

## Parcial 2.

Nombres: Wilmer Sebastián Pérez Cuastumal

Juan Jerónimo Castaño Rivera

Oscar Andrés Gutiérrez Estepa

- a) Consulte y presente el modelo y problema de optimización de los siguientes modelos de aprendizaje de máquina.

→ Análisis de Componentes Principales: PCA  $\hat{X} \approx ZW^T$

Dado un conjunto de datos  $X \in \mathbb{R}^{n \times d}$  (donde  $n = \text{muestras}$  y  $d = \text{dimensionalidad}$  original), PCA busca encontrar una transformación lineal que proyecte los datos a un espacio de menor dimensión donde  $Z \in \mathbb{R}^{n \times k}$  son los datos transformados.  $W \in \mathbb{R}^{d \times k}$  (con  $k < d$ )

$$Z = XW,$$

- Los componentes principales son combinaciones lineales de la varianza originales
- El primer componente principal captura la máxima varianza, el segundo (ortogonal al primero) captura la siguiente mayor varianza, y así sucesivamente.

El objetivo de PCA es encontrar los vectores de proyección  $W$  que maximicen la varianza de los datos proyectados.

$$\therefore W^* = \arg \min_W \|X_n - Z_n W^T\|$$

$\downarrow$        $\underbrace{\quad}_{\text{datos originales}} \quad \underbrace{\quad}_{\text{los datos generalizados}}$

$$= \min_W E_x \{ \|X_n - Z_n W^T\|_2^2 \} = E_x \{ \langle X_n - Z_n W^T, X_n - Z_n W^T \rangle \}$$

$$= E_x \{ X_n X_n^T - 2 X_n (Z_n W^T)^T + Z_n W^T (Z_n W^T)^T \}$$

$$= E_x \{ X_n X_n^T - 2 X_n W W^T X_n^T + X_n W W^T W W^T X_n^T \}$$

$$= E_x \{ X_n X_n^T - X_n W W^T X_n^T \}$$

constante

$$= \min_W - E_x \{ X_n W W^T X_n^T \} = \min_W - E_x \{ Z_n Z_n^T \}$$

$$= -E_x \{ Z_n^T Z_n \} = -E_x \{ W^T X_n^T X_n W \} = -W^T \underbrace{E_x \{ X_n^T X_n \}}_{R_{xx} \in \mathbb{R}^{p \times p}} W$$

$R_{xx} \in \mathbb{R}^{p \times p}$   
Covarianza

$$\therefore = \min_W - W^T \sum_x W = \max_W W^T \sum_x W$$

$$L(W, \lambda) = W^T \sum_x W - \lambda (W^T \sum_x W - 1)$$

Solución mediante multiplicadores de Lagrange para una componente

$$\frac{\partial L(W, \lambda)}{\partial W} = 2 \sum_x W - 2 \lambda W = 0$$

$$\boxed{\sum_x W = \lambda W}$$

## → Mapeo uniforme de aproximación y proyección: UMAP

UMAP es una técnica de reducción de dimensión que modela el espacio de alta dimensión como estructura topológica difusa y la optimiza en el espacio de baja dimensión preservando distancias globales y locales.

∴ Representación en Alta dimensión:

$$P_{nn} = \exp\left(-\frac{\|X_n - X_n'\| - P_n}{\zeta_n}\right) \quad \begin{cases} \zeta_n : \text{distancia de vecino más cercano } X_n \\ P_n : \text{se determina un número fijo de vecinos} \end{cases}$$

La relación difusa se vuelve simétrica como:

$$P_{nn} = P_{nn'} + P_{nn'} - P_{nn} P_{nn'}$$

∴ Proyección de baja dimensión.

Las relaciones en el espacio de baja dimensión se fijan mediante una distribución t-Student de cola pesada.

$$q_{ij} = (1 + \alpha \|z_n - z_n'\|^{2b})^{-1} : \alpha, b \text{ como parámetros de la distribución.}$$

Generalmente se fijan en 1.

