

# Clase 7: Optimización con restricciones

Martín Errázquin (merrazquin@fi.uba.ar)

Análisis Matemático para Inteligencia Artificial

Clase 7

## 1 Optimización con restricciones

## 2 Optimización matemática

- Fused Logistic Regression
- Histogram-based Decision Trees

## Optimización con restricciones

# Optimización con restricciones de igualdad

Dado el problema de optimización con restricciones:

$$\begin{aligned} & \text{opt } f(x_1, \dots, x_n) \\ & \text{s.a. } g_1(x_1, \dots, x_n) = 0 \\ & \quad \vdots \\ & \quad g_m(x_1, \dots, x_n) = 0 \end{aligned} \tag{1}$$

donde  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  y  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$  con  $i = 1, \dots, m$ .

Se denomina **función Langrangiana** a la función de  $n + m$  variables  $\mathcal{L}$  definida por:

$$\mathcal{L}(\bar{\lambda}, \bar{x}) = f(\bar{x}) + \lambda_1 g_1(\bar{x}) + \dots + \lambda_m g_m(\bar{x}), \quad \text{con } \bar{\lambda} = (\lambda_1, \dots, \lambda_m)$$

# Teorema de condición necesaria de Lagrange

Sea  $D \subseteq \mathbb{R}^n$  abierto. Sea (1) con  $m < n$ , donde  $f, g_i$   $i = 1, m$  son funciones definidas de  $D$  en  $\mathbb{R}$ , con derivadas parciales primeras continuas en  $D$  y

$$B = \{\bar{x} \in D : g_i(\bar{x}) = 0, i = 1, m\}$$

el conjunto de soluciones factibles.

Entonces, si  $\bar{x}^*$  es un óptimo local de (1) tal que la matriz jacobiana de  $\bar{g}(\bar{x}^*)$  tal que  $|J\bar{g}(\bar{x}^*)_m| \neq 0$ , existen  $m$  números reales  $\lambda_1^*, \dots, \lambda_m^*$  tales que son solución del sistema:

$$\nabla f(\bar{x}^*) + \sum_{i=1, m} \lambda_i^* \nabla g_i(\bar{x}^*) = \bar{0}$$

Las soluciones factibles que verifican esta ecuación se denominan *puntos estacionarios*.

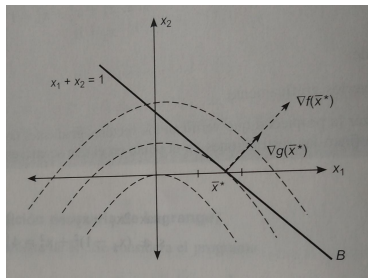
Los números  $\lambda_1, \dots, \lambda_m$  se denominan **multiplicadores de Langrange** asociados a las  $m$  restricciones en el punto  $\bar{x}^*$ .

# Ejemplo

$$\begin{aligned} \min f(x_1, x_2) &= x_1^2 + 2x_2 \\ \text{s.a. } x_1 + x_2 &= 1 \end{aligned}$$

Así definimos:

$$g(x_1, x_2) = x_1 + x_2 - 1.$$



- El conjunto de soluciones factibles es la recta  $x_2 = -x_1 + 1$ .
- Las curvas de nivel de la función  $f(x_1, x_2) = x_1^2 + 2x_2$  son parábolas de la forma  $x_1^2 + 2x_2 = k, k \in \mathbb{R}$ .

$\therefore \bar{x}^* = (1, 0)$  y  $f(\bar{x}^*) = 1$ .

Calculamos los vectores gradientes:  $\nabla f(x_1, x_2) = (2x_1, 2)$  y  $\nabla g(x_1, x_2) = (1, 1)$ , ambos vectores son l.d. en  $\bar{x}^*$ .

