Problema de Optimización con Variables de Holgura:

$$\min_{w,b,\xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

$$\text{sujeto a: } y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$$

$\xi_i$  son variables de holgura que permiten violaciones del margen.  $C$  es el parámetro de regularización que controla el trade-off entre maximizar el margen y minimizar errores de clasificación.

### Interpretación del parámetro C:

- **C grande:** Penaliza fuertemente errores → margen más pequeño, más ajuste a datos (riesgo de overfitting)
- **C pequeño:** Tolera errores → margen más amplio, mayor generalización (riesgo de underfitting)

underfitting)

## Formulación Dual y Multiplicadores de Lagrange

### Lagrangiano del Problema Primal:

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \mu) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i [y_i (\mathbf{w}^T \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^n \mu_i \xi_i$$

donde  $\alpha_i \geq 0$  y  $\mu_i \geq 0$  son multiplicadores de Lagrange.

### Problema Dual (después de aplicar condiciones KKT):

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{sujeto a: } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^n \alpha_i y_i = 0$$

### Vector de Pesos Óptimo:

$$\mathbf{w}^* = \sum_{i=1}^n \alpha_i^* y_i \mathbf{x}_i$$

Solo los vectores de soporte (donde  $\alpha_i > 0$ ) contribuyen a  $\mathbf{w}^*$ .

### Sesgo Óptimo:

$$b^* = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left( y_i - \sum_{j=1}^n \alpha_j^* y_j \mathbf{x}_j^T \mathbf{x}_i \right)$$

donde  $\mathcal{S}$  es el conjunto de vectores de soporte con  $0 < \alpha_i < C$ .

### Función de Decisión:

$$f(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^n \alpha_i^* y_i \mathbf{x}_i^T \mathbf{x} + b^* \right)$$

## Kernel Trick y SVM No Lineal

El kernel trick permite trabajar implícitamente en espacios de alta dimensión sin calcular explícitamente la transformación  $\phi(\mathbf{x})$ . Se reemplaza el producto punto  $\mathbf{x}_i^T \mathbf{x}_j$  por una función kernel  $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ .

### Problema Dual con Kernel:

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{sujeto a: } 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^n \alpha_i y_i = 0$$

### Función de Decisión con Kernel:

$$f(\mathbf{x}) = \text{sign} \left( \sum_{i=1}^n \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}) + b^* \right)$$

### Kernel Lineal:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

### Kernel Polinomial:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d$$

donde  $d$  es el grado del polinomio,  $\gamma$  es un parámetro de escala, y  $r$  es el coeficiente independiente.

### Kernel RBF (Radial Basis Function o Gaussiano):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$$

donde  $\gamma = \frac{1}{2\sigma^2}$ . Es el kernel más popular por su flexibilidad. Proyecta a un espacio infinito-dimensional.

### Kernel Sigmoide:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$$

Similar a redes neuronales de dos capas. No siempre es una función kernel válida (debe satisfacer condiciones de Mercer).

### Condiciones de Mercer:

Para que una función  $K$  sea un kernel válido, debe ser:

- Simétrica:  $K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i)$
- Semi-definida positiva: la matriz Gram  $\mathbf{K}$  debe tener autovalores no negativos

## Funcionamiento del Algoritmo

1. **Preprocesamiento:** Normalizar/estandarizar características (crítico para SVM).
2. **Selección de kernel:** Elegir kernel apropiado (lineal para datos linealmente separables, RBF para relaciones no lineales complejas).
3. **Optimización:** Resolver el problema dual usando algoritmos especializados:
  - **SMO (Sequential Minimal Optimization):** Optimiza pares de  $\alpha_i$  iterativamente
  - **Chunking:** Divide problema grande en subproblemas manejables
  - **Decomposition methods:** Working set selection strategies

4. **Identificación de vectores de soporte:** Muestras con  $\alpha_i > 0$  se convierten en vectores de soporte.
5. **Cálculo del sesgo:** Usar vectores de soporte marginales ( $0 < \alpha_i < C$ ) para calcular  $b^*$ .
6. **Predicción:** Evaluar función de decisión  $f(\mathbf{x})$  usando solo vectores de soporte.

