

# **CLASIFICADOR KNN Y OPTIMIZACIÓN HIPERPARÁMETROS**

**APRENDIZAJE DE MAQUINA I - CEIA - FIUBA**

***Dr. Ing. Facundo Adrián Lucianna***

***Dr. Ing. Álvaro Gabriel Pizá***

# REPASO CLASE ANTERIOR

- Definición de Machine Learning
- Tipos de aprendizaje:
  - Aprendizaje supervisado: Regresión y clasificación
  - Aprendizaje no supervisado: Agrupamiento y reducción dimensional
  - Aprendizaje profundo: Redes neuronales

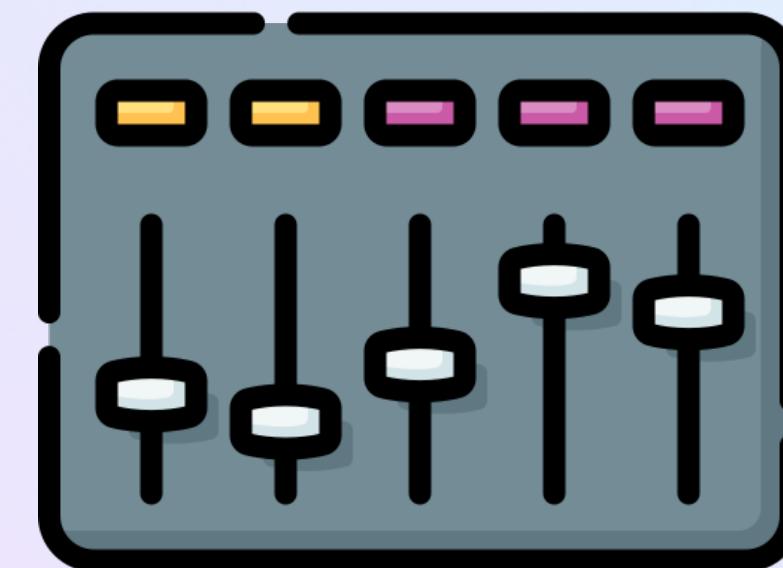
# REPASO CLASE ANTERIOR

Observation →

Features					Label
Position	Experience	Skill	Country	City	Salary (\$)
Developer	0	1	USA	New York	103100
Developer	1	1	USA	New York	104900
Developer	2	1	USA	New York	106800
Developer	3	1	USA	New York	108700
Developer	4	1	USA	New York	110400
Developer	5	1	USA	New York	112300
Developer	6	1	USA	New York	116100
Developer	7	1	USA	New York	117800

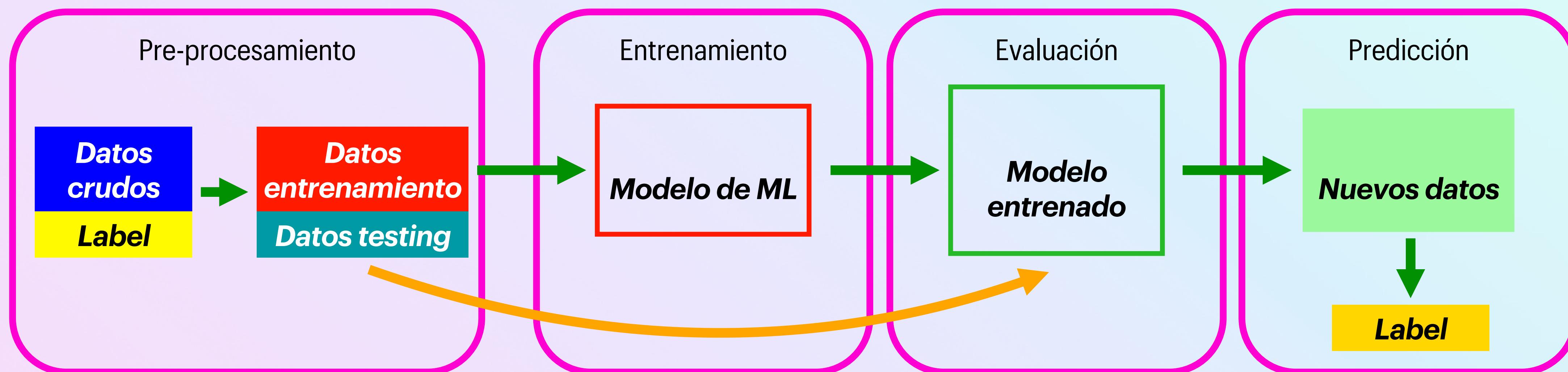
# REPASO CLASE ANTERIOR

- Los algoritmos de Machine Learning tienen parámetros “internos” que no dependen de los datos. Estos parámetros se llaman **hiperparámetros**. Por ejemplo, una red neuronal tiene como hiperparámetros la función de activación o la constante de entrenamiento.



- Llamamos **generalización** a la capacidad del modelo de hacer predicciones nuevas utilizando datos nuevos.

# REPASO CLASE ANTERIOR

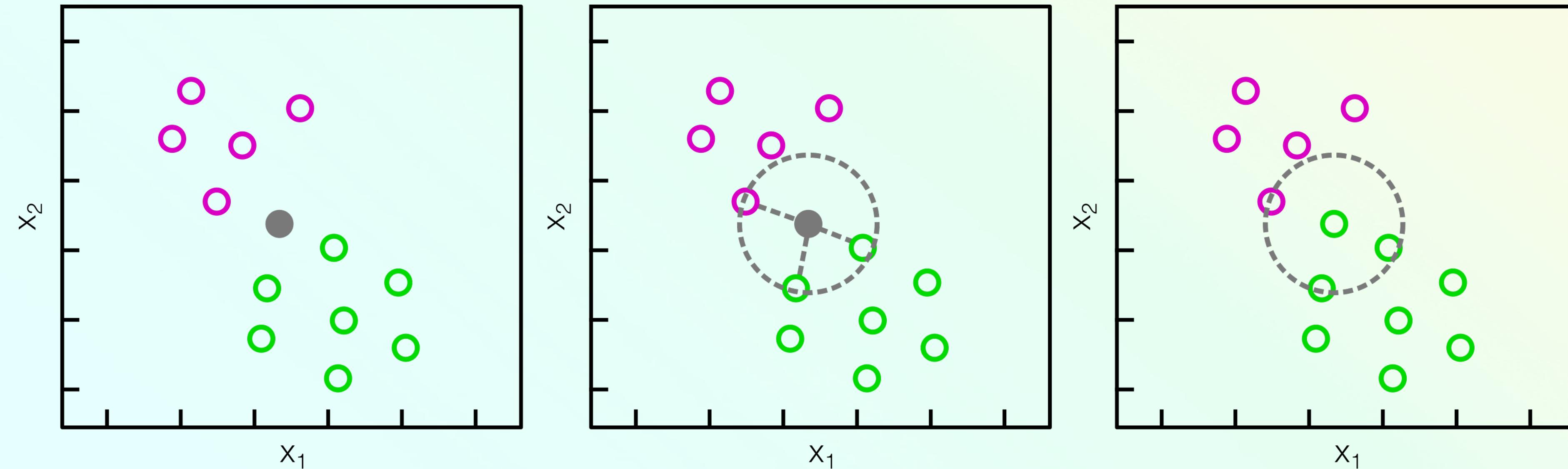


# **CLASIFICADOR KNN**

# KNN

El clasificador de  $k$  vecinos más cercanos (KNN o  $k$ -NN), es un algoritmo que utiliza la proximidad de sus vecinos para hacer clasificaciones sobre la agrupación de un punto.

La idea se basa de la **suposición** de que se pueden encontrar puntos similares cerca uno del otro en base a votación de pluralidad (se elige la clase en función de la moda de la clase de sus vecinos). Este modelo no obtiene una salida de probabilidad, solo nos dice de qué clase es.



# KNN

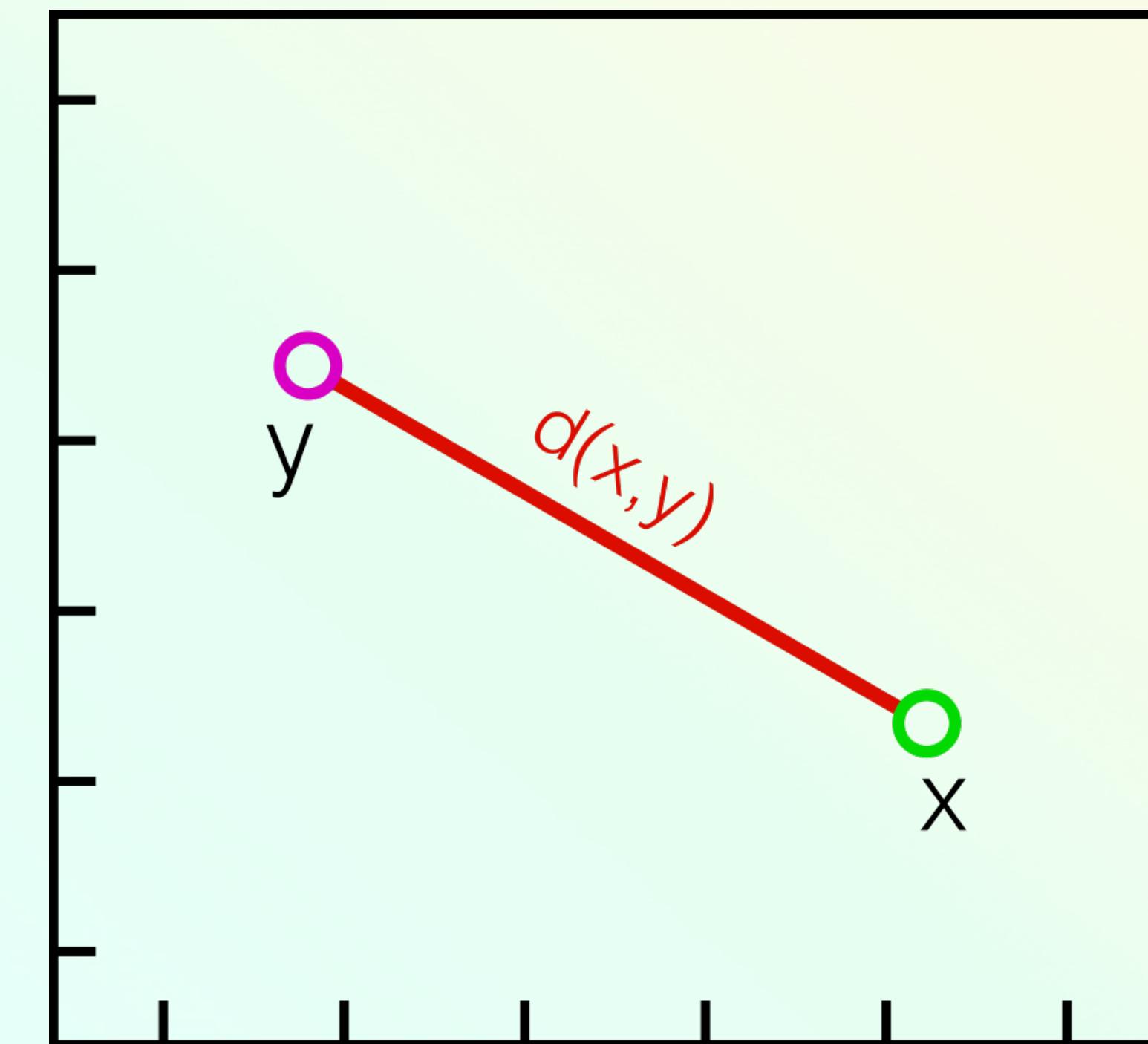
Como vimos, este algoritmo se fija en la distancia entre observaciones. ¿Ahora la pregunta es como medimos la distancia?

Hay múltiples maneras de medir distancia entre dos puntos geométricos. Vamos a definir algunas.

# KNN

**Distancia euclíadiana** (modulo 2): Es la más conocida, es la mínima distancia (una recta) entre dos puntos en un espacio euclidiano. Es adecuada para datos numéricos continuos.

