

## PCA (Principal component analysis)

El objetivo de PCA es reducir la dimensionalidad de un conjunto de datos, proyectándolo sobre un nuevo sistema de coordenadas (componentes principales) de manera que:

- se refiera a mayor cantidad posible de varianza de los datos originales
- se obtengan nuevas variables no correlacionadas entre sí
- se facilite la visualización y el procesamiento de los datos

### Problema de optimización

Se deben encontrar una proyección lineal hacia un subespacio de menor dimensión  $\mathcal{L}$  tal que la varianza proyectada sea máxima dado un conjunto de datos  $X \in \mathbb{R}^{n \times d} \rightarrow$  características

muestras

Norma

Entonces se quiere encontrar un conjunto de vectores orthonormales  $w_1, \dots, w_k \in \mathbb{R}^d$  que resuelvan

$$\max_{w_i \in \mathbb{R}^d, \|w_i\|=1} \text{Var}(X w_i) = w_i^\top S w_i$$

sujeto a  $w_i^\top w_j = 0$  para  $i \neq j$  donde  $S = \frac{1}{n} X^\top X$   
es la matriz de covarianza

El problema se resuelve encontrando los autovectores  
correspondientes a los  $k$  mayores autovalores de  $S$ , correspondientes  
a los vectores  $w_1, \dots, w_k$ . Los autovalores indican cuanta  
varianza explica cada componente

En resumen: El modelo PCA busca maximizar la varianza neta  
dejando proyección (real orthonormal) y se resuelve mediante  
a través de la descomposiciónpectral de la matriz de  
covarianza

UMAP (Uniform Manifold Approximation and Projection)  
Es una técnica de reducción de dimensión no lineal basada en conceptos de topología algebraica y teoría 3D geométrica.

Su objetivo es: → Preservar la estructura local y global de los datos

- reducir la dimensionalidad de manejar la "morfología" en el espacio entre los datos
- ser útil tanto para visualización como para procesamiento de aprendizaje automático

UMAP asume que los datos están distribuidos sobre una variedad de baja dimensión y redondean singularidades.

1. Construcción de un grafo fuzzy simplicial de alta dimensional: se mide la similitud entre puntos usando una función de afinidad basada en distancias (máximos cercanos) y se construye una distribución de probabilidad local alrededor de cada punto

2. Construcción de un grafo fuzzy simplicial en baja dimensión: se calculan los puntos en un espacio de menor dimensión y se tratan de preservar la similitud entre puntos; aparece una representación más embobinada de manifiestos (los relaciones locales)

## Problema de optimización

UMAP define una función de costo basada en la divergencia cruzada entre + entropía entre los grados frey de alta y baja dimensión. La función obviamente minimizar es:

$$d = \sum_{(i,j)} (1 - w_{ij}) \cdot \log(q_{ij}) + (1 - w_{ij}) \cdot \log(1 - q_{ij})$$

donde:  $w_{ij}$  es la probabilidad entre puntos  $i$  y  $j$  en alta dimensión

$q_{ij}$ : probabilidad (similaridad) entre los puntos embobeados  $i$  y  $j$  de baja dimensión

$$q_{ij} = (1 + \|y_i - y_j\|^{2b})^{-1}$$

La función se optimiza con descenso estocástico del gradiente (SGD) o mini batch SGD

En resumen UMAP realiza una reducción no lineal de dimensión preservando tanto la estructura local como global, resolviendo un problema de optimización de divergencia cruzada entre distribuciones de proximidad de puntos

## Naïve Bayes Gaussiano

Se basa en el teorema de Bayes con el supuesto de independencia condicional entre las características dado a clase. Se asume que las variables predictivas tienen una distribución gaussiana.