## SVM Multiclasificación

SVM es naturalmente binario. Para problemas multiclasificación se utilizan estrategias:

### One-vs-Rest (OvR) o One-vs-All:

Entrenar  $K$  clasificadores binarios, uno por clase. Cada clasificador distingue una clase de todas las demás.

$$f_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + b_k$$

Clasificación:  $\arg \max_k f_k(\mathbf{x})$

### One-vs-One (OvO):

Entrenar  $\binom{K}{2} = \frac{K(K-1)}{2}$  clasificadores, uno por cada par de clases. Clasificación por votación: cada clasificador vota por una clase, se elige la clase con más votos.

### Comparación OvR vs OvO:

- **OvR:** Menos modelos ( $K$ ), más rápido, pero puede sufrir de desbalanceo de clases
- **OvO:** Más modelos ( $\frac{K(K-1)}{2}$ ), pero cada modelo entrena con menos datos y puede ser más preciso

## Aplicaciones Principales

### Visión por Computadora

Reconocimiento facial, detección de objetos, clasificación de imágenes,

### Bioinformática

Clasificación de proteínas, predicción de estructura secundaria, clasificación de

reconocimiento de escritura manuscrita  
(MNIST)

secuencias de ADN, análisis de expresión  
génica

### Text Mining

Clasificación de documentos,  
categorización de noticias, análisis de  
sentimientos, detección de spam

### Detección de Anomalías

Intrusión en redes, detección de fraude  
financiero, control de calidad industrial,  
diagnóstico de fallas

## Métricas de Rendimiento

### Margin de Confianza:

La distancia de un punto al hiperplano indica confianza:

$$\text{confidence}(\mathbf{x}) = \frac{|f(\mathbf{x})|}{\|\mathbf{w}\|} = \frac{|\mathbf{w}^T \mathbf{x} + b|}{\|\mathbf{w}\|}$$

### Hinge Loss (función de pérdida de SVM):

$$\mathcal{L}_{\text{hinge}}(y, f(\mathbf{x})) = \max(0, 1 - y \cdot f(\mathbf{x}))$$

Pérdida cero si la muestra está correctamente clasificada fuera del margen ( $y \cdot f(\mathbf{x}) \geq 1$ ).

Penaliza linealmente violaciones del margen.

## Ventajas y Limitaciones

### Ventajas:

- Efectivo en espacios de alta dimensión (incluso cuando  $m > n$ )
- Memoria eficiente: solo almacena vectores de soporte (típicamente pequeño subconjunto)
- Versátil con diferentes funciones kernel
- Robusto frente a overfitting, especialmente en alta dimensión
- Garantías teóricas sólidas (teoría PAC, dimensión VC)

- Función de decisión determinada solo por vectores de soporte, robusto a outliers lejanos

### **Limitaciones:**

- Costoso computacionalmente para grandes datasets:  $O(n^2)$  a  $O(n^3)$
- Sensible a elección de kernel y ajuste de hiperparámetros ( $C, \gamma$ )
- No proporciona estimaciones probabilísticas directas (requiere Platt scaling o métodos adicionales)
- Dificultad con datasets muy grandes ( $n > 10^5$ )
- Requiere normalización cuidadosa de características
- Interpretabilidad limitada con kernels no lineales

## **Complejidad Computacional**

- **Entrenamiento (método general):**  $O(n^2 \cdot m)$  a  $O(n^3 \cdot m)$  debido a la optimización cuadrática
- **Entrenamiento (SMO):** Entre  $O(n^2)$  y  $O(n^3)$ , pero mucho más eficiente en práctica
- **Predicción:**  $O(n_{sv} \cdot m)$  donde  $n_{sv}$  es el número de vectores de soporte (típicamente  $n_{sv} \ll n$ )
- **Espacio:**  $O(n_{sv} \cdot m)$  para almacenar vectores de soporte

## **K-NEAREST NEIGHBORS (KNN)**

*K Vecinos Más Cercanos - Aprendizaje Basado en Instancias*

---

### **Fundamento Teórico**

KNN es un algoritmo de aprendizaje no paramétrico y basado en instancias (instance-based o lazy learning). No construye un modelo explícito durante el entrenamiento; simplemente

almacena el dataset completo. La clasificación se basa en el principio de que puntos cercanos en el espacio de características tienden a pertenecer a la misma clase. Es un método de aprendizaje local que hace predicciones basándose únicamente en la vecindad inmediata del punto a clasificar. La decisión se toma por votación democrática o promedio ponderado de los K vecinos más cercanos.