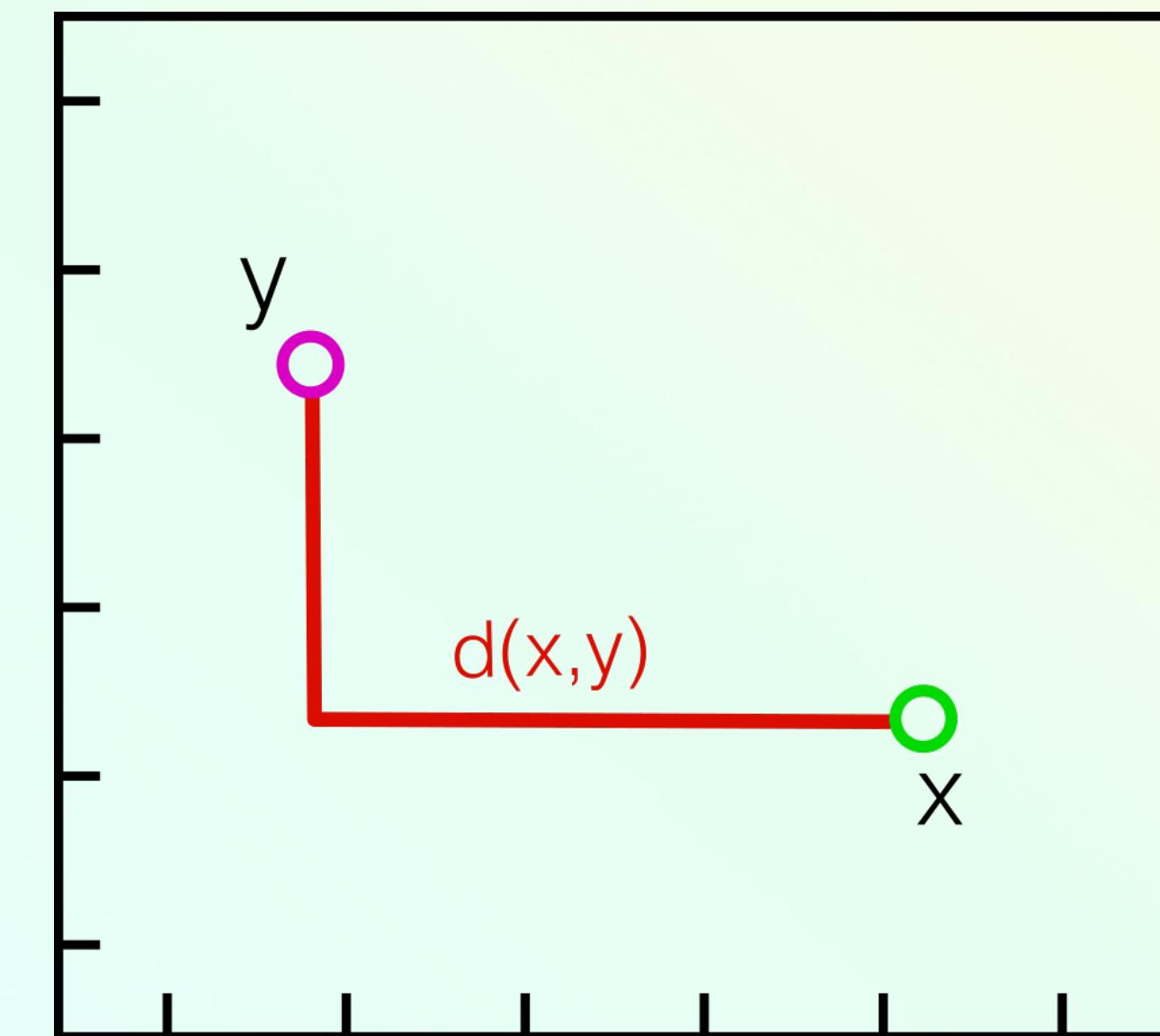
$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$



# KNN

**Distancia de Manhattan** (modulo 1): Es la medida del valor absoluto entre dos puntos. Se conoce también como distancia taxi o de cuadra de ciudad, ya que mide distancias como en una ciudad. Es adecuada para datos que pueden tener correlaciones no lineales y no sigue la suposición de varianzas iguales en todas las dimensiones.

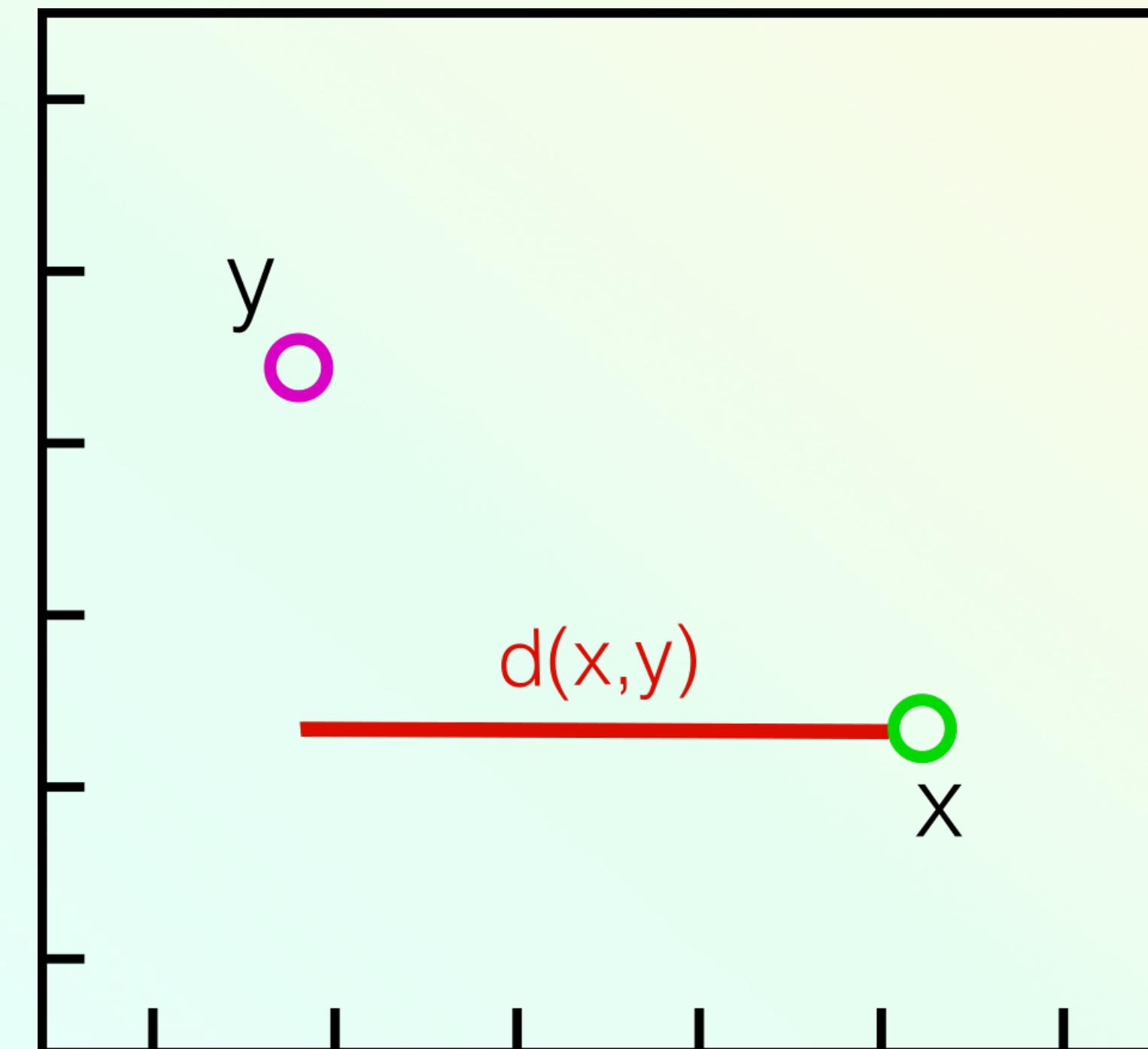
$$d(x, y) = \sum_{i=1}^n |x_i - y_i|$$



# KNN

**Distancia de Chebyshev** (modulo infinito): Se calcula como la diferencia máxima entre las coordenadas de dos puntos. Es adecuada cuando las dimensiones son independientes y la distancia máxima es relevante.

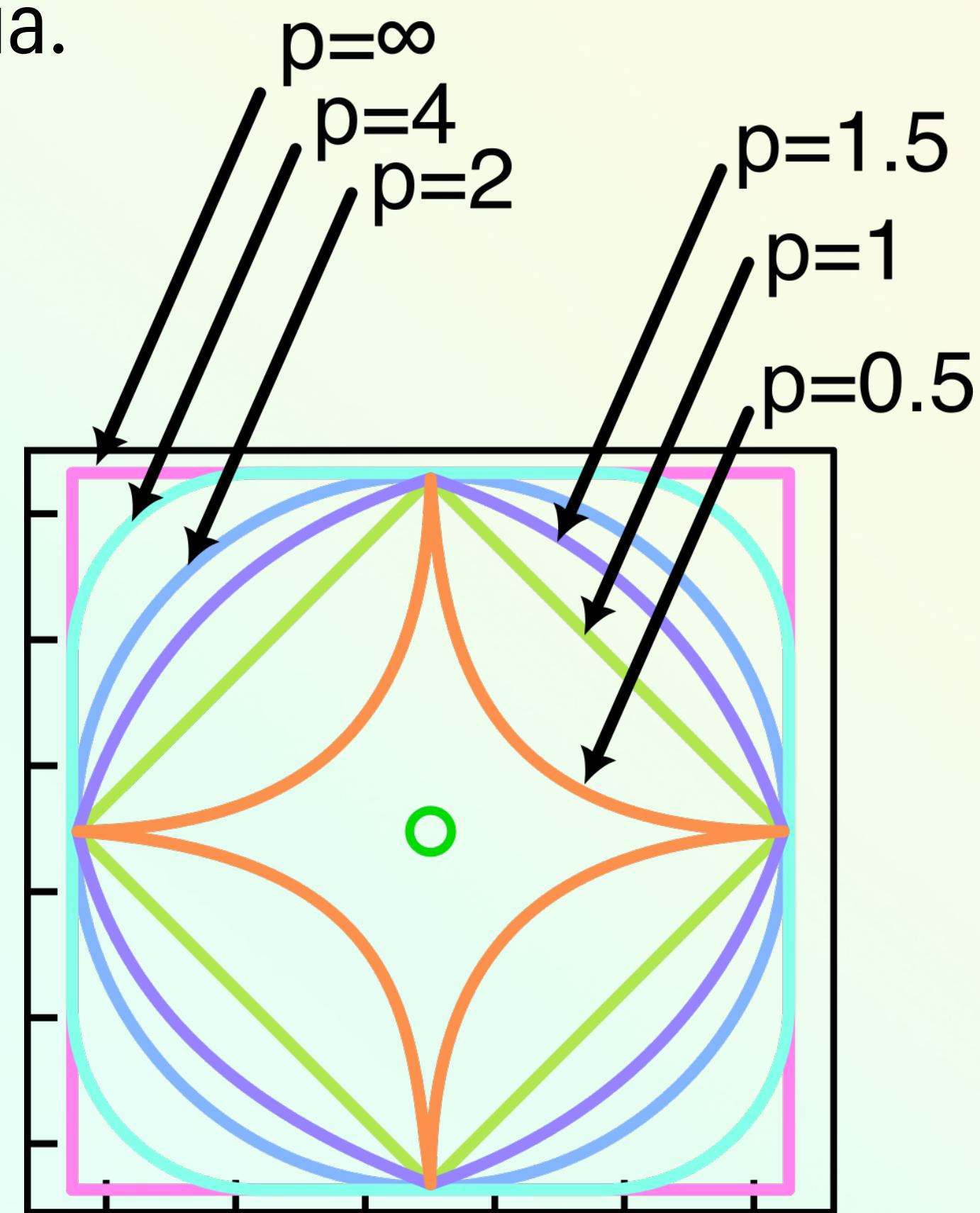
$$d(x, y) = \max(|x_i - y_i|)$$



# KNN

**Distancia de Minkowski:** Es una medida generalizada que incluye las anteriores. Posee un parámetro,  $p$ , es la que permite variar el tipo de distancia.

$$d_p(x, y) = \left( \sum_{i=1}^n (x_i - y_i)^p \right)^{1/p}$$

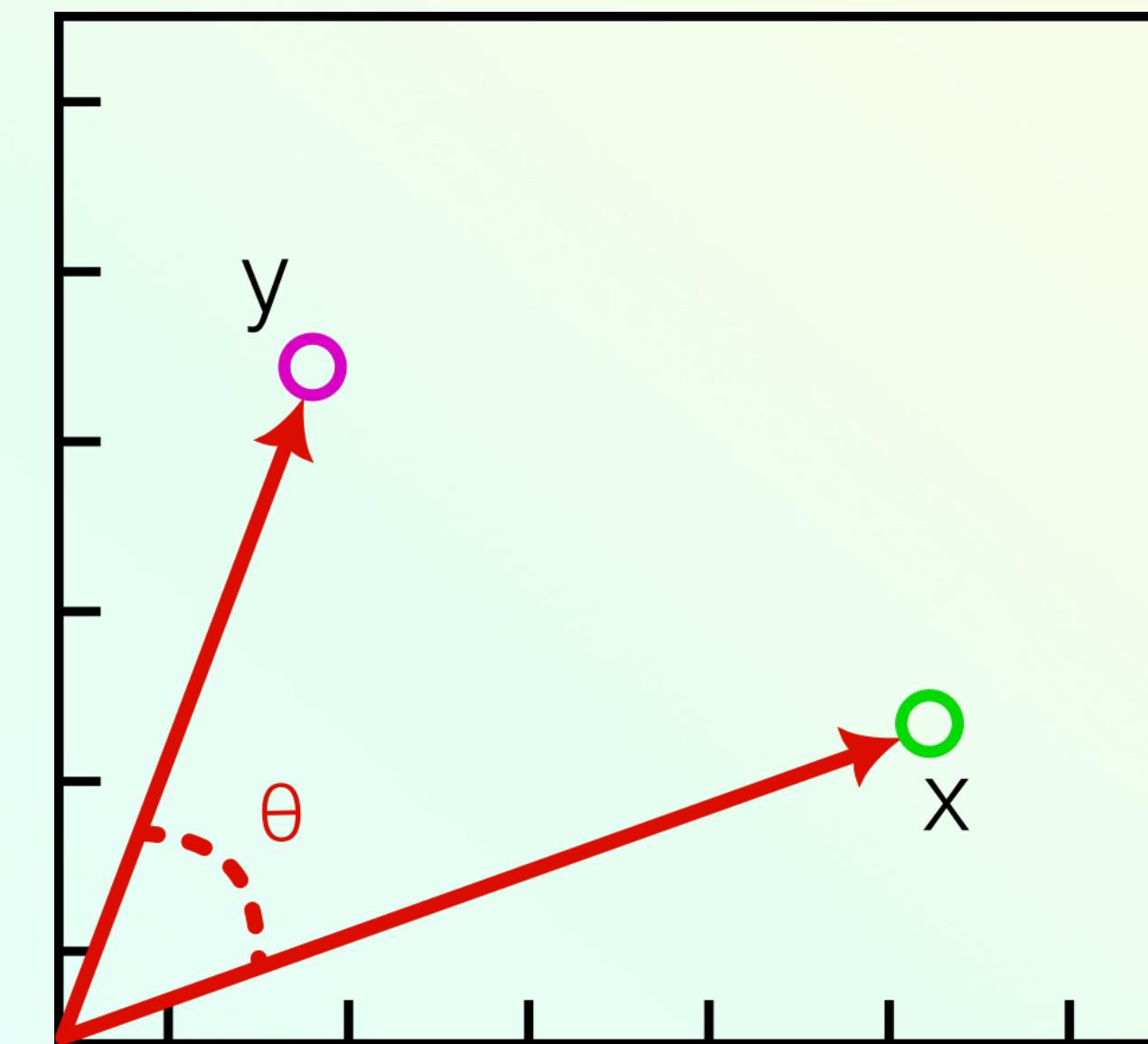


# KNN

**Distancia Coseno:** La similitud coseno mide la similitud entre dos vectores como el coseno del ángulo entre ellos, y la distancia es 1 menos la similitud coseno. Es adecuada para datos donde la magnitud de los vectores es irrelevante, pero si su orientación.