Dado un vector de características  $\mathbf{x}$  y una clase  $y$  el modelo aplica el teorema de Bayes

$$P(y|x) = \frac{P(y) \cdot P(x|y)}{P(x)}$$

$$\hat{y} = \operatorname{argmax}_y [P(y) \cdot P(x|y)]$$

cte en todas las clases ↑

Supuesto Naïve (Independencia condicional)  $P(x|y) = \prod_{j=1}^d P(x_j|y)$

y cada  $x_j|y$  se modela como distribución normal

$$P(x_j|y) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(x_j - \mu_{y,j})^2}{2\sigma_j^2}\right)$$

GNB aprende los parámetros conjuntamente utilizando el algoritmo de máxima verosimilitud. Para cada clase  $y$ :

$$\text{prior } P(y) = \frac{n_y}{n}$$

$$\text{media } \hat{\mu}_{y,j} = \frac{1}{n_y} \sum_{i:y_i=j} x_{ij}$$

$$\text{varianza } \hat{\sigma}_{y,j}^2 = \frac{1}{n_y} \sum_{i:y_i=j} (x_{ij} - \hat{\mu}_{y,j})^2$$

# SGDClassifier

## Problema de optimización

Dado un conjunto de entrenamiento  $\{(x_i, y_i)\}_{i=1}^n$ , donde  $x_i \in \mathbb{R}^d$  y  $y_i \in \{-1, 1\}$ :

$$\min_{w,b} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, w^T x_i + b) + \alpha R(w)$$

Donde:

- $\mathcal{L}(y, \hat{y}) \rightarrow$  Función de pérdida convexa
- $R(w) \rightarrow$  Término de regularización como:
  - L2:  $\frac{1}{2} \|w\|_2^2$
  - L1:  $\|w\|_1$
  - Elastic Net: L1 y L2
- $\alpha > 0 \rightarrow$  Parámetro de regularización

## Modelo

SGDClassifier entrena un modelo lineal de la forma:

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

$W \rightarrow$  Vectores de pesos entrenados  
 $b \rightarrow$  Sesgo

La predicción se realiza con base en el signo del margen lineal.

## Logistic Regression

### Modelo

Dado un conjunto de entrenamiento  $\{(x_i, y_i)\}_{i=1}^n$ , con:

$$- x_i \in \mathbb{R}^d$$

$$- y_i \in \{0, 1\}$$

$$P(y_i = 1 | x_i) = \sigma(w^T x_i + b)$$

donde  $\sigma(z)$  es la función sigmoidal:

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

La predicción se hace con:

$$\hat{y}_i = \begin{cases} 1 & \text{Si } \sigma(w^T x_i + b) \geq 0.5 \\ 0 & \text{en otros casos} \end{cases}$$

## Problema de optimización

$$\min_{w,b} \frac{1}{N} \sum_{i=1}^n [-y_i \log(\hat{p}_i) - (1-y_i) \log(1-\hat{p}_i)] + \alpha R(w)$$

dónde:

$$\hat{p}_i = \sigma(w^T x_i + b)$$

$R(w) \rightarrow$  Regularización

## Linear discrimination Analysis

Se asume que cada clase  $y \in \{1, 2, \dots, K\}$  tiene una distribución gaussiana con:

$$x|y=k \sim N(\mu_k, \Sigma)$$

Se asume:

- Covarianzas iguales  $\Sigma_1 = \dots = \Sigma_K = \Sigma$

- Vectores de medias por clase  $\mu_k$

- Priori de clase  $\pi_k = P(y=k)$

La regla de decisión se basa en la función discriminante lineal:

$$\mathcal{L}(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

Se asigna la clase:

$$\hat{y} = \arg \max_k \mathcal{L}_k(x)$$

LDA proyecta los datos en un espacio lineal donde la separación entre clases es óptima según la razón de fisher:

$$J(W) = \frac{W^T S_B W}{W^T S_W W}$$

donde:

$S_B$  = Matriz de dispersión entre clases

$S_W$  = Matriz de dispersión dentro de clases

### Problema de optimización:

Con reducción de dimensionalidad (2 clases):

$$\max_W \frac{(W^T (\mu_1 - \mu_2))^2}{W^T S_W W}$$

Para multiclas se proyectan los datos sobre un sub-espacio

de dimensión  $\leq K-1$  y se encuentran los vectores propios más relevantes de  $S_W^{-1} S_B$

## K Neighbors Classifier

Dado un conjunto de datos  $\{(x_i, y_i)\}_{i=1}^n$

- $x_i \in \mathbb{R}^d$

- $y_i \in \{1, 2, \dots, K\}$

$$\hat{y} = \text{modo}(\{y_j \in N_K(x)\})$$

donde

- $N(x) \rightarrow$  Conjunto de los  $K$  vecinos más cercanos a  $x$  según una métrica de distancia
- $\text{modo}() \rightarrow$  Clase más frecuente entre esos vecinos

