

# Naive Bayes (GaussianNB)

Naive Bayes Gaussiano asume que cada característica  $x_i$  sigue una distribución normal y que las características son independientes condicionalmente a cada clase.

Problema de optimización:

$$\hat{y} = \underset{c_k}{\operatorname{argmax}} p(c_k) \prod_{i=1}^d p(x_i | c_k)$$

donde  $p(x_i | c_k)$  es la densidad de una normal  $\mathcal{N}(\mu_{ki}, \sigma^2_{ki})$  estimado como máxima verosimilitud.

- Calcular probabilidades y luego usar Bayes para clasificar.

# SGDClassifier

SGDClassifier entrena un modelo lineal  $f(x) = w^T x + b$  utilizando descenso por gradiente estocástico. Puede emplear distintas funciones de pérdida (log-loss, hinge, ...)

## Problema de optimización

$$\min_{w, b} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i(w^T x_i + b)}) + \lambda \|w\|^2$$

- No es un modelo en si por si solo, sino una forma de entrenar modelos lineales.

# Logistic Regressor

Logistic Regressor ajusta un modelo lineal de la forma  $f(x) = w^T x + b$  y modela la probabilidad mediante la función sigmoide:

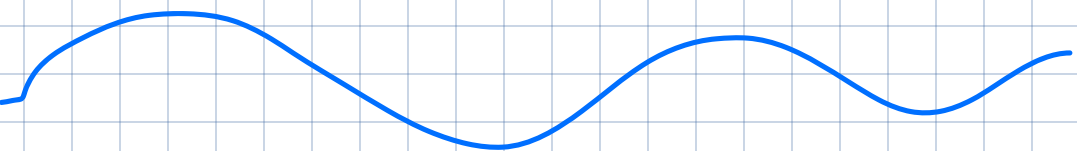
$$p(y=1|x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

## Problema de optimización

$$\min_{w, b} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i(w^T x_i + b)}) + \lambda \|w\|^2$$

- No es una regresión en el sentido clásico, sino un clasificador probabilístico. En vez de predecir

directamente una clase, predecir la probabilidad de  $y=1$  dado  $x$ , usando la función sigmoide.



## Linear Discriminant Analysis

LDA modela cada clase  $C_k$  como una distribución normal multivariada con la misma matriz de covarianza  $\Sigma$ , pero distinta media  $\mu_k$ . La regla de decisión es:

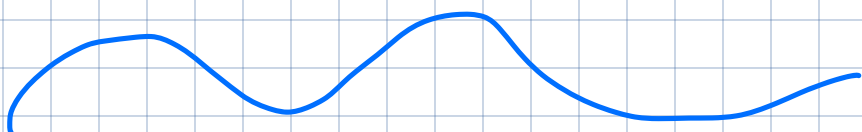
$$\hat{y} = \underset{k}{\operatorname{argmax}} \left( x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k \right)$$

donde  $\pi_k$  es la probabilidad a

priori de la clase  $c_k$ .

## Problema de optimización

No se resuelve una pérdida explícita, sino que se estiman los parámetros  $\mu_k$ ,  $\Sigma$  y  $\pi_k$  por máxima verosimilitud.



## K Neighbors Classifier.

K Neighbors Classifier asigna una clase a un punto nuevo  $x$  en función de las clases mayoritarias de sus  $K$  vecinos más cercanos en el conjunto de entrenamiento según una métrica de distancia.

# Problema de optimización

No realiza entrenamiento ni optimización explícita. Es un método basado en memoria (lazy learning).

$$\hat{y} = \text{mayoría entre } \left\{ \begin{array}{l} y_i \text{ de los } K \text{ vecinos} \\ \text{más cercanos a } x \end{array} \right\}$$



## Linear SVC

Linear SVC ajusta un modelo lineal de clasificación utilizando una función de pérdida tipo hinge (bisagra), propia de los máquimas de vectores de soporte (SVM):

$$f(x) = w^T x + b$$

# Problema de optimización

$$\min_{w, b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i (w^T x_i + b))$$

donde  $C$  controla la penalización por errores.

## SVC (Support Vector Classifier)

SVC implementa una máquina de vectores de soporte que puede usar funciones kernel para separar clases no lineales. El modelo es:

$$f(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b$$

donde  $K$  es una función kernel  
y  $\alpha_i$  son coeficientes aprendidos.

Problema de optimización (dual):

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

sujeito a  $0 \leq \alpha_i \leq C$ ,  $\sum_i \alpha_i y_i = 0$ .

## Random Forest Classifier

Random Forest Classifier es un ensamble de árboles de decisión. Cada árbol se entrena con una muestra aleatoria del conjunto de datos (con reemplazo) y selecciona una



submuestras aleatoria al dividir nodos.

Predicción: se hace por votación mayoritaria entre todos los árboles.

$$\hat{y} = \text{majority} \{T_1(x), T_2(x), \dots, T_M(x)\}$$

Problema de optimización:

Cada árbol se entrena para minimizar una medida de impureza

(Gini o entropía) en cada división.

No hay función global que se optimice para todo el bosque.