$$d_c(x, y) = 1 - S_c(x, y)$$

$$S_c(x, y) = \cos(\theta) = \frac{x \cdot y}{\|x\| \|y\|}$$



# KNN

**Distancia de Canberra:** Es una métrica de distancia ponderada que se utiliza comúnmente para datos numéricos y pondera más las diferencias en las dimensiones donde los valores son pequeños. Es la distancia de Manhattan ponderada.

$$d(x, y) = \sum_{i=1}^n \frac{|x_i - y_i|}{|x_i| + |y_i|}$$

# KNN

**Distancia de Jaccard:** Se utiliza comúnmente en conjuntos o datos binarios. Mide la similitud entre dos conjuntos como el tamaño de su intersección dividido por el tamaño de su unión.

$$J(x, y) = \frac{|x \cap y|}{|x \cup y|}$$

$$d_J(x, y) = 1 - J(x, y)$$

Asimetrico

x	1	0	1	0	0	0	1	1
y	1	0	0	0	1	0	0	1

$\left. \begin{array}{l} J=2/5 \\ d_J=3/5 \end{array} \right\}$

**Distancia de Hamming:** Se usa típicamente con vectores booleanos, en donde se mide la cantidad de elementos del vector que son diferentes entre sí.

x	1	0	1	0	0	0	1	1
y	1	0	0	0	1	0	0	1

$\left. \begin{array}{l} \\ d_H=3 \end{array} \right\}$

# KNN

**Distancia de Gower:** Es una métrica de distancia que puede manejar datos mixtos (numéricos y categóricos) y tiene en cuenta la escala de las variables y la similitud entre las categorías. Esta entre 0 y 1.

- Datos numéricos: Se calcula usando la distancia de Manhattan, pero para cada atributo se la divide por el rango de valores (poblacional o de muestra).
- Datos categóricos: Si el atributo es igual es 0, sino es 1.

# KNN

Dada la métrica de distancia, debemos definir el valor de  $k$ , que es quien define con cuantos vecinos se usará para determinar la clasificación de un punto.

Por ejemplo, si  $k=1$ , la observación se asignará a la misma clase de su vecino más cercano.

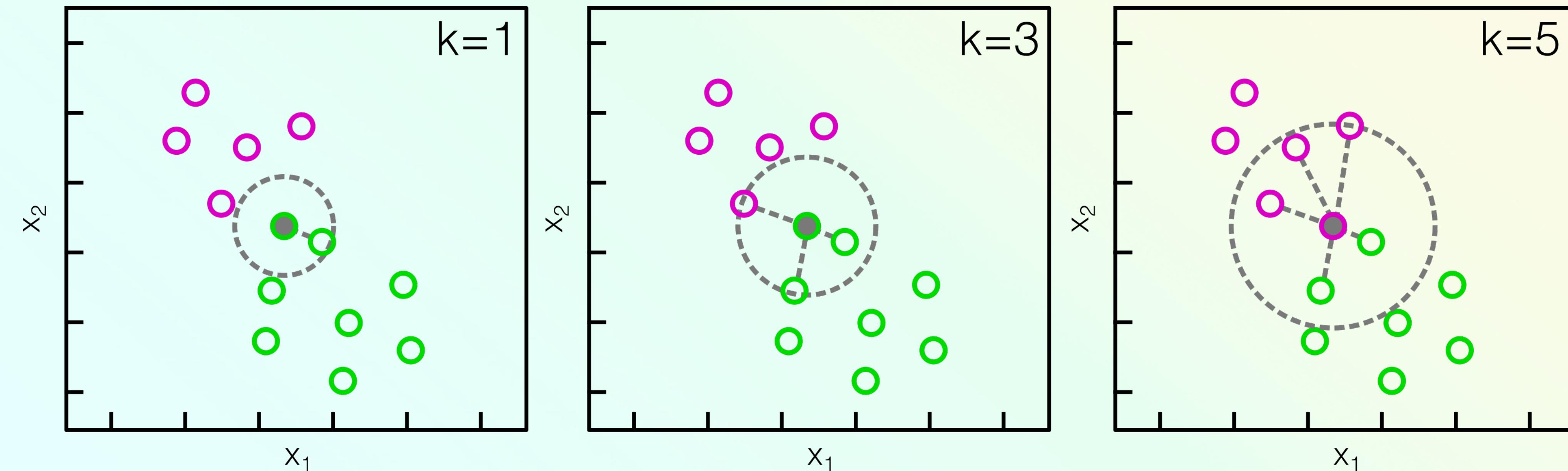
Definir  $k$ , el cual es un hiper-parámetro justo al tipo de distancia elegida, es un acto de equilibrio.

Valores bajos de  $k$  pueden tener una varianza alta, pero un sesgo bajo, y valores altos de  $k$  un sesgo alto y poca varianza.

En general, se recomienda tener un **número impar** para  $k$  para evitar empates en la clasificación.

Este algoritmo no tiene “entrenamiento” ya que debe guardar todo el dataset para evaluar a nuevos valores a que clase pertenece. Si el dataset de entrenamiento es muy grande, puede tener dificultades para almacenarse o ejecutarse.

# KNN



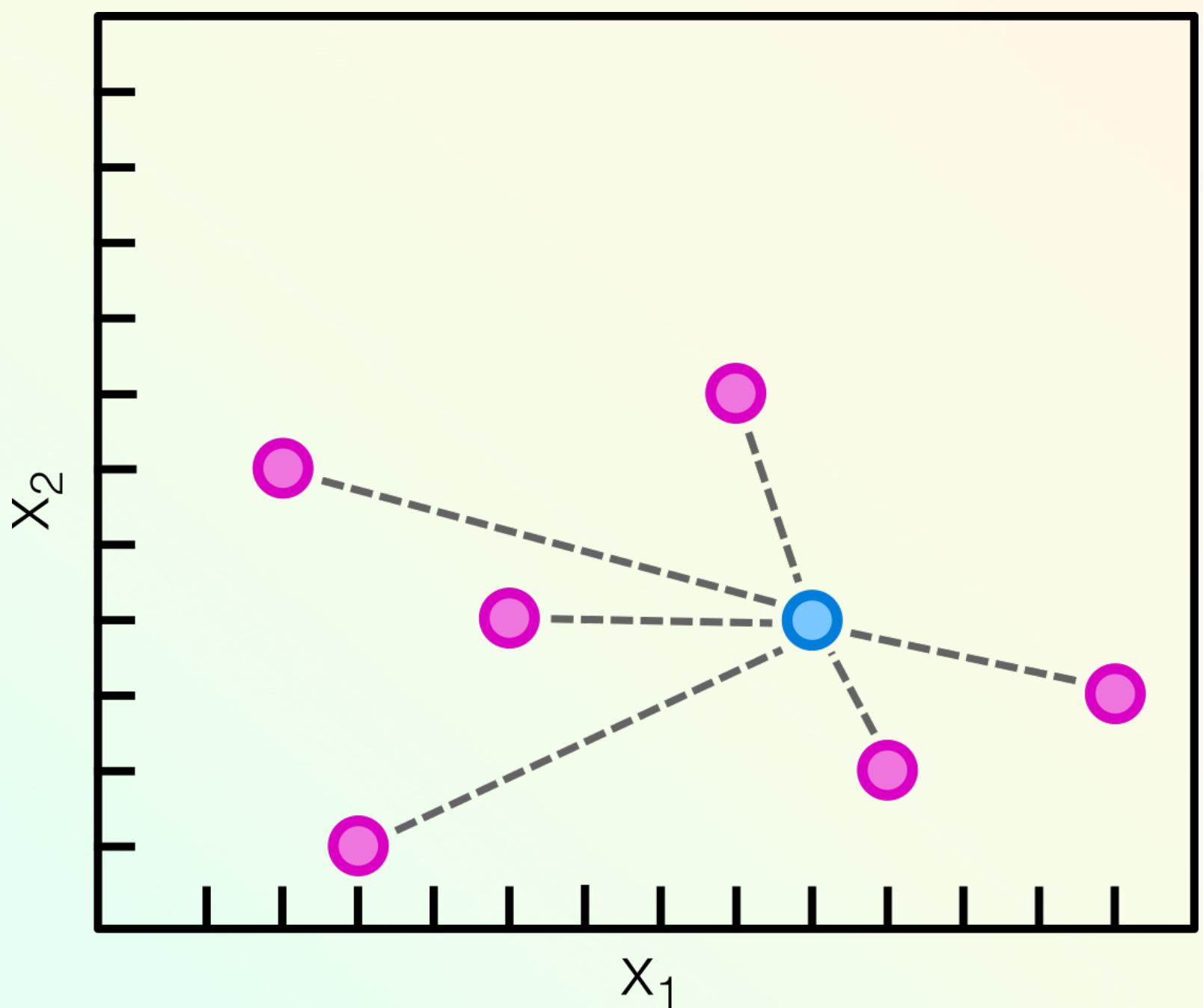
**ENCONTRANDO A LOS  
VECINOS**

# ENCONTRANDO A LOS VECINOS

Un detalle que estamos pasando por alto en este modelo es la implementación de la búsqueda de los vecinos.

La forma más básica de hacer esto es por **fuerza bruta**. Es dado una nueva observación, calcular la distancia a todos con respecto al set de entrenamiento.

El problema es que no escala bien con dataset grandes ya que tiene una complejidad de  $O(DN^2)$ , con  $N$  observaciones y  $D$  atributos.



# ENCONTRANDO A LOS VECINOS

Entonces necesitamos usar algoritmos de búsquedas. Uno popular es **K-D Tree**, el cual arma un árbol de búsqueda.

Se basa en el concepto de que si un punto A está muy distante de B y B está cerca de C, entonces A está lejos de C sin necesidad de explícitamente calcularlo. En este caso se reduce a complejidad máxima de  $O[DN \log(N)]$ .

Este algoritmo divide el espacio D dimensional en donde cada nodo va separando una dimensión por vez en dos.

Para entenderlo veamos un ejemplo en 2 dimensiones con los siguientes puntos:

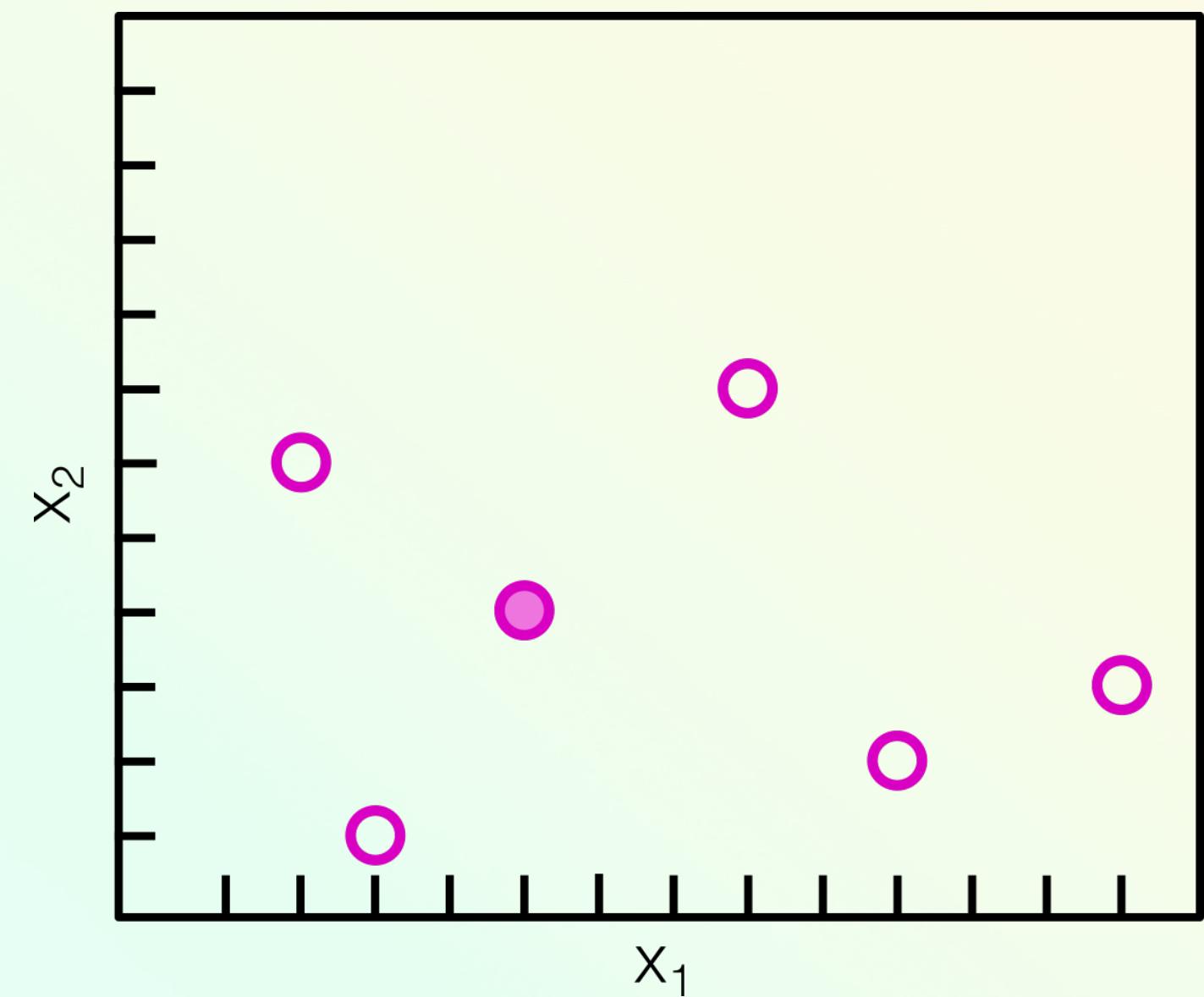
Dimensiones	P1	P2	P3	P4	P5	P6
$x_1$	2	3	5	8	10	13
$x_2$	6	1	4	7	2	3

# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
<b>P3</b>	5	4
P4	8	7
P5	10	2
P6	13	3

Arrancamos tomando un punto para armar el árbol, elijamos el punto **P3**:

(5,4)

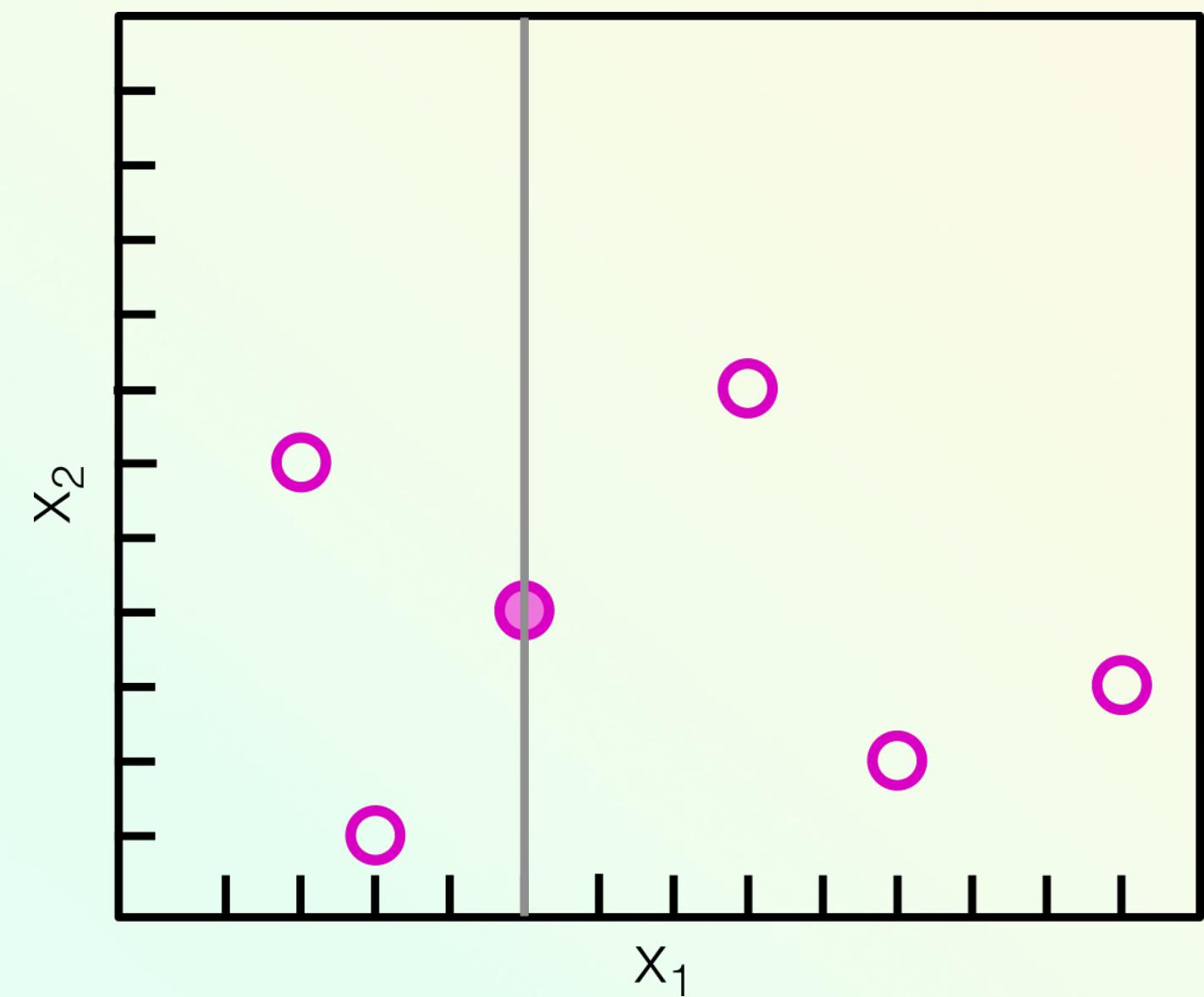


# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
<b>P3</b>	5	4
P4	8	7
P5	10	2
P6	13	3

Arrancamos tomando un punto para armar el árbol, elijamos el punto **P3**: Separamos el espacio en dos con respecto al eje  $x_1$

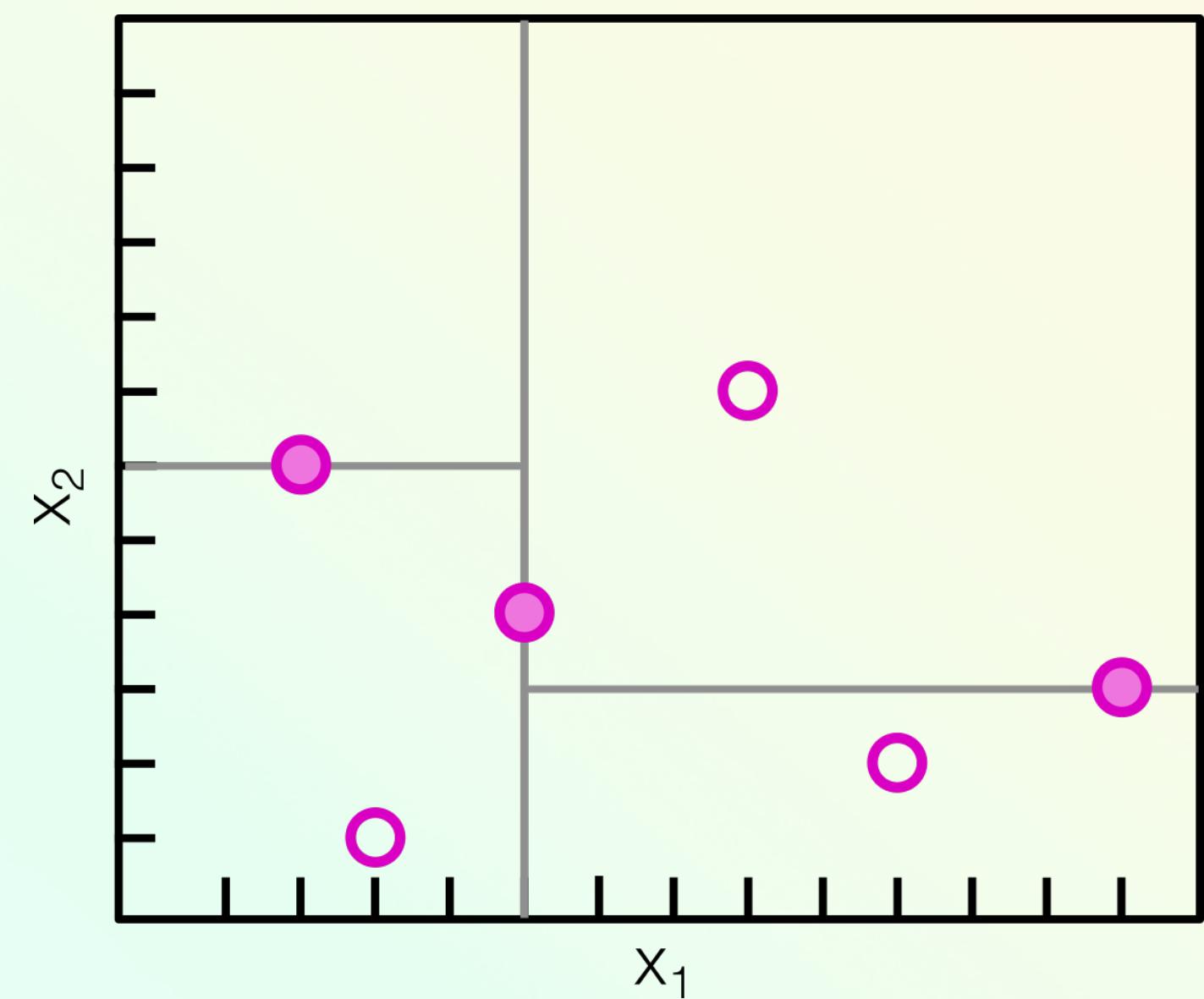
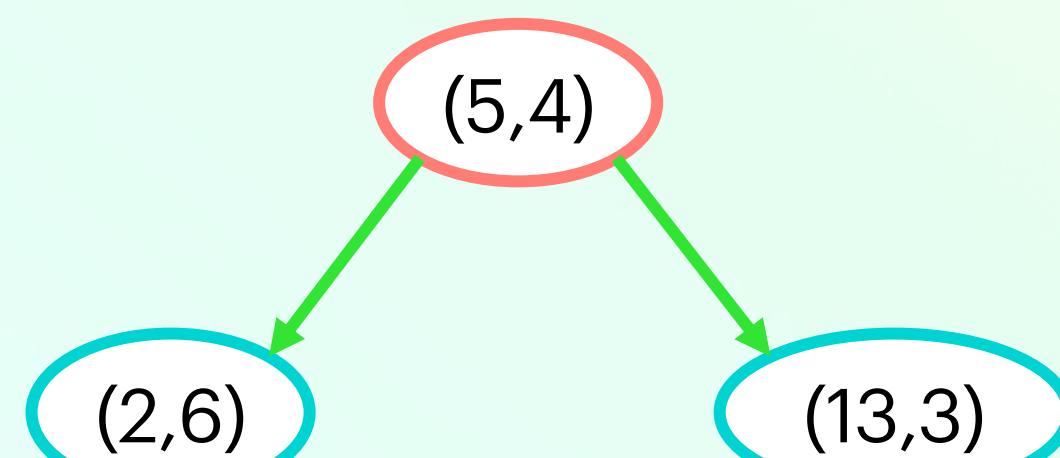
(5,4)



# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
<b>P3</b>	5	4
P4	8	7
P5	10	2
<b>P6</b>	13	3