# Optimización con restricciones de desigualdad

## Condiciones necesarias de primer orden de Fritz-John

Dado el problema de optimización con restricciones:

$$\begin{aligned} \text{opt } & f(x_1, \dots, x_n) \\ \text{s.a. } & g_1(x_1, \dots, x_n) \leq 0 \\ & \vdots \\ & g_m(x_1, \dots, x_n) \leq 0 \end{aligned} \tag{2}$$

donde  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  y  $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$  con  $i = 1, \dots, m$ . Sea  $\bar{x}^*$  un punto tal que  $I = \{i : g_i(\bar{x}^*) = 0\}$ ,  $f, g_i$  diferenciables en  $\bar{x}^*$ . Entonces, si  $g_i, i \notin I$  son continuas en  $\bar{x}^*$  se verifica que es solución y existen escalares  $\lambda_0, \lambda_i, i \in I$  no todos nulos tales que:

$$\lambda_0 \nabla f(\bar{x}^*) + \sum_{i=1, m} \lambda_i \nabla g_i(\bar{x}^*) = \bar{0}$$

para  $0 \leq \lambda_0, \lambda_i, i \in I$ , y  $g_i(\bar{x}^*) \leq 0, i = 1, m$ .

# Dualidad en Optimización: Problema dual

Dado el problema primal:

$$\begin{aligned} \min f(x_1, \dots, x_n) \\ \text{s.a. } g(x_1, \dots, x_n) \leq b \\ x_i \geq 0 \end{aligned}$$

Tenemos el problema dual:

$$\begin{aligned} \max D(y_1, \dots, y_m) \\ \text{s.a. } h(y_1, \dots, y_m) \geq b \\ y_j \leq 0 \end{aligned}$$

- Se define  $D(y_1, \dots, y_m) \doteq \min_{\bar{x}} \mathcal{L}(\bar{\lambda}, \bar{x})$ .
- Si  $\bar{x}^*$  es una solución factible del problema primal e  $\bar{y}^*$  es una solución del dual entonces  $f(\bar{x}^*) \geq D(\bar{y}^*)$ .
- Si un problema no tiene un óptimo finito entonces el otro no es factible.



# Función Convexa

**Def:** Un subconjunto  $S \subset \mathbb{R}^n$  es **convexo** si para cada par de puntos  $x, y \in S, \alpha \in [0, 1]$  se verifica que  $z = \alpha x + (1 - \alpha)y \in S$ .

**Obs:**  $X_1, X_2$  son dos conjuntos convexos, entonces,

- $X_1 \cap X_2$  es convexo.
- $X_1 + X_2 = \{x_1 + x_2 \in \mathbb{R}^n : x_1 \in X_1, x_2 \in X_2\}$ .
- si  $L$  es una transformación lineal,  $L(X_1)$  es convexa.

## Función Convexa

**Def:** Sea  $M \subset \mathbb{R}^n, M \neq \emptyset$  convexo,  $f : M \rightarrow \mathbb{R}$ . Entonces se dice que:

- $f$  es convexa en  $M$  sii  $\forall x, y \in M, \forall \alpha \in [0, 1]$  se verifica que:

$$f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$$

## Desigualdad de Jensen

- $f$  es estrictamente convexa en  $M$  sii  $\forall x, y \in M, \forall \alpha \in (0, 1)$  se verifica que:

$$f(\alpha x + (1 - \alpha)y) < \alpha f(x) + (1 - \alpha)f(y)$$

# Condiciones para convexidad de funciones diferenciables

Sea  $M \subset \mathbb{R}^n$ ,  $M \neq \emptyset$  abierto convexo,  $f : M \rightarrow \mathbb{R}$  diferenciable, se dice que:  $f$  es convexa en  $M \Leftrightarrow \forall x, y \in M, f(y) \geq f(x) + \nabla f(x)(y - x)$   
O bien,  $(\nabla f(y) - \nabla f(x))(y - x) \geq 0$

**Prop:** Sea  $M \subset \mathbb{R}^n$ ,  $M \neq \emptyset$  abierto convexo,  $f : M \rightarrow \mathbb{R}$   $f \in \mathcal{C}^2$ . Entonces,  $f$  es convexa en  $M$  sii  $\forall x \in M, y^T Hf(x)y \geq 0$ , para cualquier  $y \in \mathbb{R}^n$ .