## Formulación Matemática

### Distancia Euclídea (L2):

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2 = \sqrt{\sum_{k=1}^m (x_{ik} - x_{jk})^2}$$

La métrica más común. Mide la distancia "en línea recta" entre dos puntos.

### Distancia Manhattan (L1 o City Block):

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 = \sum_{k=1}^m |x_{ik} - x_{jk}|$$

Suma de diferencias absolutas. Útil cuando las características tienen escalas muy diferentes o en espacios de alta dimensión.

### Distancia Minkowski (generalización):

$$d(\mathbf{x}_i, \mathbf{x}_j) = \left( \sum_{k=1}^m |x_{ik} - x_{jk}|^p \right)^{1/p}$$

Generaliza L1 ( $p = 1$ ) y L2 ( $p = 2$ ). Cuando  $p \rightarrow \infty$ , se obtiene la distancia de Chebyshev:

$$d = \max_k |x_{ik} - x_{jk}|$$

### Distancia de Mahalanobis:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \Sigma^{-1} (\mathbf{x}_i - \mathbf{x}_j)}$$

donde  $\Sigma$  es la matriz de covarianza. Considera correlaciones entre características y es invariante a transformaciones lineales.

### Distancia Coseno (similaridad angular):

$$d(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\| \cdot \|\mathbf{x}_j\|} = 1 - \cos(\theta)$$

Mide el ángulo entre vectores. Útil en espacios de alta dimensión (ej. text mining, donde la magnitud es menos importante que la dirección).

### Distancia de Hamming (para variables binarias):

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{k=1}^m \mathbb{1}(x_{ik} \neq x_{jk})$$

Cuenta el número de posiciones donde los bits difieren. Usada para strings y datos categóricos binarios.

## Regla de Clasificación

### KNN por Votación Mayoritaria (clasificación):

$$\hat{y} = \arg \max_{c \in \mathcal{C}} \sum_{i \in \mathcal{N}_K(\mathbf{x})} \mathbb{1}(y_i = c)$$

donde  $\mathcal{N}_K(\mathbf{x})$  es el conjunto de índices de los K vecinos más cercanos a  $\mathbf{x}$ , y  $\mathcal{C}$  es el conjunto de clases.

### KNN Ponderado por Distancia:

$$\hat{y} = \arg \max_{c \in \mathcal{C}} \sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i \cdot \mathbb{1}(y_i = c)$$

$$w_i = \frac{1}{d(\mathbf{x}, \mathbf{x}_i) + \epsilon} \quad \text{o} \quad w_i = \exp \left( -\frac{d(\mathbf{x}, \mathbf{x}_i)^2}{2\sigma^2} \right)$$

Asigna mayor peso a vecinos más cercanos.  $\epsilon$  es una constante pequeña para evitar división por cero.

### KNN para Regresión (promedio):

$$\hat{y} = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x})} y_i$$

### KNN para Regresión (ponderado):

$$\hat{y} = \frac{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i \cdot y_i}{\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i}$$

### Estimación de Probabilidad de Clase:

$$P(Y = c | \mathbf{x}) = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x})} \mathbb{1}(y_i = c)$$

Proporción de vecinos que pertenecen a la clase  $c$ . Proporciona una estimación de probabilidad.

## Selección del Parámetro K

La elección de K es crítica y afecta directamente el sesgo-varianza del modelo:

### K pequeño (K=1):

- **Sesgo bajo:** Modelo muy flexible, captura detalles finos
- **Varianza alta:** Sensible a ruido y outliers
- **Frontera de decisión:** Irregular y compleja
- **Riesgo:** Overfitting

### K grande ( $K \approx n$ ):

- **Sesgo alto:** Modelo menos flexible, suaviza decisiones
- **Varianza baja:** Robusto al ruido
- **Frontera de decisión:** Más suave y simple
- **Riesgo:** Underfitting

### Heurísticas para seleccionar K:

- **Regla general:**  $K = \sqrt{n}$  como punto de partida
- **K impar:** Para clasificación binaria, usar K impar evita empates
- **Validación cruzada:** Método más robusto - probar varios valores de K y elegir el que minimiza error de validación
- **Elbow method:** Graficar error vs K y buscar el "codo" de la curva

### Error de Validación Cruzada Leave-One-Out (LOOCV):

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^n \mathbb{1}(y_i \neq \hat{y}_{-i})$$

donde  $\hat{y}_{-i}$  es la predicción para  $x_i$  usando todos los datos excepto la muestra  $i$ .