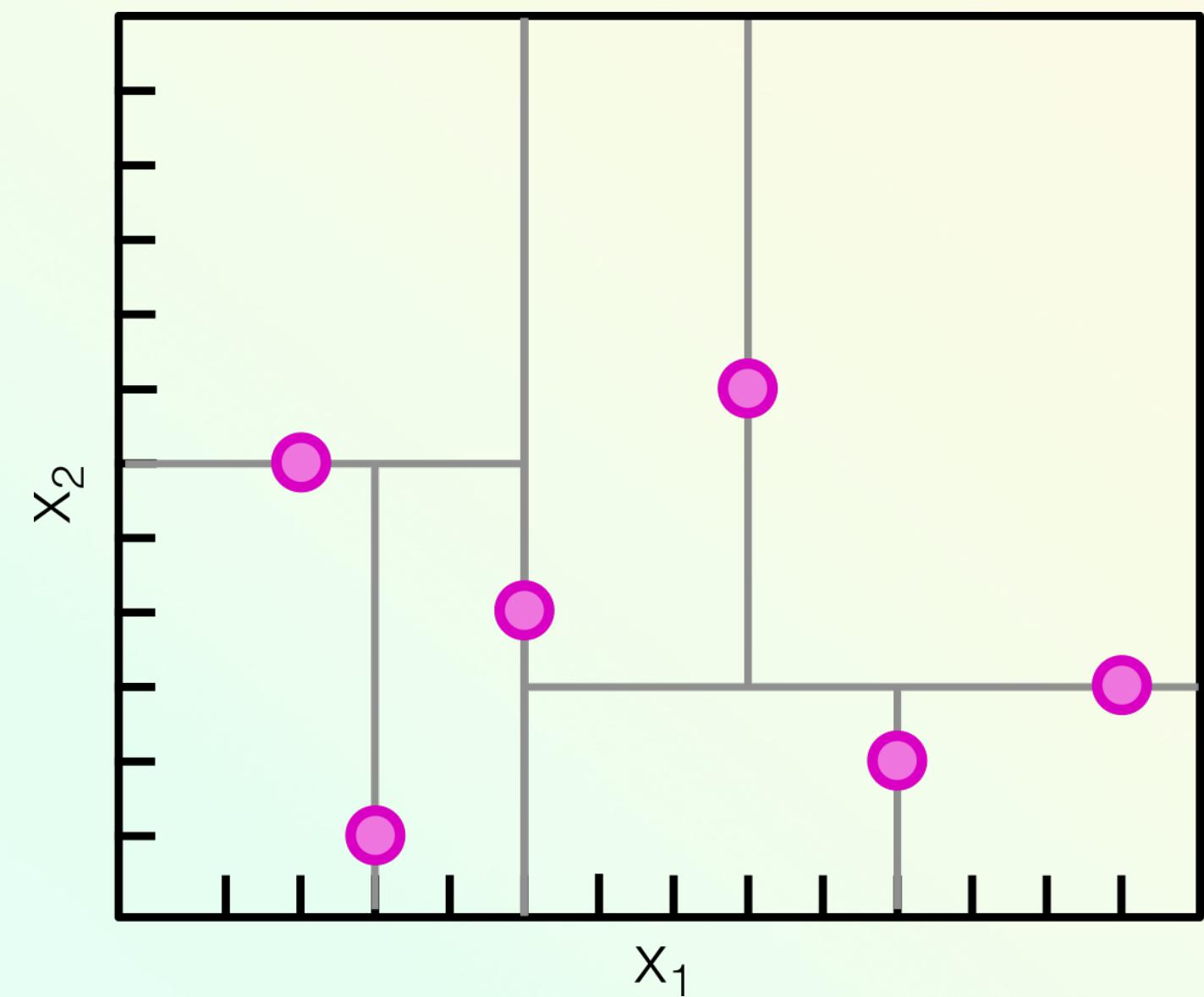
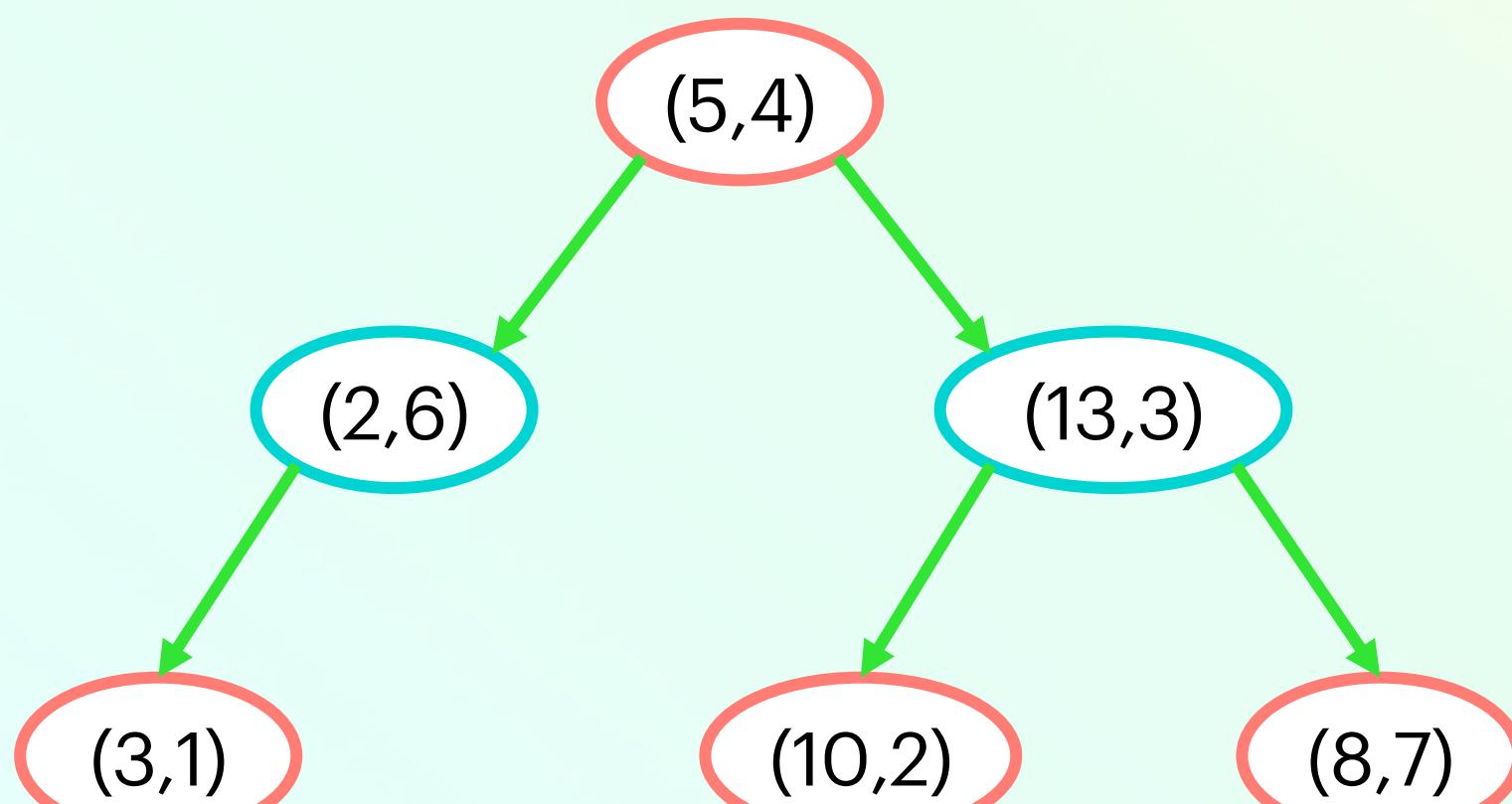
Una vez separado el espacio, para cada lado, buscamos quien está arriba o debajo inmediatamente con respecto al eje  $x_2$ , y los usamos para separar con respecto a este eje:



# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

Finalmente, para la siguiente profundidad, organizamos a los siguientes nodos con respecto a  $x_1$ , y así sucesivamente...

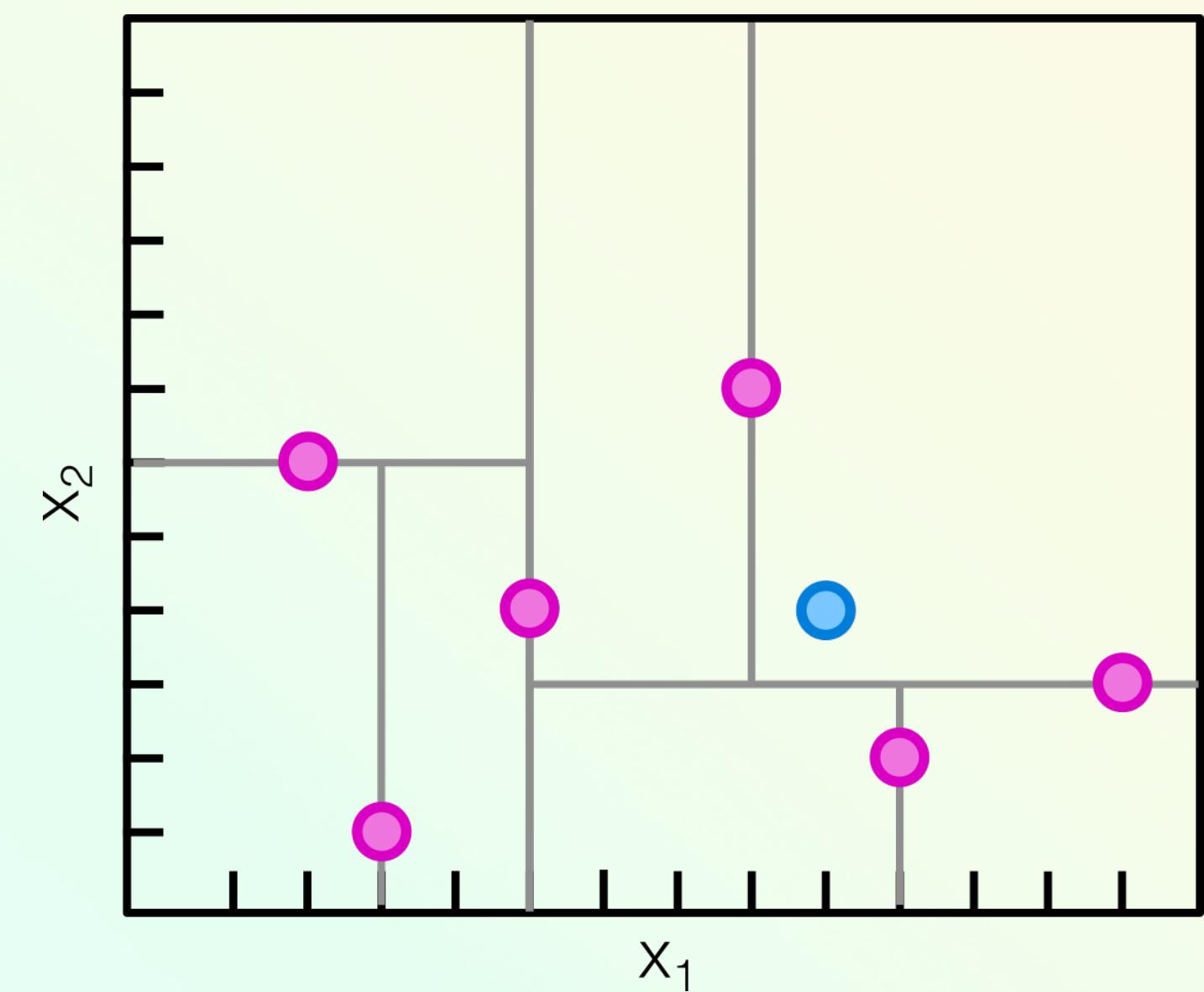
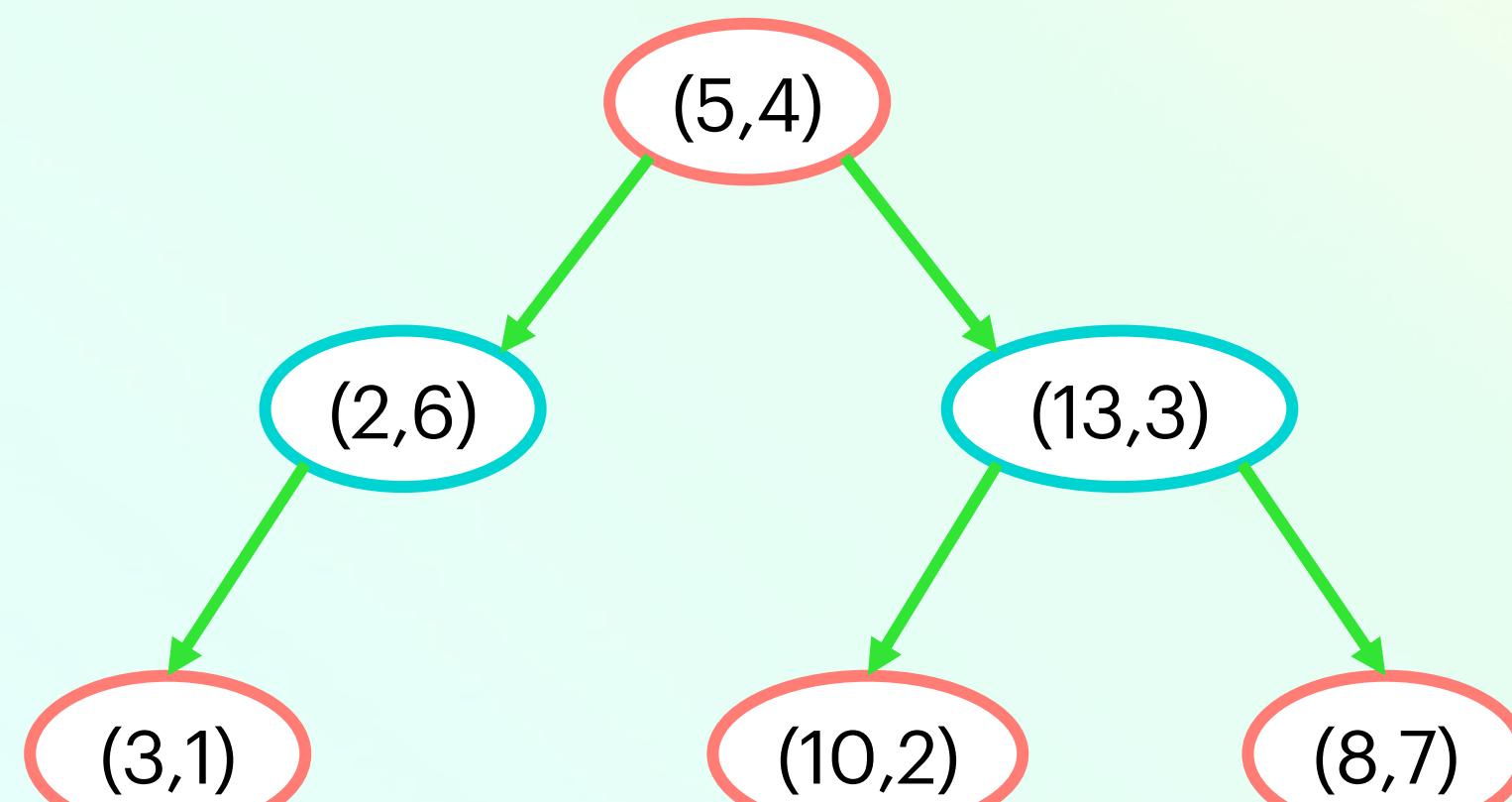


# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

Una vez que construimos el árbol, podemos usarlo para calcular distancia de nuevo puntos, por ejemplo, del punto (9,4).

Se encuentra la hoja en el que el nodo se encuentra...