El objetivo es minimizar la discrepancia entre las distribuciones de similitud en alta y baja dimensión, medida mediante la entropía cruzada

$$C(p, q) = \sum_{n \neq n'} (-P_{nn'} \log(q_{nn'}) - (1 - P_{nn'}) \log(1 - q_{nn'}))$$

La función de pérdida, combina atracción (para puntos cercanos en alta dimensión) y repulsión (para puntos lejanos)

## → Naïve Bayes Gaussiano: GNB

Es un algoritmo de aprendizaje supervisado basado en el teorema de Bayes, que asume que las características siguen una distribución normal y son independientes entre sí. Es ampliamente utilizado en clasificación, especialmente cuando los datos son continuos.

$$\underbrace{P(y|x)}_{\substack{\text{Independencia} \\ \text{Condicional}}} = \frac{P(x|y) \cdot P(y)}{P(x)} \quad P(x|y) = \prod_{i=1}^k P(x_i|y)$$

Distribución Gaussiana →  $P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{\|x_i - \mu_y\|^2}{2\sigma_y^2}\right)$

Para características continuas

El objetivo es maximizar la probabilidad posterior (MAP) para clasificar un nuevo dato  $x$

$$\hat{y} = \arg \max_y P(y) \prod_{i=1}^k P(x_i|y) \quad : \quad \mu_y = \frac{1}{n_y} \sum_{x \in y} x_i, \quad \sigma_y^2 = \frac{1}{n_y} \sum_{x \in y} (x_i - \mu_y)^2$$

Se clasifica usando MAP // parcial 1.

## → Clasificadores con gradiente descendente estocástico: SGDClassifier // scikit-learn

Modelo de aprendizaje supervisado lineal que utiliza el algoritmo de gradiente descendente estocástico (SGD) para optimizar una función de pérdida.

Para un dato de entrada  $\mathbf{x} \in \mathbb{R}^d$ , la predicción es:

$$\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + b) : \text{Clasificación binaria}$$

$$y = \arg \max_k (\mathbf{w}_k^\top \mathbf{x} + b_k) : \text{Para multiclase, one vs all}$$

• Problema de optimización

$$J(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n L(\mathbf{w}, b; \mathbf{x}_i, y_i) + \alpha R(\mathbf{w})$$

funciones de pérdida

Loss

funciones de regularización

• Loss: Mide cuán mal está la predicción del modelo. El SGDClassifier puede usar diferentes funciones.

- Hinge loss (para SVM Lineal)

$$L = \max(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b))$$

- Log loss (para regresión logística):

$$L = -\log \left( \frac{1}{1 + e^{-y_i(\mathbf{w}^\top \mathbf{x}_i + b)}} \right)$$

Regularización: Evita el sobreajuste penalizando pesos grandes. Las más comunes son:

- Regularización  $L_2$  (Ridge):

$$R(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_2^2 = \frac{1}{2} (w_1^2 + w_2^2 + \dots + w_d^2) : \text{Penaliza pesos grandes de forma cuadrática.}$$

- Regularización  $L_1$  (Lasso):

$$R(\mathbf{w}) = \|\mathbf{w}\|_1 = |w_1| + |w_2| + \dots + |w_d|$$

• SGD actualiza  $\mathbf{w}$  y  $b$  usando el gradiente (Derivada) de la pérdida.

$$\nabla_L$$

$$\mathbf{w} \leftarrow \mathbf{w} - n (\nabla_{\mathbf{w}} L + \alpha \nabla_{\mathbf{w}} R)$$

## → Regresión logística:

Modelo de clasificación supervisada que estima la probabilidad de que una muestra pertenezca a una clase (binaria o multiclase), la regresión logística siempre utiliza la función log-loss y se optimiza típicamente con desenso de gradiente o métodos de Newton-Raphson.

$$P(Y=1|X) = \sigma(W^T W + b) = \frac{1}{1 + e^{-(W^T W + b)}} : \text{la probabilidad de que pertenezca a la clase 1.}$$

Para  $n$  muestras, la perdida es:

$$J(W, b) = -\frac{1}{n} \sum_{i=1}^n [y_i \log(P(Y_i|X_i)) + (1-y_i) \log(1-P(Y_i|X_i))]$$

$\hookrightarrow \underset{W, b}{\text{Min}} J(W, b) + \alpha R(W)$  : se busca  $W, b$  que minimicen  $J(W, b)$

$$\nabla_W J(W, b) = \frac{1}{n} \sum_{i=1}^n (P(Y_i|X_i) - y_i) X_i$$

$$\frac{\partial J(W, b)}{\partial b} = \frac{1}{n} \sum_{i=1}^n (P(Y_i|X_i) - y_i)$$

## → Análisis Discriminante lineal LDA

Modelo de clasificación supervisada y reducción de dimensionalidad que presupone que los datos siguen una distribución gaussiana. Maximiza la separación entre clases mientras minimiza la varianza introclase.