## Funcionamiento del Algoritmo

### 1. Fase de Entrenamiento (Training Phase):

- Almacenar todo el dataset de entrenamiento  $\{(x_i, y_i)\}_{i=1}^n$
- No hay construcción de modelo ni optimización de parámetros
- Tiempo:  $O(1)$  (instantáneo)

### 2. Preprocesamiento crítico:

- Normalización/estandarización de características (OBLIGATORIA)
- Sin normalización, características con mayor escala dominan la distancia
- Z-score:  $x' = \frac{x-\mu}{\sigma}$

- Min-Max:  $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$

### 3. Fase de Predicción (Query Phase):

- Dado un nuevo punto  $\mathbf{x}_{query}$ :
- Calcular distancia a todas las muestras de entrenamiento:  $d(\mathbf{x}_{query}, \mathbf{x}_i)$  para  $i = 1, \dots, n$
- Ordenar distancias en orden ascendente
- Seleccionar los K vecinos más cercanos
- Clasificación: votación mayoritaria (o ponderada)
- Regresión: promedio (o ponderado) de valores

#### Estructuras de Datos para Eficiencia:

Para datasets grandes, calcular distancias a todas las muestras es prohibitivo. Se usan estructuras espaciales:

- **KD-Tree:** Árbol binario que partitiona el espacio. Búsqueda:  $O(\log n)$  en baja dimensión, pero degenera a  $O(n)$  en alta dimensión (curse of dimensionality)
- **Ball Tree:** Similar a KD-Tree pero usa hiperesferas. Mejor que KD-Tree en dimensiones moderadas
- **Locality Sensitive Hashing (LSH):** Aproximación probabilística. Muy eficiente en alta dimensión
- **Cover Tree:** Garantiza búsqueda en  $O(\log n)$  bajo ciertas condiciones

## Propiedades Teóricas

### Límite de Error de 1-NN (Cover y Hart, 1967):

$$R^* \leq R_{1NN} \leq 2R^*$$

donde  $R^*$  es el error de Bayes (error óptimo teórico) y  $R_{1NN}$  es el error de 1-NN cuando  $n \rightarrow \infty$ . El error de 1-NN está acotado al doble del error óptimo.

### Consistencia de KNN:

KNN es universalmente consistente si:

- $K \rightarrow \infty$  cuando  $n \rightarrow \infty$
- $\frac{K}{n} \rightarrow 0$  cuando  $n \rightarrow \infty$

Esto significa que  $\lim_{n \rightarrow \infty} R_{KNN} = R^*$  bajo estas condiciones.

### Maldición de la Dimensionalidad:

En alta dimensión ( $m$  grande), la razón entre distancia máxima y mínima tiende a 1:

$$\lim_{m \rightarrow \infty} \frac{d_{max} - d_{min}}{d_{min}} \rightarrow 0$$

Esto significa que todos los puntos se vuelven equidistantes, y el concepto de "vecino cercano" pierde significado. Para mantener densidad constante en el espacio, se requiere  $n = O(e^m)$  muestras.

## Variantes y Extensiones

### Radius Neighbors (vecinos por radio):

En lugar de K vecinos fijos, usar todos los vecinos dentro de un radio  $r$ :

$$\mathcal{N}_r(\mathbf{x}) = \{i : d(\mathbf{x}, \mathbf{x}_i) \leq r\}$$

Útil cuando la densidad de datos varía en diferentes regiones del espacio.

### Locally Weighted KNN:

Ajustar un modelo local (ej. regresión lineal) en la vecindad:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b$$

donde  $\mathbf{w}$  y  $b$  se estiman minimizando:

$$\sum_{i \in \mathcal{N}_K(\mathbf{x})} w_i (y_i - \mathbf{w}^T \mathbf{x}_i - b)^2$$

### Adaptive KNN:

Usar diferentes valores de K según la región del espacio:

$$K(\mathbf{x}) = f(\text{densidad local, complejidad local})$$

K más grande en regiones de baja densidad o alta complejidad.