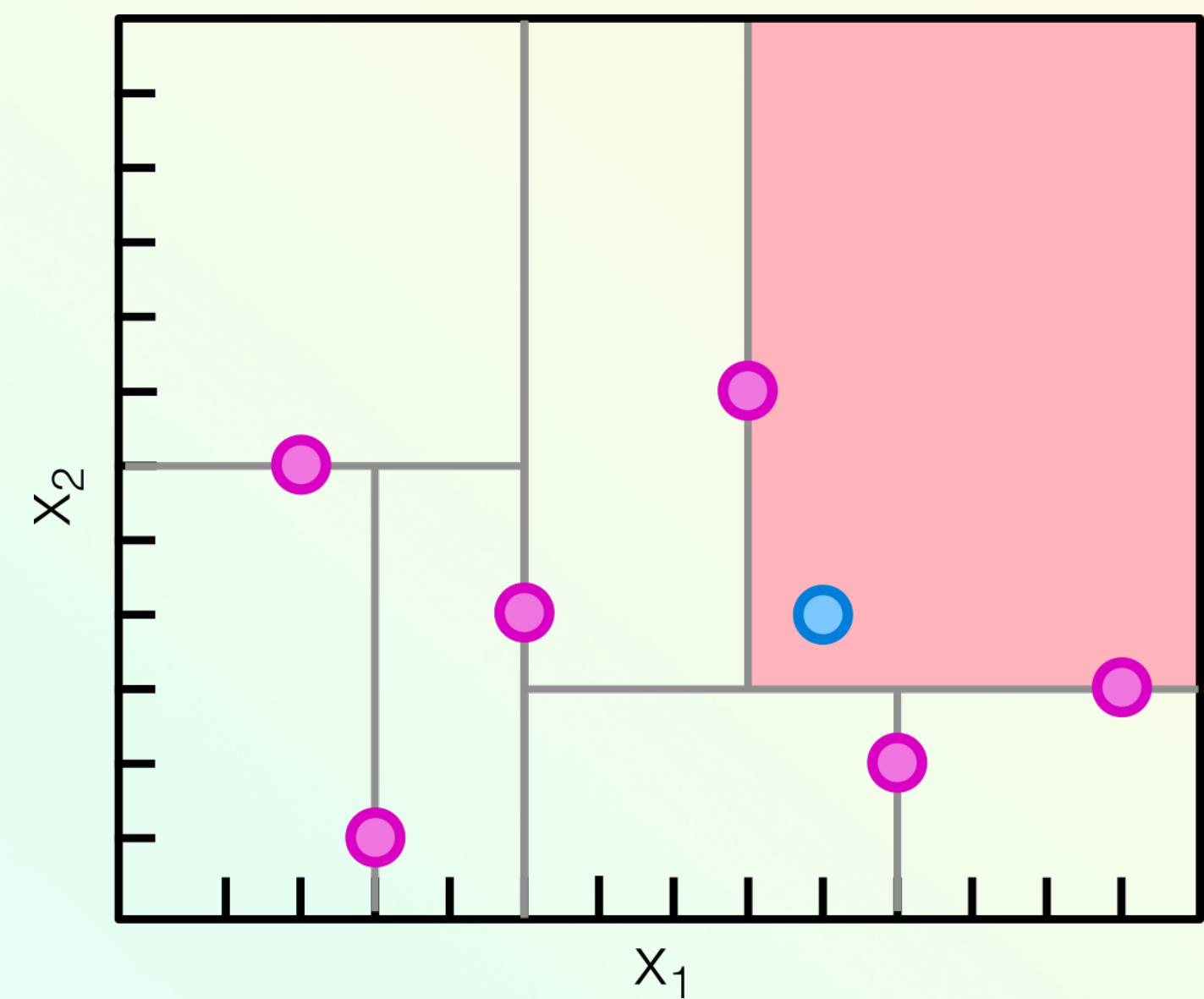
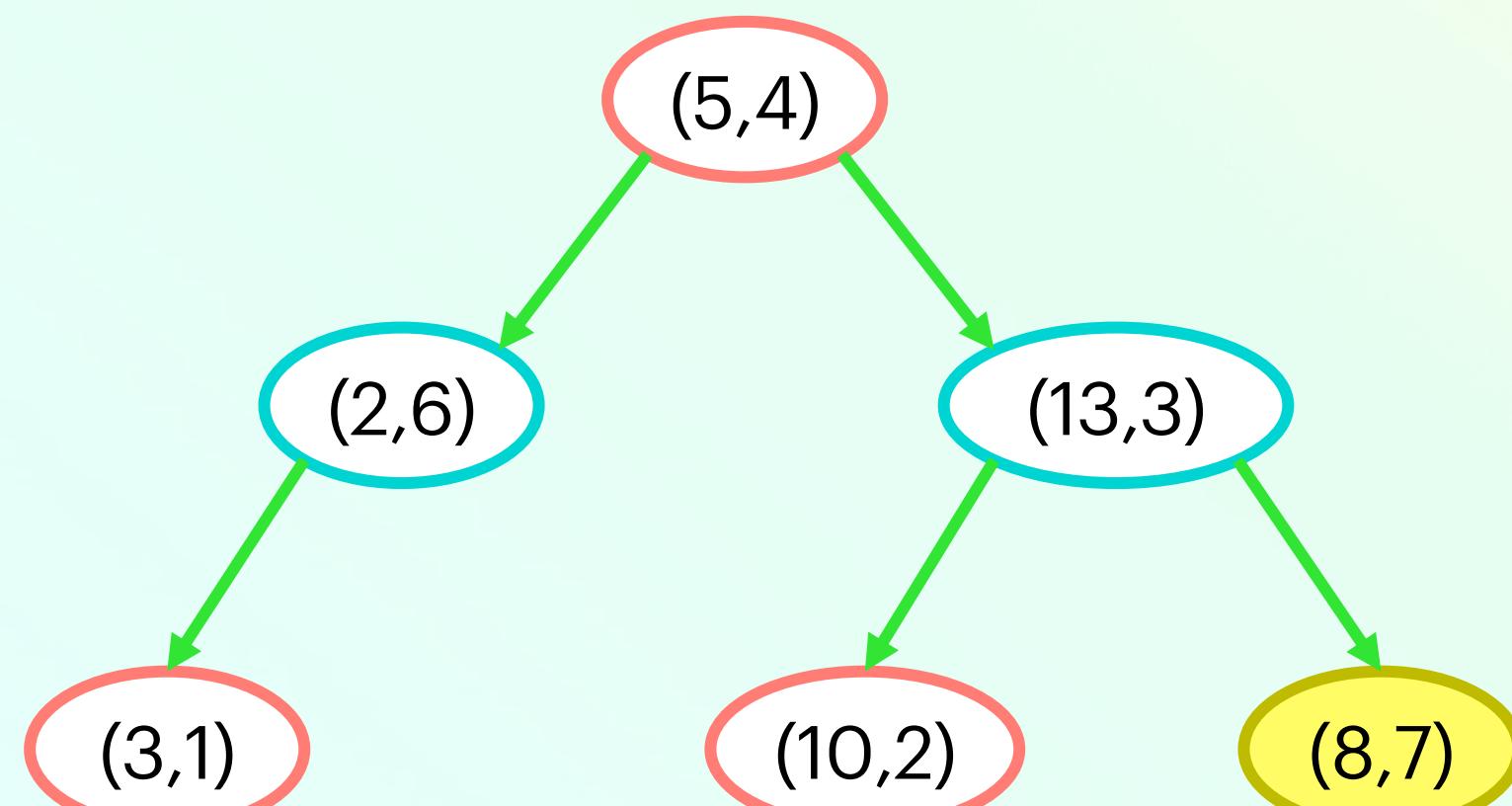


# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

Una vez que construimos el árbol, podemos usarlo para calcular distancia de nuevo puntos, por ejemplo, del punto (9,4).

Se encuentra la hoja en el que el nodo se encuentra...

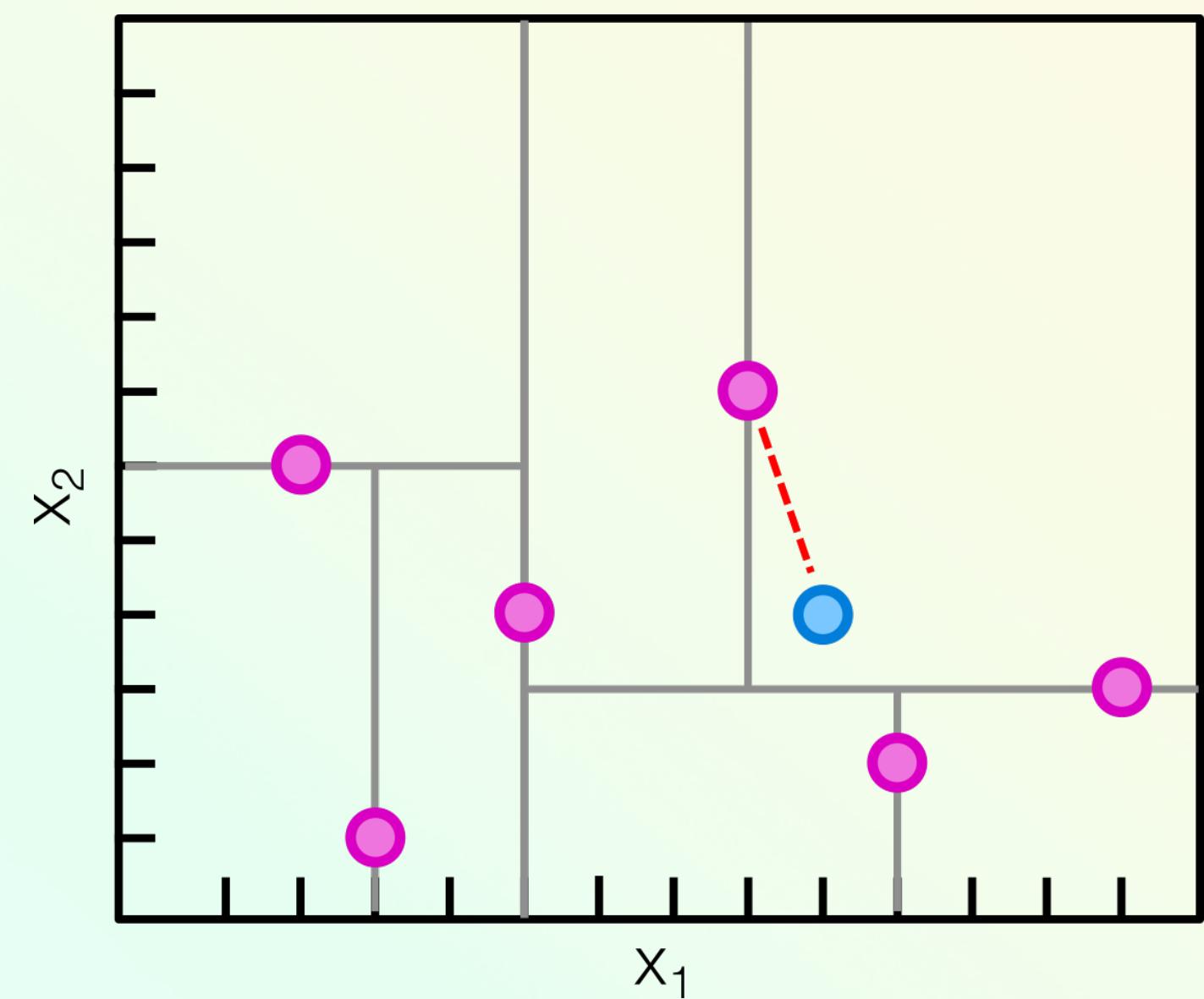
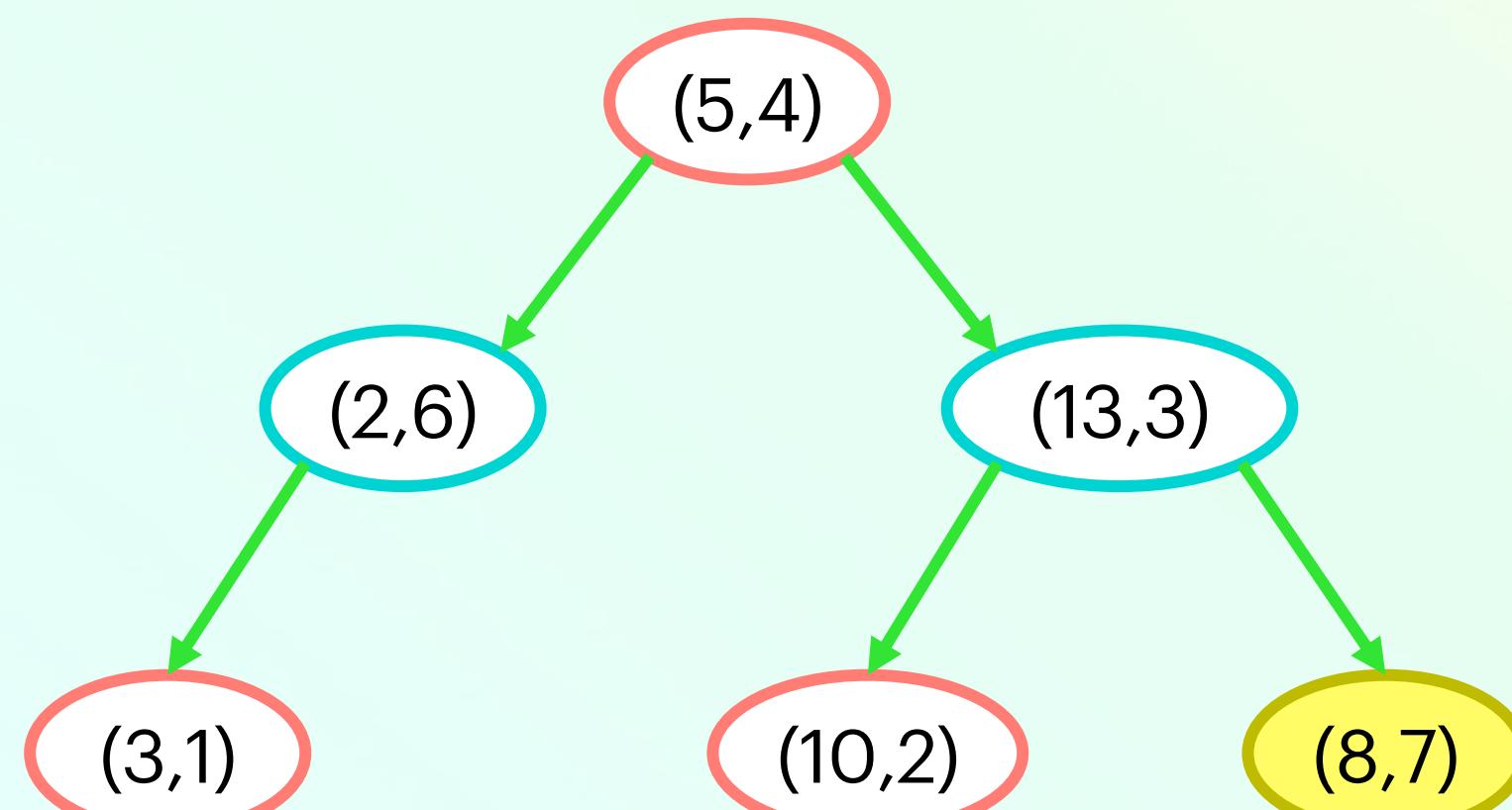


# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

Una vez que construimos el árbol, podemos usarlo para calcular distancia de nuevo puntos, por ejemplo, del punto (9,4).