$\hat{Y} = \arg \max_k f_k(X)$  : Para cada clase  $k$ , LDA calcula una función discriminante lineal

$$\hookrightarrow f_k(X) = X^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(Y=k)$$

Problema de optimización:

$$J(W) = \underset{W}{\operatorname{Max}} \frac{W^T S_B W}{W^T S_w W} \quad \left. \begin{array}{l} S_B \sim \text{Matriz de dispersión entre clases} \\ S_w \sim \text{Matriz de dispersión introclase} \end{array} \right.$$

$$\frac{\partial J(W)}{\partial W} = 2S_B W (W^T S_w W) - 2S_w W (W^T S_B W) = 0$$

$$S_B W = \left( \frac{W^T S_B W}{W^T S_w W} \right) S_w W$$

$\rightarrow$  Escalar  $\rightarrow J(W)$

$\therefore$  El mayor autovalor ( $\lambda_{\max}$ ) corresponde a la dirección  $W$  que maximiza la separación entre clases.

Matrices de dispersión

$$S_B W = S_w W \rightarrow S_w^{-1} S_B W = \lambda W$$

AutoVector

AutoValor

$$\hookrightarrow \left\{ \begin{array}{l} S_B = \sum_{k=1}^K N_k (\mu_k - \mu)(\mu_k - \mu)^T : N_k \rightarrow \text{Número de muestras} \\ \mu \rightarrow \text{Media global} \end{array} \right.$$

$$\left\{ \begin{array}{l} S_w = \sum_{k=1}^K \sum_{x_i \in C_k} (x_i - \mu_k)(x_i - \mu_k)^T : C_k \rightarrow \text{Conjunto de muestras de la clase } k \end{array} \right.$$

## → K Neighbors Classifier ; KNN

Algoritmo de aprendizaje supervisado no paramétrico utilizado para clasificación (y regresión). A diferencia de modelos como SVM o regresión logística, KNN no aprende una función de decisión durante el entrenamiento, sino que realiza predicciones basadas en la proximidad de los datos en el espacio.

Dado un conjunto de entrenamiento:

$$D = \{(x_i, y_i)\}_{i=1}^n, \quad x_i \in \mathbb{R}^d, \quad y_i \in \{c_1, \dots, c_K\}$$

Para una nueva muestra  $x_{new}$  se define la función de predicción como

$$\hat{y} = \text{moda}(\{y_i | x_i \in \underbrace{N_K(x_{new})}_{\text{Conjunto de los } K \text{ vecinos más cercanos a } x_{new}}\})$$

KNN no optimiza una función de pérdida tradicional. En su lugar, minimiza implicitamente el error de clasificación local mediante: cálculo de distancias, selección de vecinos y votación mayoritaria.

### → Support Vector Classifier: SVC

Es un algoritmo de aprendizaje supervisado para clasificación binaria y multiclase, basado en el concepto de Máximo Margen en Máquinas de vectores de soporte (SVM). Su objetivo es encontrar un hiperplano óptimo que separe las clases con el mayor margen posible.

$$D = \{(x_i, y_i)\}_{i=1}^n, \quad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, +1\}$$

:  $f(x) = \text{Sign}(W^T x + b) \rightarrow$  con  $W$  y  $b$  se busca el hiperplano de mayor separación

- Problema de optimización (linealmente separable)

$$\min_{w, b} \frac{1}{2} \|w\|^2 : \text{suje}to a \quad y_i(w^T x_i + b) \geq 1 \quad \forall i$$

Garantizan que todos los puntos estén correctamente clasificados y fuera del margen

- Problema de optimización, (No linealmente separable)

$$\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i : \text{suje}to a \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \wedge \quad \xi_i \geq 0 \quad \forall i$$

↓ controla el comportamiento entre margen y error.

↓ Variable de holgura que permite errores

- El problema puede expresarse en su forma dual para facilitar la solución y permitir el uso de kernels.