### Distance Metric Learning:

Aprender una métrica de distancia óptima en lugar de usar distancia euclídea estándar:

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{M} (\mathbf{x}_i - \mathbf{x}_j)}$$

donde  $\mathbf{M}$  es una matriz semi-definida positiva que se aprende de los datos (ej. Large Margin Nearest Neighbor - LMNN).

## Aplicaciones Principales

### Sistemas de Recomendación

Filtrado colaborativo, recomendación de productos, películas, música basado en usuarios similares

### Reconocimiento de Patrones

Clasificación de imágenes simples, reconocimiento de dígitos escritos a mano, reconocimiento de gestos

### Imputación de Datos

Rellenar valores faltantes usando promedio de K vecinos más similares, común en preprocesamiento

### Detección de Anomalías

Identificar outliers como puntos con vecinos lejanos, detección de fraude, intrusión en redes

### Compresión de Datos

Vector quantization, clustering, reducción de dimensionalidad mediante preservación de vecindades locales

### Diagnóstico Médico

Clasificación de enfermedades basada en síntomas y pruebas clínicas similares a casos históricos

## Métricas de Rendimiento

### Silhouette Score (para clustering/validación):

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

donde  $a(i)$  es la distancia promedio intra-cluster y  $b(i)$  es la distancia promedio al cluster más cercano. Rango:  $[-1, 1]$ , valores altos indican buena separación.

### Calinski-Harabasz Index:

$$CH = \frac{tr(\mathbf{B}_k)/(K - 1)}{tr(\mathbf{W}_k)/(n - K)}$$

Razón de varianza between-cluster a within-cluster. Valores altos indican clusters bien definidos.

## Ventajas y Limitaciones

### Ventajas:

- Extremadamente simple de entender e implementar (algoritmo intuitivo)
- Sin fase de entrenamiento (lazy learning) - adaptación instantánea a nuevos datos
- No paramétrico - no asume distribución de datos
- Naturalmente maneja clasificación multiclas
- Efectivo para fronteras de decisión irregulares y localmente complejas
- Útil como baseline para comparación con otros métodos
- Puede proporcionar estimaciones de probabilidad

### Limitaciones:

- **Maldición de dimensionalidad:** Rendimiento degradado severamente en alta dimensión ( $m > 20$ )
- **Costoso en predicción:**  $O(n \cdot m)$  por consulta sin estructuras de datos especiales
- **Memoria intensivo:** Debe almacenar todo el dataset de entrenamiento

- **Sensible a escala:** Requiere normalización obligatoria
- **Sensible a características irrelevantes:** Todas las características afectan la distancia por igual
- **Desbalance de clases:** Clases mayoritarias dominan en votación
- **Interpretabilidad limitada:** Difícil explicar por qué se hizo una predicción
- **Elección de K:** Requiere validación cruzada, no hay valor universal óptimo

## Complejidad Computacional

- **Entrenamiento:**  $O(1)$  - solo almacena datos
- **Predicción (búsqueda naive):**  $O(n \cdot m)$  - calcula distancia a todas las muestras
- **Predicción (con KD-Tree):**  $O(m \log n)$  en baja dimensión, degenera a  $O(n \cdot m)$  cuando  $m > 20$
- **Predicción (con Ball Tree):** Similar a KD-Tree, mejor en dimensión moderada
- **Predicción (con LSH):**  $O(m \cdot K)$  tiempo de consulta sublineal en alta dimensión (aproximado)
- **Espacio:**  $O(n \cdot m)$  - almacena dataset completo
- **Construcción de KD-Tree:**  $O(n \log n \cdot m)$

### Trade-offs Prácticos:

- Para  $n < 10^4$  y  $m < 20$ : KD-Tree o Ball Tree eficiente
- Para  $m > 20$  o  $n > 10^5$ : considerar métodos aproximados (LSH, Annoy)
- Para datasets grandes: considerar algoritmos alternativos (Random Forest, XGBoost) que escalan mejor
- Para tiempo real: pre-indexar con estructuras espaciales o usar approximate nearest neighbors