

Ejercicios TAM

- Clasificación multiclase con datos punto en Python. A continuación se compara el rendimiento de los métodos utilizando validación cruzada.

Ejercicios

Consulte y presente el modelo y problema de optimización de los siguientes clasificadores:

1) Naive Bayes Gaussian NB

Es un modelo de clasificación que se usa para predecir una clase (ejm: "sí" o "no", "spam" o "no-spam") a partir de ciertos datos de entrada (como edad, altura, temperatura, etc).

Este modelo se basa en probabilidades. Usa el Teorema de Bayes para calcular la probabilidad de que un ejemplo pertenezca a una clase, suponiendo que los datos siguen una distribución Gaussiana (Normal) y que cada característica es independiente de las demás.

• Modelo matemático

Para predecir la clase más probable y , el modelo usa el Teorema de Bayes:

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

$P(y|x)$ → Probabilidad de la clase y dado los datos x .

$P(y)$ → Probabilidad de la clase y . (antes de ver los datos)

$P(x|y)$ → Probabilidad de ver los datos si la clase fuera y .

$P(x)$ → Probabilidad de ver esos datos (No depende de la clase).

El modelo elige la clase que tiene la mayor probabilidad. Es decir:

$$\hat{y} = \arg \max_y P(y) \cdot P(x|y)$$

NOTA: Para que sea más fácil calcular $P(x|y)$, se asume que todas las variables son independientes entre sí. (Esto se llama "ingenuo" o naive). Entonces:

$$P(x|y) = \prod_{i=1}^n P(x_i|y)$$

Como este clasificador es Gaussian NB, también se supone que cada $x_i|y$ sigue una distribución normal (Gaussiana):

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(x_i - \mu_{iy})^2}{2\sigma_{iy}^2}\right)$$

μ_{iy} : media de la característica x_i para la clase y

σ_{iy}^2 : varianza de la característica x_i para la clase y .

• Problema de optimización.

A diferencia de otros modelos como la regresión logística, aquí no se optimiza una función complicada. En su lugar, se hace una tarea muy concreta:

Estimar los siguientes valores a partir de los datos:

1. La media de cada característica para cada clase:

$$\bar{y}_{ij} = \frac{1}{N_y} \sum_{j:y_j=y_i} x_{ij}$$

2. Las varianzas de cada característica para cada clase:

$$\sigma_{ij}^2 = \frac{1}{N_y} \sum_{j:y_j=y_i} (x_{ij} - \bar{y}_{ij})^2$$

3. La probabilidad de cada clase (frecuencias en los datos).

$$p(y) = \frac{N_y}{N}$$

Conclusión del problema de optimización: Estimar las medias y varianzas y las probabilidades de clase a partir de los datos. No se usa una función de pérdida compleja, sino fórmulas cerradas.

2) SGD classifier

Es un modelo de clasificación que aprende los parámetros a poco a poco, usando un método llamado **Descenso de Gradiente Estocástico**. Este clasificador puede representar distintos modelos como:

• Regresión logística (para clasificación binaria)

• SVM (Máquinas de Vector de Soporte)

• Perceptrón

Todo depende del tipo de función de pérdida que se elija.

Lo más importante es que este clasificador optimiza una función objetivo iterativamente, actualizando sus parámetros para minimizar errores.

Modelo

Intenta aprender una función que separe las clases usando una combinación lineal de las variables de entrada.

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

$w \rightarrow$ Vector de pesos (parámetros que el modelo aprende)

$b \rightarrow$ sesgo (bias)

La predicción se hace con: $\hat{y} = \text{sign}(f(x)) = \text{sign}(w^T x + b)$

Problema de optimización

Aquí es donde entra el "descenso de gradiente estocástico": el objetivo es encontrar los mejores parámetros w y b para que el modelo cometa la menor cantidad de errores.

Esto se logra minimizando una función de pérdida (coste), que depende del tipo de modelo.

$$\text{Función objetivo general: } \min_{w, b} \frac{1}{N} \sum_{i=1}^N L(y_i, w^T x_i + b) + \lambda R(w)$$

L → función de pérdida para un ejemplo

$R(w)$ → Regularización (opcional), para evitar sobreajuste

λ → parámetro que controla la importancia de la regularización

¿Qué significa "descenso de gradiente estocástico"?

En vez de calcular el promedio sobre todos los datos (como en el gradiente clásico), el modelo actualiza los parámetros

usando solo un dato a la vez (o unos pocos), lo que hace mucho más rápido para grandes conjuntos de datos.

3) Logistic Regression

Es un modelo de clasificación binaria (aunque se puede extender a múltiples clases) que sirve para predecir una categoría (por ejemplo: "spam" o "no spam") a partir de un conjunto de datos de entrada (como edad, altura, puntaje, etc.).

Aunque se llama "regresión", en realidad su salida no es un # cualquiera, sino una probabilidad de que un ejemplo pertenezca a cierta clase. Y al final, predice esa clase con mayor probabilidad.

(Modelo binomial)

Suponer que se tiene un vector de características $x \in \mathbb{R}^n$, y se quiere predecir una clase $y \in \{0, 1\}$.