## Condiciones Suficientes de Optimalidad Global

$$\begin{aligned} & \text{opt } f(x_1, \dots, x_n) \\ & \text{s.a. } g_1(x_1, \dots, x_n) = 0 \\ & \quad \vdots \\ & \quad g_m(x_1, \dots, x_n) = 0 \end{aligned}$$

con  $m < n$  donde  $f$  y  $g_i, i = 1, \dots, m$  son funciones  $\mathcal{C}^1$  en un conjunto abierto  $D \in \mathbb{R}^n$ . Entonces se verifica que si  $f$  es convexa en el conjunto de soluciones factibles  $B$  y, las funciones  $g_i$  son lineales, **todos los puntos estacionarios son mínimos globales.**

# Programación Lineal

Un problema de programación lineal es donde tanto la función objetivo como las funciones que definen las restricciones son lineales. La forma general es:

$$\begin{aligned} \text{opt } & c_1x_1 + \dots + c_nx_n \\ \text{s.a. } & a_{11}x_1 + \dots + a_{1n}x_n \leq b_1 \\ & \vdots \\ & a_{m1}x_1 + \dots + a_{mn}x_n \leq b_m \\ & d_{11}x_1 + \dots + d_{1n}x_n \geq e_1 \\ & \vdots \\ & d_{r1}x_1 + \dots + d_{rn}x_n \geq e_r \\ & g_{11}x_1 + \dots + g_{1n}x_n = b_1 \\ & \vdots \\ & g_{s1}x_1 + \dots + g_{sn}x_n = b_s \end{aligned}$$

**Definición:** Sea  $S \subset \mathbb{R}^n$  convexo  $S \neq \emptyset$ ,  $\bar{x}^* \in S$  es un punto extremo de  $S$  si  $\bar{x}^*$  no puede expresarse como combinación lineal convexa de puntos de  $S$  distintos de él.

- 1 Es un problema convexo ya sea de minimización o maximización.
- 2 La solución óptima, si existe, es global.
- 3 Nunca existen óptimos locales que no sean globales.
- 4 Puede tener o no solución, en caso de existir se encuentra en único punto o en infinitos.

**Objetivo:** Se busca hallar  $\min \bar{c}\bar{x}$ , s.a.  $A\bar{x} = \bar{b}$ ;  $\bar{x} \geq 0$ ,  
donde  $A \in \mathbb{R}^{m \times n}$ ,  $m < n$ ,  $rg(A) = m$ .

**Métodos estándar de solución:** Algoritmo Simplex y revised Simplex, Interior Point (IPM).

# Programación cuadrática

En el caso de que la función objetivo sea una cuadrática convexa:

$$\begin{aligned} \min_{x \in \mathbb{R}^d} \quad & \frac{1}{2} x^T Q x + c^T x \\ \text{s.a : } & Ax \leq b \end{aligned}$$

donde  $A \in \mathbb{R}^{m \times d}$ ,  $b \in \mathbb{R}^m$ ,  $c \in \mathbb{R}^d$ .

La matriz  $Q \in \mathbb{R}^{d \times d}$  es simétrica y definida positiva, y por lo tanto la función objetivo es convexa. El langrangiano está dado por:

$$L(x, \lambda) = \frac{1}{2} x^T Q x + c^T x + \lambda^T (Ax - b)$$

Para obtener  $D(\lambda) = \max_x L(x, \lambda)$  buscamos  $x$  derivando  $L$  y despejando:

$$\nabla L(x) = Qx + (c + A^T \lambda) = 0 \Rightarrow x = -Q^{-1}(c + A^T \lambda)$$

Y substituyendo obtenemos la lagrangiana dual:

$$D(\lambda) = -\frac{1}{2}(c + A^T \lambda)^T Q^{-1}(c + A^T \lambda) - \lambda^T b.$$

# Optimización matemática

## Fused Logistic Regression

## El problema

Se tiene un vector de múltiples features, todas categóricas, al cual se le aplica un [CatBoost Encoding](#) para obtener "embeddings" de cada valor. Luego, se aplica una [regresión logística](#).