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

### → Random Forest Classifier

Modelo de aprendizaje supervisado basado en ensamblado de árboles de decisión, que combinan múltiples árboles para mejorar la precisión y reducir el sobreajuste. Es ampliamente utilizado por su robustez y rapidez para manejar datos complejos sin preprocesamiento extensivo.

$$D = \{(x_i, y_i)\}_{i=1}^n \rightarrow \text{conjunto de entrenamiento} : \text{construye } T \text{ árboles } \{h_t(x)\}_{t=1}^T \text{ así:}$$

Pasos:

- Muestras aleatorias: se crean  $n$  subconjuntos de entrenamiento mediante bootstrap de (conjunto con reemplazo)
- Construcción de árboles: para cada subconjunto, en cada nodo se seleccionan características al azar. Se elige la mejor división. (Ej. Maximizando Gini o Entropía). El árbol crece hasta que todas las hojas sean puros.

$$\text{Gini} = 1 - \sum_{k=1}^K P_k^2 : P_k \rightarrow \text{proporción de muestras de la clase } k \text{ en el nodo}$$

$$\text{Entropía} = - \sum_{k=1}^K P_k \log(P_k)$$

- Predictión: clasificación  $\rightarrow$  votación mayoritaria entre todos los árboles

$$y = \text{moda}\{h_1(x), h_2(x), \dots, h_T(x)\}$$

Probabilidad  $\rightarrow$  promedio de las probabilidades de clase de los árboles.

$$P(y=c|x) = \frac{1}{T} \sum_{t=1}^T I[h_t(x) = c]$$

### → Gaussian Process Classifier GPC

Modelo probabilístico no paramétrico para clasificación, basado en procesos gaussianos (GP). Es una distribución sobre funciones, definido por una media y covarianza (kernel). En clasificación binaria se modela una función latente f(x)

$$f(x) \sim GP(0, k(x, x'))$$

La probabilidad de clase se obtiene aplicando una función sigmoidal a la salida del GP.

$$P(y=1|x) = \Phi(f(x)) : \Phi(\cdot) \text{ puede ser la función logística o la función sigmoidal}$$

$$\therefore D = \{(x_i, y_i)\}_{i=1}^n ; y_i \in \{0, 1\} \quad D \rightarrow \text{conjunto de entrenamiento}$$

El objetivo es calcular la distribución posterior sobre la función latente f y usarla para predecir:

$$P(Y_*|X_*, X, Y) = \int P(Y_*|f_*) \cdot P(f_*|X_*, X, Y) df_* : X_* \rightarrow \text{nuevo punto}$$

Esto no es resoluble analíticamente, por lo que se usa una aproximación variacional o laplace approximation para la inferencia.

$f_* \sim N(\mu_*, \sigma_*^2) \rightarrow$  Dada una nueva entrada  $X_*$ , se calcula la distribución predictiva sobre  $f_*$  y se obtiene la probabilidad de clase.

$$P(Y_* = 1 | X_*) = \int \Phi(f_*) \cdot N(f_* | \mu_*, \sigma_*^2) df_*$$

## → Clasificadores basados en Deep Learning

El objetivo es aprender una función de decisión no lineal compleja mediante una red neuronal profunda, optimizando una función de pérdida basada en probabilidad y regularización.

∴ Arquitecturas principales para clasificación.

### • Redes Neuronales feedforward MLP:

Estructura: capas densamente conectadas (Input → hidden → output)

$$z_j^{(l)} = \sum_{i=1}^n w_{ij}^{(l)} x_i + b_j^{(l)}$$

↓      ↓      ↓      ↗  
 Hidden Layer      Entrada      Sesgo de la neurona j  
 Pesos de la conexión

$$a_j^{(l)} = \sigma(z_j^{(l)})$$

función de activación

$$a_j^{(l)} \rightarrow \text{activación de capa } l$$

Output layer:

$$\left\{ \begin{array}{l} \text{clase binaria (1 neurona con Sigmoid)} \\ q = \sigma(w^{(l)} a^{(l-1)} + b^{(l)}) , \quad \sigma(z) = \frac{1}{1+e^{-z}} \\ \text{clase Multiclas (K neuronas con Softmax)} \\ \hat{q}_k = \frac{e^{z_k^{(l)}}}{\sum_{j=1}^k e^{z_j^{(l)}}} \quad \text{donde} \quad z^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)} \end{array} \right.$$