La regresión logística modela la probabilidad de la clase 1 con la siguiente función:

$$P(y=1|x) = \sigma(w^T x + b)$$

w → Vector de pesos (parámetros que el modelo aprende)

b → sesgo (bias)

$\sigma(z)$ → función sigmoidal, que convierte cualquier número entre 0 y 1 en probabilidad entre 0 y 1.

Si $\sigma(z) = \frac{1}{1+e^{-z}}$ entonces: $P(y=1|x) = \frac{1}{1+e^{-(w^T x + b)}}$

La predicción final es:

$$\hat{y} = \begin{cases} 1 & \text{si } P(y=1|x) \geq 0.5 \\ 0 & \text{si } P(y=1|x) < 0.5 \end{cases}$$

Problema de optimización

El objetivo es encontrar los valores óptimos de w y b que hagan que las predicciones se ajusten lo mejor posible a los datos reales.

Para eso, se define una función de pérdida basada en la verosimilitud negativa (log-loss o entropía cruzada), y se minimiza:

$$\text{función de pérdida (binaria)}: L(w, b) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{p}_i) + (1-y_i) \log(1-\hat{p}_i)]$$

Donde: $\hat{p}_i = \sigma(W^T x_i + b)$
 $N = \# \text{ total de ejemplos}$

esta función penaliza mucho las predicciones que están muy lejos de la clase correcta.

Regularización (opcional pero común): Para evitar sobreajuste, se puede agregar una penalización a los pesos (regularización L2):

$$\min_{w,b} L(w, b) + \lambda \|w\|_2^2$$

Donde: $\|w\|_2^2 = \sum_j w_j^2$
 $\lambda = \text{controla cuánto se penaliza el tamaño de los pesos}$

optimización (búlganos, etc.)

la función se minimiza con algoritmos numéricos como:

- Gradiente descendente
- LBFGS (un método de optimización basado en segundo orden)
- SAG ó SAGA (variantes más rápidas del gradiente).

4) Linear Discriminant Analysis

(Análisis Discriminante Lineal) es un modelo de clasificación supervisado que tiene 2 grandes propósitos:

- Clasificar datos: Predecir a qué clase pertenece un nuevo ejemplo.
- Reducir la dimensión (opcional): Transformar los datos (en un espacio más pequeño) conservando la separación entre clases.

LDA asume que los datos de cada clase se distribuyen de forma Gaussiana (normal), con la misma matriz de covarianza para todas las clases, pero con diferentes medias.

Modelo

Suponer que se tiene k clases, y se quiere predecir la

Clase $y \in \{1, 2, \dots, K\}$, dado un vector de características $x \in \mathbb{R}^n$.

LDA modela la distribución de los datos por clase con:

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

Cada clase K tiene su propia media μ_k .

Todas las clases comparten la misma matriz de covarianza Σ .

Regla de decisión

usando el teorema de Bayes, el modelo calcula la probabilidad de que x pertenezca a cada clase y predice la de mayor probabilidad:

$$\hat{y} = \arg \max_k \delta_k(x) \quad \text{Función discriminante lineal para la clase } k.$$

$$(f(x) = \log P(y=k)) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(y=k)$$

Problema de optimización

El objetivo de LDA es encontrar la proyección lineal $w^T x$ que maximice la separación entre clases.

Para clasificación: máxima separación entre clases.

Encontrar una proyección w que maximice la razón entre la varianza entre clases y la varianza dentro de las clases:

$$\max_w \frac{w^T S_B w}{w^T S_w w} \quad S_B: \text{matriz de dispersión entre clases}$$

$S_w = \sum_{k=1}^K \sum_{i \neq j} (x_i - \mu_k)(x_j - \mu_k)^T$ intra-clase

se calculan así:

$$S_B = \sum_{k=1}^K N_k (\mu_k - \mu)(\mu_k - \mu)^T; \quad S_w = \sum_{k=1}^K \sum_{i \neq j} (x_i - \mu_k)(x_j - \mu_k)^T$$

Donde μ_k : media del grupo k ; μ : media general; N_k : cantidad de muestras en la clase k .

Este es un problema clásico de álgebra lineal: se resuelve encontrando los autovalores y autovectores de $S_w^{-1} S_B$.

5) K-Nearest Neighbors classifier

Modelo de clasificación supervisada no paramétrico.

significa que no entrena un modelo con parámetros como pesos o bias (como lo hacen la regresión logística o SVM), sino que guarda los datos de entrenamiento y toma decisiones en el momento de la predicción.

Modelo

Dado un conjunto de entrenamiento con puntos $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, donde cada $x_i \in \mathbb{R}^n$ y $y_i \in \{1/2, 1, -1\}$, el modelo se define así:

Para un nuevo punto x , se hace lo siguiente:

1. Calcular la distancia entre x y todos los datos del entrenamiento.

• Por defecto, se usa la distancia Euclídea:

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

2. Elegir los k vecinos más cercanos según esa distancia

3. Asignar la clase más frecuente entre esos k vecinos:

$$\hat{y} = \text{modo}(\{y_i : x_i \text{ es vecino más cercano a } x\})$$

Esto significa que el modelo no aprende fórmulas matemáticas, sino que toma decisiones comparando directamente con los datos ya vistos.