En producción, el modelo se despliega en un mini servidor donde debe realizar predicciones de verdadero-falso para *una observación a la vez*.

Debido a limitaciones de la librería Category Encoders, que asume un DataFrame como entrada, el código se ve aproximadamente así:

```
def predict(self, input_dict):  
    x = to_dataframe(input_dict)  
    x = self.encoder.transform(x)  
    prediction = self.regressor.predict(x)  
    return prediction > self.threshold
```

Y esto tiene (en benchmark) una latencia promedio de 39 ms.



Para cada feature  $i = 1, \dots, p$ , el encoder construye un dataframe con dos columnas: COUNT y SUM. La regla de transformación es:

$$\tilde{X}(X) = \begin{cases} \frac{SUM(X) + \mu \cdot a}{COUNT(X) + a} & \text{si } COUNT(X) > 1 \\ \mu & \text{si } COUNT(X) = 1 \end{cases}$$

donde  $\mu = \bar{y}$  es la media global y  $a = 1$  es un hp. de regularización.

Luego, pasando a la regresión logística:

- ❶ cada feature  $i$  es multiplicada por su peso  $\beta_i$ :  $s = \sum_{i=1}^p \beta_i \cdot \tilde{X}_i$
- ❷ se suma el intercept:  $z = s + z_0$
- ❸ se aplica la función logística  $\sigma(t) = \frac{1}{1+e^{-t}}$ :  $\hat{y} = \sigma(z)$
- ❹ se devuelve la comparación con un threshold aprendido  $pred = \hat{y} > \theta$

- 1 Se pueden usar diccionarios comunes  $\rightarrow$  se puede evitar instanciar un DataFrame.
- 2 Si  $\tilde{X} = \mu$  cuando  $COUNT = 1$ , no tiene sentido diferenciarlo de  $COUNT = 0$  (cuando no está presente)  $\rightarrow \tilde{X}$  deja de ser partida.
- 3 Si no se pregunta por el valor de  $COUNT$  no tiene sentido guardar  $SUM$  y  $COUNT$  separados  $\rightarrow$  se puede guardar el embedding  $\tilde{X}_i$  precomputado.
- 4 Cada feature tiene un mapeo propio  $\rightarrow$  en vez de  $\tilde{X}_i$  se puede guardar precomputado  $\beta_i \cdot \tilde{X}_i$  (aunque se debe guardar un  $\beta_i \cdot \mu_i \forall i$ ).
- 5 Para un  $\theta \in (0, 1)$  fijo  $\exists \sigma^{-1}(\theta) = \text{logit}(\theta)$ , luego  $\sigma(z) > \theta \Leftrightarrow z > z_\theta \rightarrow$  se puede precomputar  $\text{logit}(\theta)$  y evitar computar  $\sigma(z)$ .
- 6 Separando  $z = s + z_0$  tenemos  $z > \text{logit}(\theta) \Leftrightarrow s > \text{logit}(\theta) - z_0 \rightarrow$  podemos precomputar  $z_\theta = \text{logit}(\theta) - z_0$  y evitar computar esa suma.

## El resultado

Tras la drástica reducción en operaciones, tenemos que en total el modelo está realizando solamente un dict lookup por cada feature (inevitable con categóricas), una suma y una comparación.