Distancia típica

Para 2 puntos  $x$  y  $z$ , la distancia euclídea es:

$$d(x, z) = \|x - z\|_2 = \sqrt{\sum_{i=1}^d (x_i - z_i)^2}$$

## 1) Modelo y problema de optimización de

### - SVC (Support Vector classifier):

SVC busca encontrar el hiperplano que maximiza el margen entre 2 clases. Es decir, la distancia entre los puntos de borde (soporte) y el hiperplano.

Función objetivo (caso lineal), margen blando:

$$\min_{w, b, \epsilon} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \epsilon_i$$

sujeto a:

$$y_i(w^T x_i + b) \geq 1 - \epsilon_i, \quad \epsilon_i \geq 0$$

w: Vector normal al hiperplano.

b: sesgo.

$\epsilon_i$ : Variables de holgura (permiten errores).

C: Hiperparámetro que controla el compromiso entre margen y error.

$y_i \in [-1, 1]$  etiqueta de la clase.

$x_i$ : punto de entrenamiento.

Para problemas NO lineales: Se utiliza un Kernel (ej: RBF, polinomial), lo que transforma el problema en un espacio de características de mayor dimensión.

### - Random Forest classifier:

Es un ensamble de muchos árboles de decisión, cada uno entrenado con una muestra diferente del conjunto de datos (bagging). La predicción se realiza por votación mayoritaria.

Algoritmo:

1). Para cada árbol toma una muestra aleatoria con reemplazo (bootstrap) del dataset. Constituye un árbol de decisión sin poda. En cada división del Árbol, se selecciona un subconjunto aleatorio de características para decidir la mejor partición.

2). Para predecir se pasa la entrada por todos los árboles, cada uno da una predicción. Se elige la clase por votación mayoritaria.

No tiene una única función objetivo explícita, pero cada árbol minimiza una medida de impureza como:

$$\text{Gini: } G = 1 - \sum_k p_k^2$$

$$\text{Entropía: } H = - \sum_k p_k \log p_k$$

P: Número de características del dataset (dimensión del vector de entrada).  
K: Número de clases (output categories).

### - Gaussian Process classifier (GPC):

Es un enfoque no paramétrico y probabilístico. Define una distribución sobre funciones y utiliza un proceso Gaussiano como prior sobre esas funciones.

Para clasificación:

Se define una función latente  $f(x) \sim GP(0, K(x, x'))$

Luego se pasa por una función sigmoidal para obtener una probabilidad:

$$P(y=1|x) = \sigma(f(x)) = \frac{1}{1 + e^{-f(x)}}$$

Problema de optimización: Usar el log-likelihood es complicado.

Como el likelihood no es gaussiano, se usa una aproximación (ej: Laplace o Expectation Propagation).

$$\max \log P(y|F) - \frac{1}{2} F^T K^{-1} F$$

K: matriz de covarianza constituida con el kernel  $K(x_i, x_j)$ .  
F: valores de la función latente por cada dato.  
y: etiquetas observadas

La ventaja que tiene el GPC es que da incertidumbre sobre las predicciones.

## - clasificador basado en Deep Learning (ej: CNN)

Una Red Neuronal convolucional (CNN) es ideal para imágenes. Aprende filtros convolucionales que capturan patrones especiales (bordes, texturas, formas).

Estructura típica:

- Capas convolucionales: aplican filtros que se aprenden
- Funciones de activación: ReLU.
- Capas de pooling: reducción de dimensionalidad.
- Capas Fully connected: para clasificación

Función objetivo: (clasificación multiclase).

$$\min_{\theta} L(y, \hat{y}) = - \sum_{i=1}^n \sum_{k=1}^K y_{ik} \log(\hat{y}_{ik})$$

$\hat{y}_{ik}$ : salida softmax de la red para clase  $k$

$y_{ik}$ : etiqueta one-hot.

Se entrena con:

- Backpropagation
- Optimizadores como SGD, Adam.
- Regularización: Dropout, batch normalization, etc.