Problema de optimización

No resuelve un problema de optimización tradicional. No tiene pesos w , ni bias b , ni una función de pérdida a minimizar. No entrena nada en el sentido clásico.

la elección del parámetro k (el # de vecinos) es clave, porque define un compromiso entre:

• Ruido (Si k es muy pequeño, como 1)

• Suavidad (Si k es grande y suaviza demasiado la frontera entre clases)

Optimización indirecta

lo más parecido a un "problema de optimización" en KNN es elegir el valor de k que minimiza el error de clasificación en validación cruzada.

Por ejemplo: $\min_{k \in \{1, 3, 5, 7, \dots\}} \text{error_validacion}(k)$

Además, puede optimizarse la métrica de distancia (Euclídea, Manhattan, etc) y la ponderación de los vecinos (Por distancia o uniforme).

b) linear SVC

Es un clasificador lineal basado en máquinas de vectores de soporte (SVM).

Su tarea principal es encontrar una linea (o un hiperplano

en más dimensiones) que separe las clases de la mejor manera posible, maximizando el margen entre ellas.

El nombre "lineal" se refiere a que la separación entre las clases es una línea recta (en 2D) o un hiperplano (en dimensiones mayores).

Se usa comúnmente para problemas de clasificación binaria, aunque puede adaptarse a múltiples clases.

Modelo

El modelo predice la clase $y \in \{-1, +1\}$. También puede trabajar con $y \in [0, 1]$ usando una función lineal.

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

La predicción se toma como: $\hat{y} = \begin{cases} +1 & \text{si } f(x) \geq 0 \\ -1 & \text{si } f(x) < 0 \end{cases}$

La idea es que el modelo encuentre w y b de forma que los ejemplos de cada clase queden a un lado diferente del hiperplano $w^T x + b = 0$, y lo más alejados posible de él.

Problema de optimización

A) Caso ideal (separación perfecta - sin errores)

El problema original de una SVM lineal consiste en maximizar el margen entre las clases, es decir, encontrar el hiperplano más "separador".

B) Caso realista (con errores): SVM suave o soft-margin

En la práctica, algunos puntos no se pueden separar perfectamente, por eso se agregan variables de holgura $\xi_i \geq 0$ que permiten violar un poco la restricción.

C) Forma utilizada por linear SVC

No resuelve directamente el problema con restricciones, sino una versión sin restricciones usando la función de pérdida hinge (bissección).

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

7) Random Forest classifier

Modelo de aprendizaje supervisado basado en árboles de decisión. Es un método de "ensamble", lo que significa que combina varios modelos (árboles) para tomar una decisión más robusta.

En vez de entrenar un solo árbol de decisión, entre muchos árboles diferentes, y combina sus predicciones (por votación) para obtener la predicción final.

Modelo: se tienen M árboles de decisión que representan la función $h_1(x), h_2(x), \dots, h_M(x)$

Suponer que se entrena M árboles de decisión, cada uno representado por una función $h_m(x)$, donde $m = 1, 2, \dots, M$.

Para un nuevo punto x , cada árbol predice una clase $y_m \in \{1, 2, \dots, C\}$.

El modelo final hace una votación mayoritaria entre los M árboles:

$$\hat{y} = \text{modo}(h_1(x), h_2(x), \dots, h_M(x))$$

Cada árbol se entrena con $\frac{1}{M}$ de los datos de entrenamiento.

• Una muestra aleatoria (con reemplazo) del conjunto de entrenamiento - esto se llama bootstrap.

• Un subconjunto aleatorio de características al decidir en cada división del árbol - esto introduce diversidad entre los árboles.

Problema de optimización: se busca la mejor división para cada nodo.

No resuelve una única función matemática global de optimización. En su lugar, cada árbol del bosque resuelve su propio problema de optimización local al crecer.

optimización (dentro de cada árbol):

en cada árbol se construye dividendo el conjunto de datos en nodos, eligiendo en cada paso las mejores divisiones (feature y umbral) que maximiza la ganancia de pureza.

Las medidas más comunes para esto son:

$$\text{índice de Gini (más usado por defecto)}: G(t) = \sum_{k=1}^C p_k(1-p_k)$$

donde p_k es la proporción de clase k en el nodo t .

$$\text{Entropía (información)}: H(t) = - \sum_{k=1}^C p_k \log_2 p_k$$

La división óptima es la que minimiza la impureza ponderada de los nodos hijos:

$$\text{impureza (padre)} = \left[\frac{N_{izq}}{N} \text{ impureza (izq)} + \frac{N_{der}}{N} \text{ impureza (der)} \right]$$

se divide el espacio en óvalos oblicuos que siguen reglas de optimización que cumplen el principio de "independencia" de los datos en el espacio.

Los óvalos tienen forma ovalada y no rectangular, y están divididos en regiones de alta pureza (regiones) y regiones de mezcla (mezcla).