El código se ve aproximadamente así:

```
def predict(self, input_dict):
    z_tilde = sum(
        self.embeddings[feature].get(value, self.mus[feature])
        for feature, value in input_dict.items()
    )
    return z_tilde > self.z_threshold
```

Y esto tiene (en benchmark) una latencia promedio de 0.01 ms, es decir **un speedup de 3800x**. En el caso de tiempos máximos, la versión original fue de 192 ms vs 0.487 ms del fused.

Como extra, el modelo fused ocupa la mitad de espacio en disco.

## Histogram-based Decision Trees

## Mini-intro de DTs

Suponiendo un árbol de decisión de regresión con función de impureza  $H(\text{samples})$  y todas features  $i = 1, \dots, p$  continuas sin NaNs, en cada nodo (con  $n$  obs.) se construye la regla de decisión buscando, para cada feature:

$$\theta = \operatorname{argmin} \frac{n_L}{n} H(x_L) + \frac{n_R}{n} H(x_R)$$

donde  $x_L/x_R$  son los conjuntos de obs. que para esa feature tienen un valor menor/mayor-igual a  $\theta$  y  $n_L/n_R$  la cantidad de obs. de cada uno.

Por ejemplo, para error cuadrático se usa:

$$H(X) = \frac{1}{n_X} \sum_{i=1}^{n_X} (y_i - \bar{y})^2$$

Cabe notar entonces que armar un nodo de  $n$  obs. y  $p$  features tiene complejidad  $\mathcal{O}(p \cdot n \cdot \log(n))$ .

## El problema

Si no se ordenaran las observaciones por cada feature, calcular los  $x_L$  y  $x_R$  sería  $\mathcal{O}(n)$  para cada posible threshold, luego cada feature sería  $\mathcal{O}(n^2)$ . Si uno primero ordena, con complejidad  $\mathcal{O}(n \log(n))$ , luego se puede quitar de  $x_R$  y agregar a  $x_L$  cada elemento en  $\mathcal{O}(1)$ .

Para un dataset grande (supongamos  $n > 10^4$ ) ordenar las features es el cuello de botella del entrenamiento de los árboles.

## Puntos de mejora

La clave: árboles sólo miran orden relativo, no valor. Entre otras cosas, son invariantes a cualquier transformación que preserve orden.

Una optimización que no nos importaba: se pueden reemplazar los valores por el `uint` del lugar que ocupan para reducir el tamaño en memoria de  $X$ .

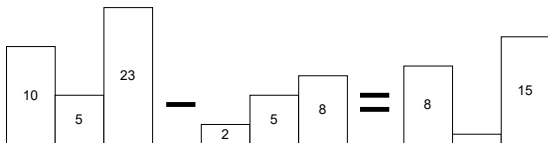
La optimización que sí nos importa: hacer *binning/quantization*. Si nos quedamos con e.g. 256 cuantiles podemos:

- Reducir a 1B el tamaño en memoria de la feature por obs. → Reducción de tamaño en memoria de hasta 8x.
- Sólo considerar  $b = 256$  posibles valores de  $\theta$  (varios órdenes de magnitud menos que  $n$ ) → encontrar un threshold pasa a ser  $\mathcal{O}(b)$ .
- Simplemente guardar algunas métricas para cada bin → uso de estructura histograma.

# El resultado

El uso de histogramas y el *binning* del dataset suele reducir en gran medida el costo en memoria de entrenar el modelo. Además, armar un nodo de  $n$  obs. y  $p$  features pasa de tener complejidad  $\mathcal{O}(p \cdot n \cdot \log(n))$  a  $\mathcal{O}(p \cdot b)$ .

Si bien el armado del histograma de los hijos es, en principio,  $\mathcal{O}(n \cdot p)$ , se pueden realizar optimizaciones extra para que las cotas sean más fuertes. Además, sólo se calcula el histograma para un hijo, y el otro es la diferencia con el padre.



En el colab adjunto se puede ver cómo una implementación sin optimizaciones tiene un speedup de aproximadamente **4x** en datasets de  $n = 10^5$  sólo utilizando histogramas.