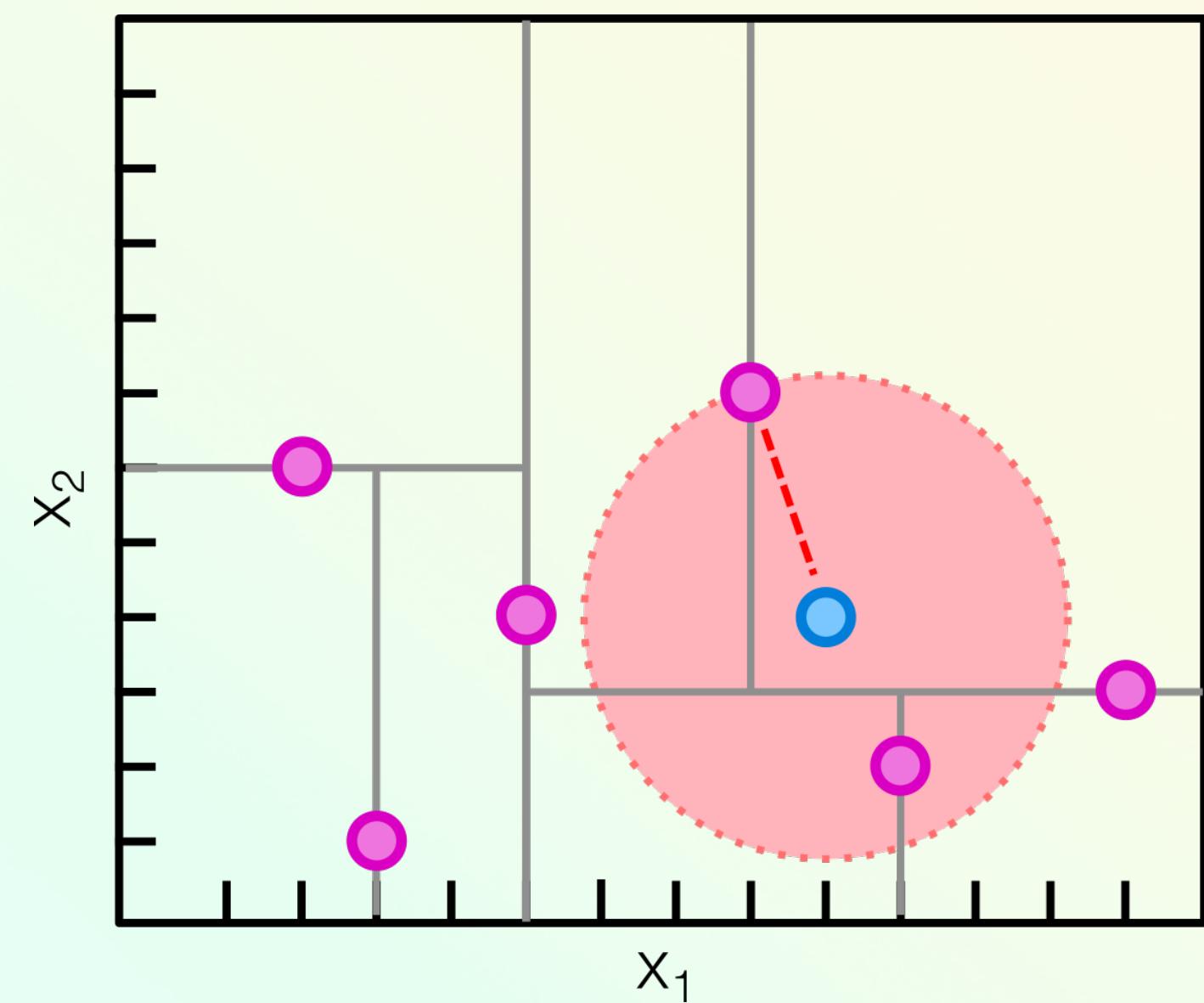
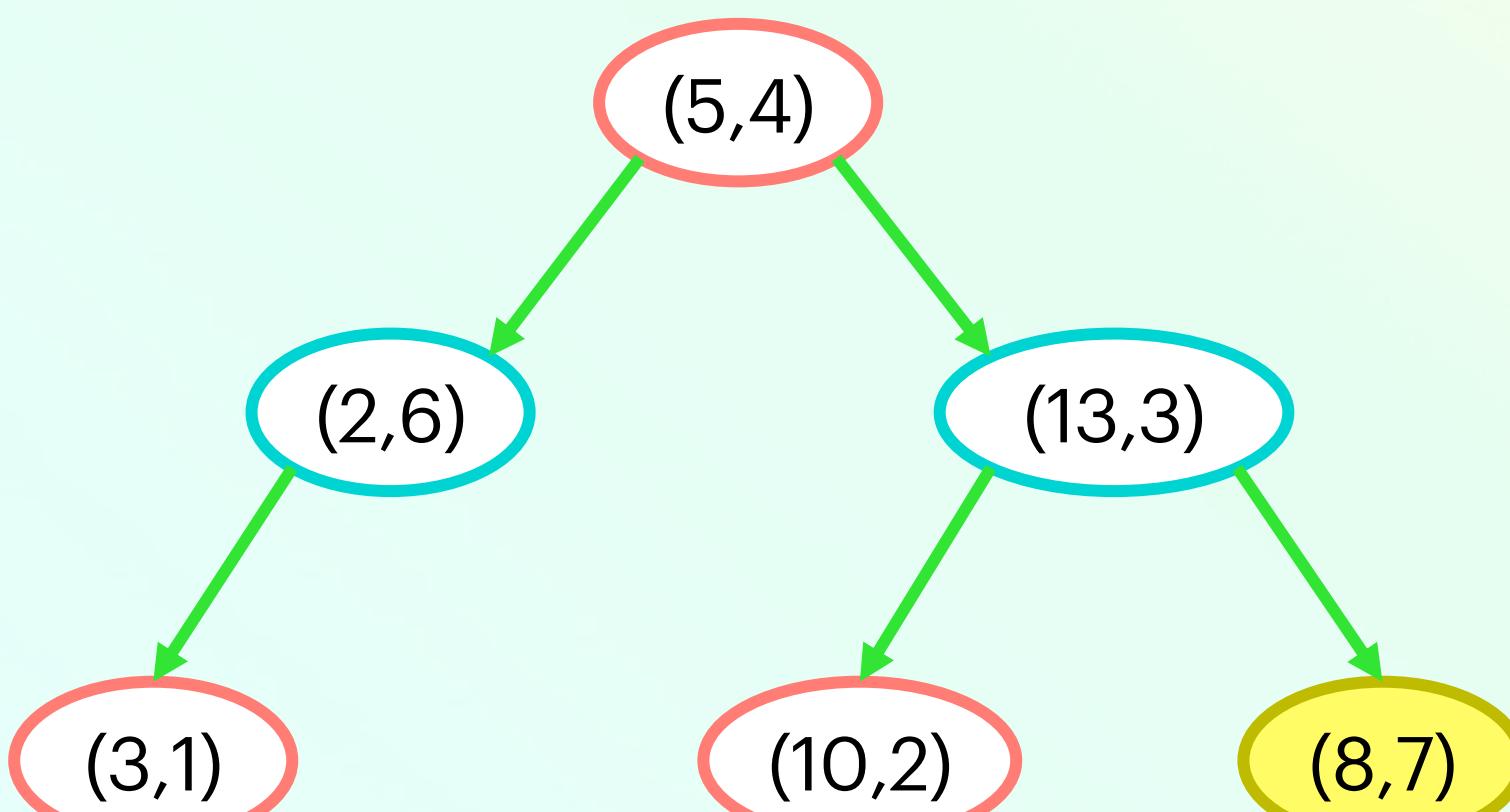
Se encuentra la hoja en el que el nodo se encuentra y se calcula la distancia...



# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

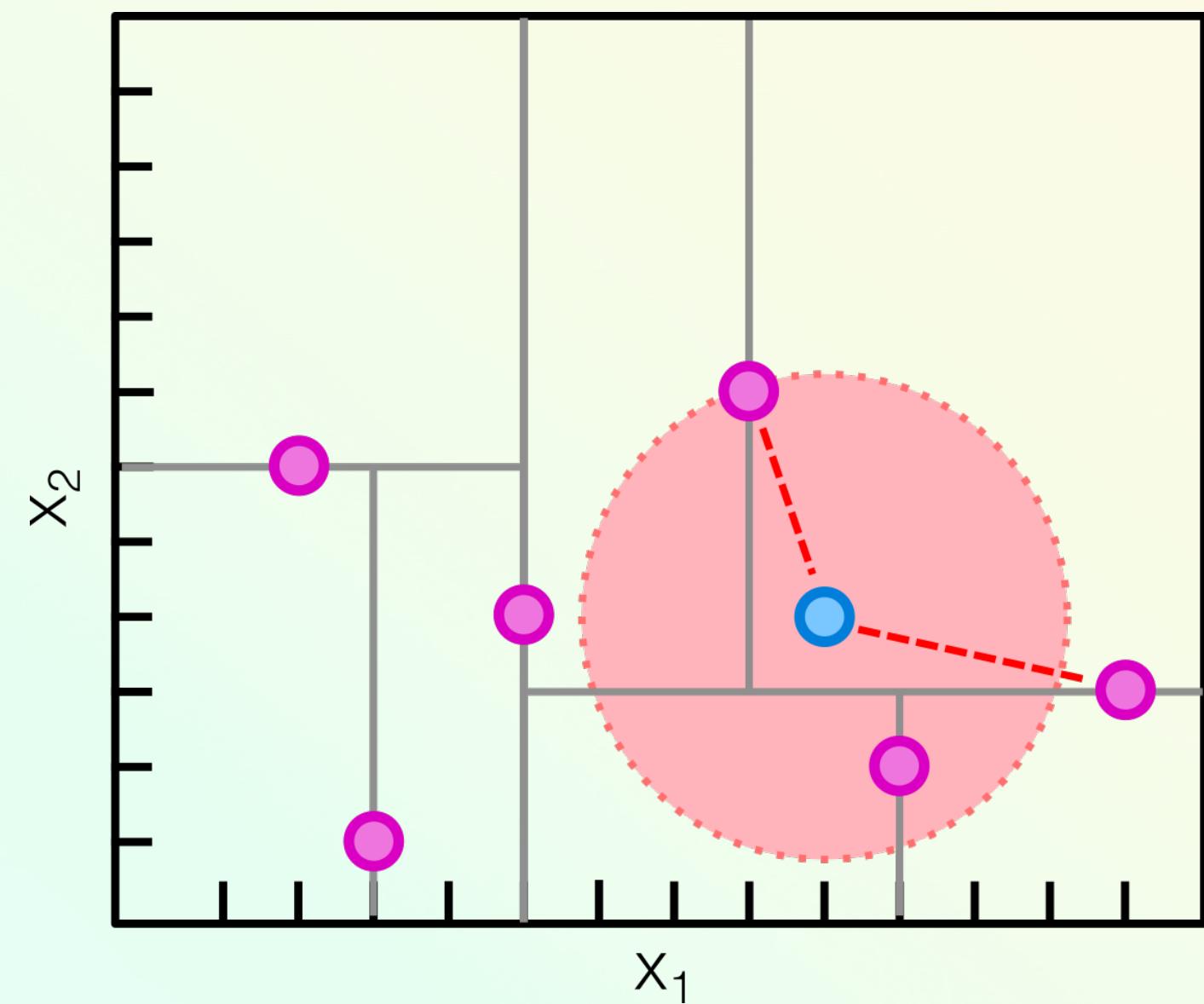
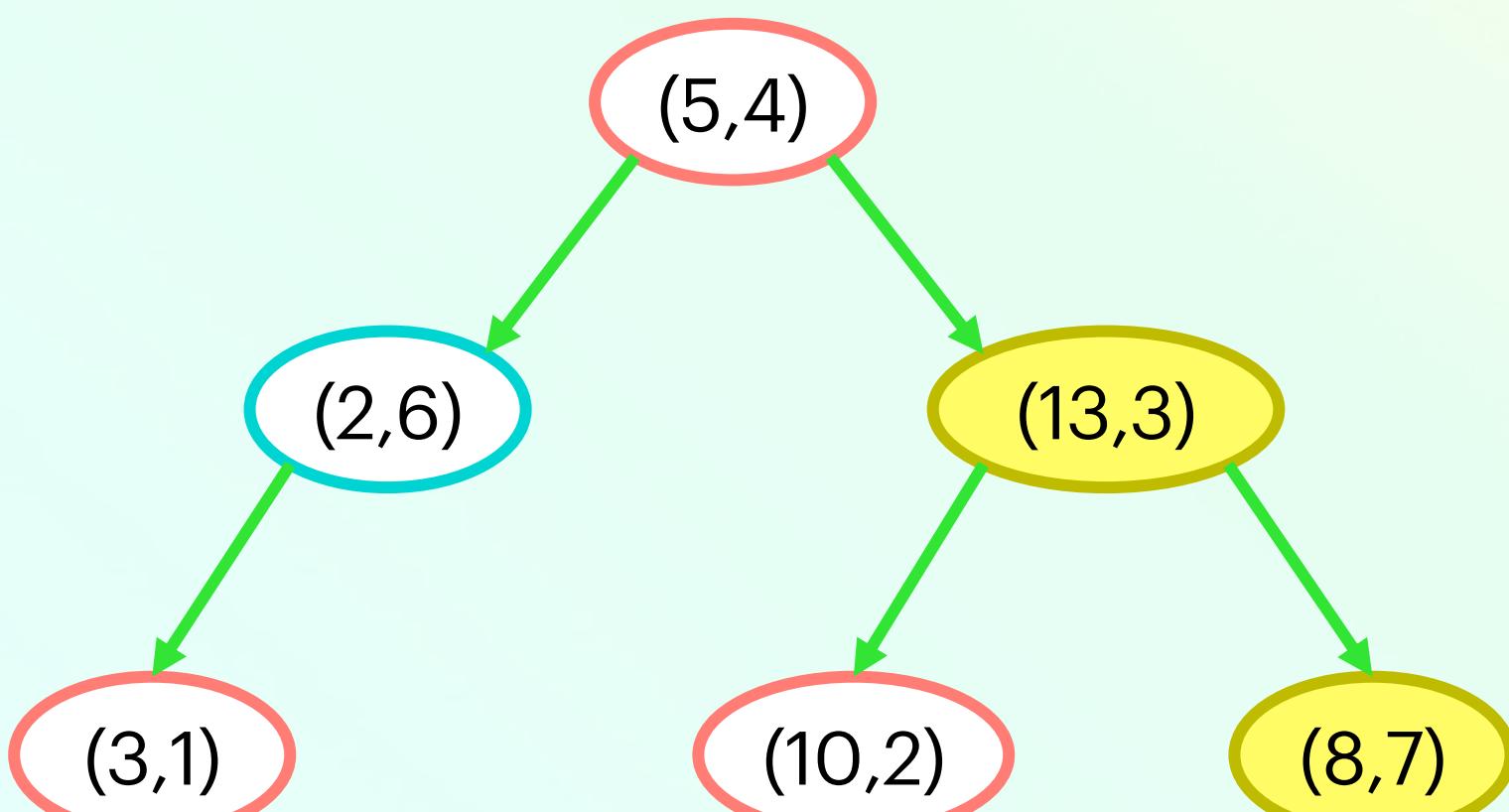
Luego se sube al árbol en función de si los ejes de cortes están a una menor distancia de la distancia calculada y se calcula la distancia...



# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

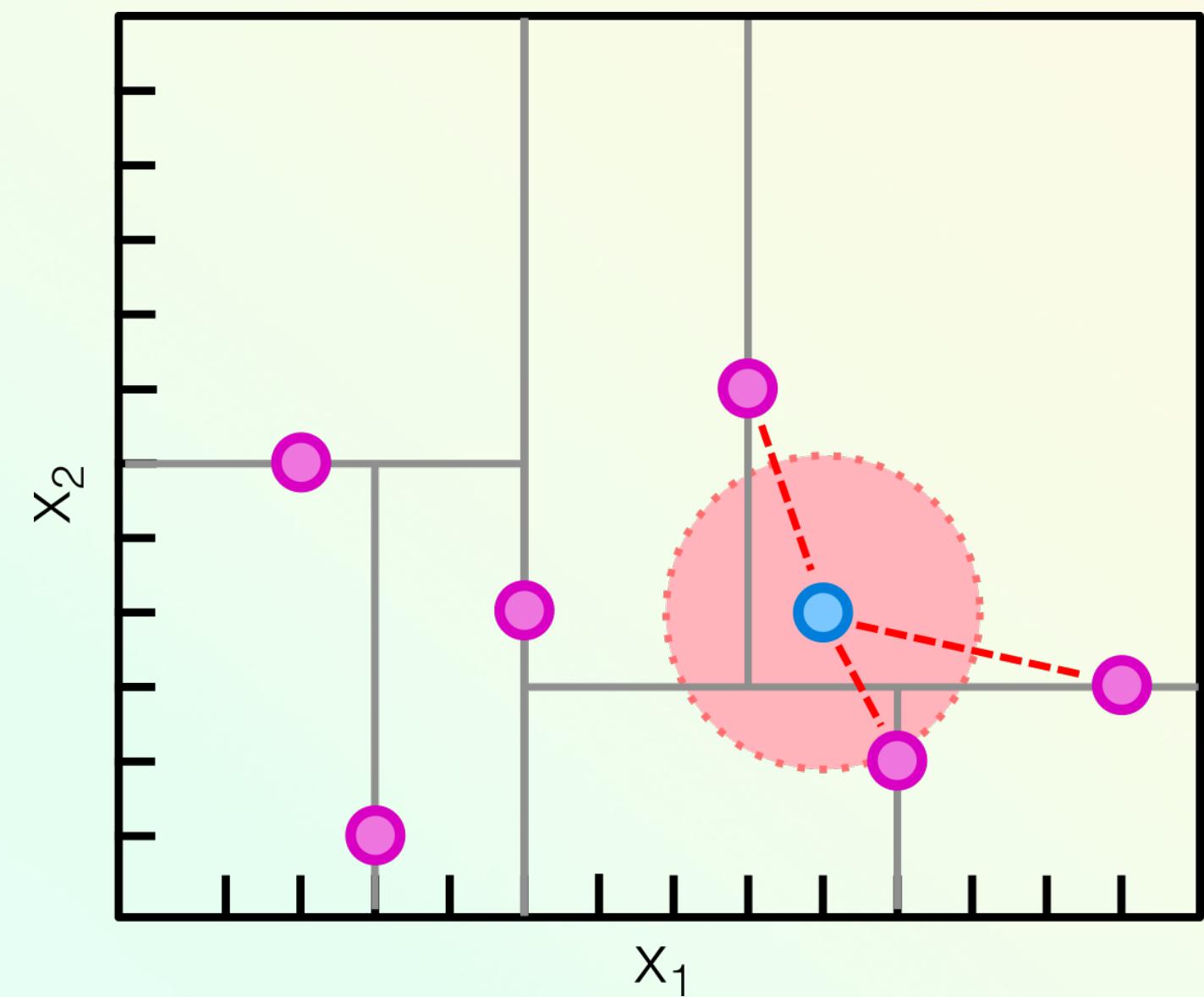
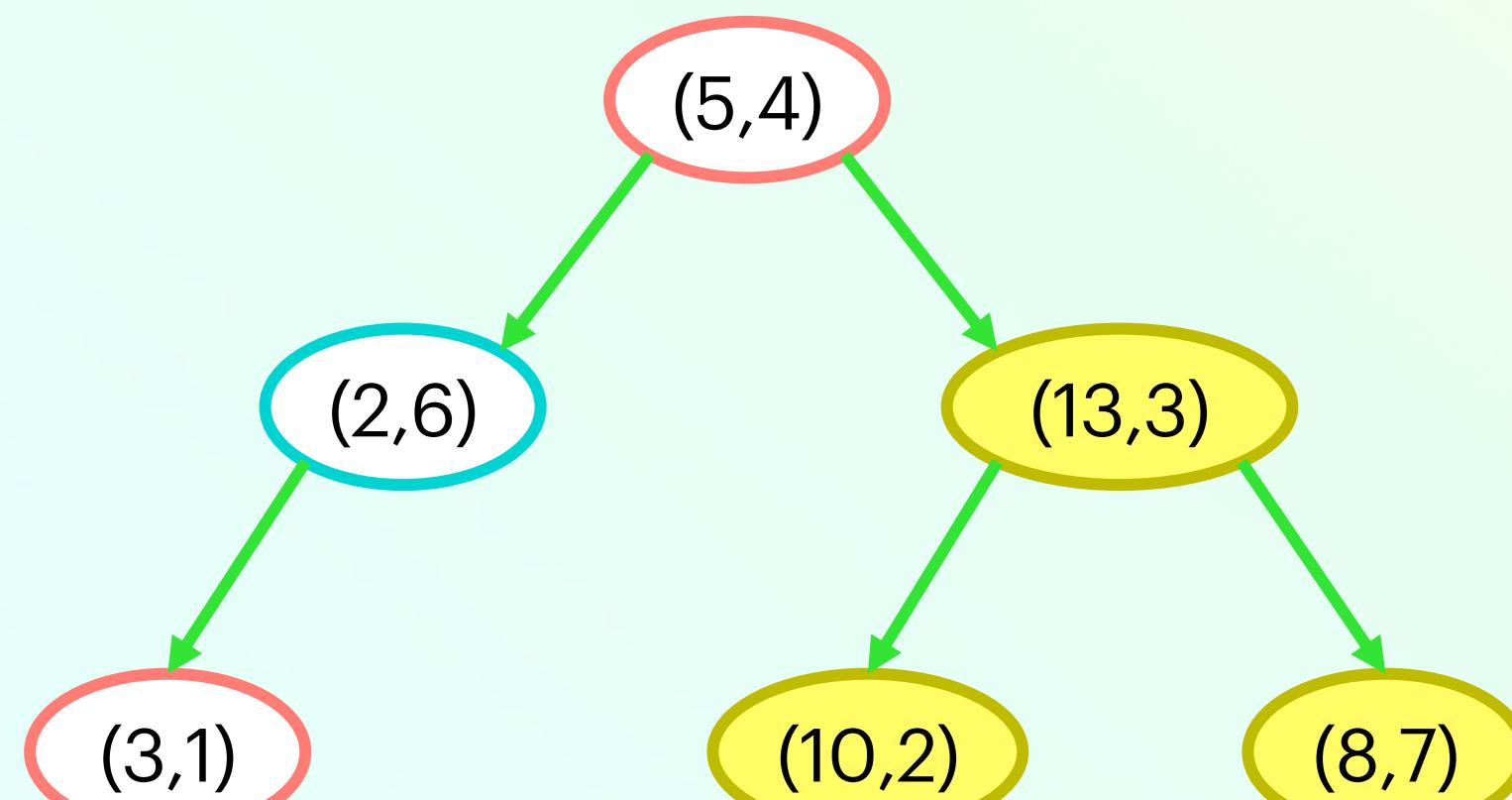
Luego se sube al árbol en función de si los ejes de cortes están a una menor distancia de la distancia calculada y se calcula la distancia...



# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

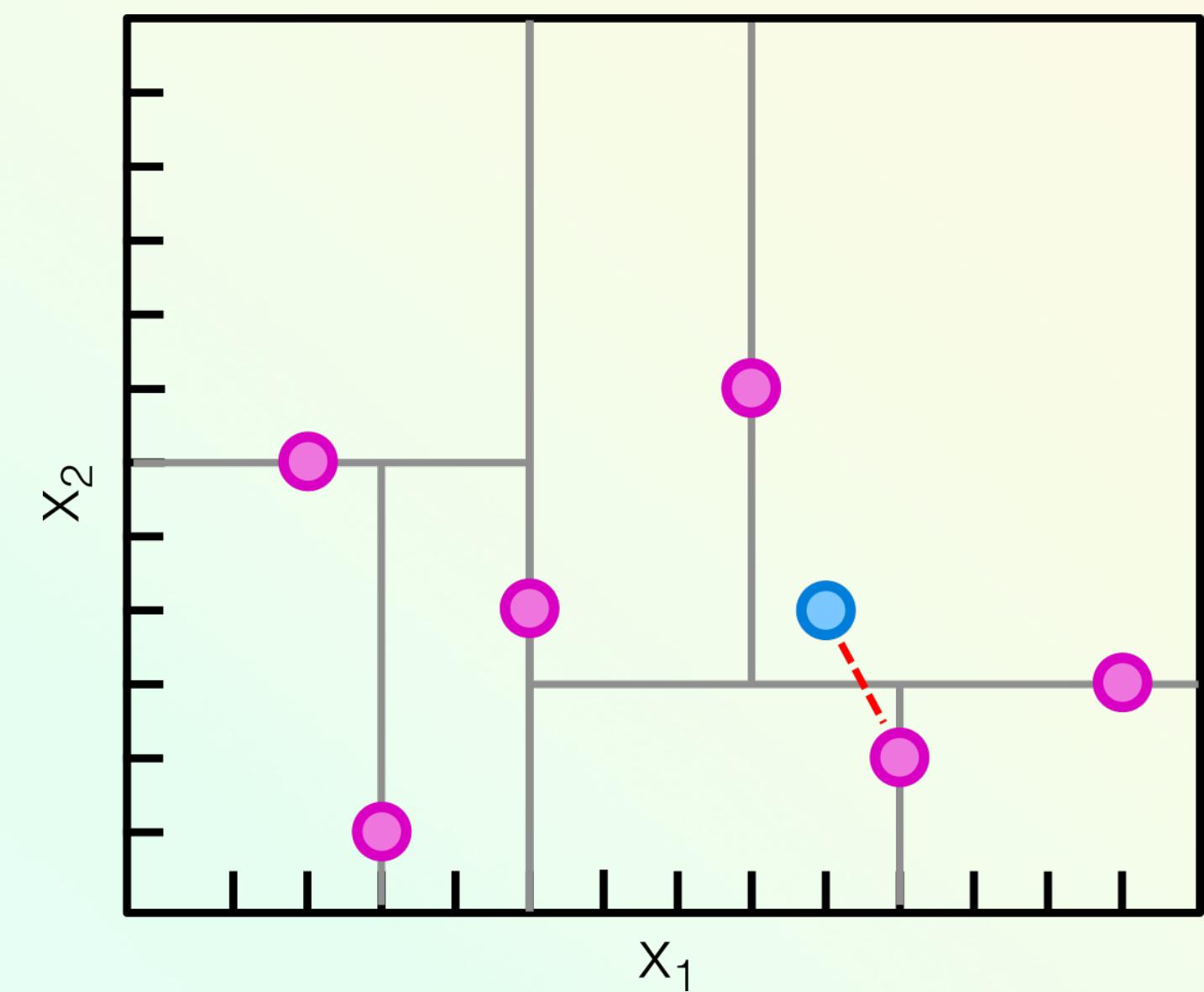
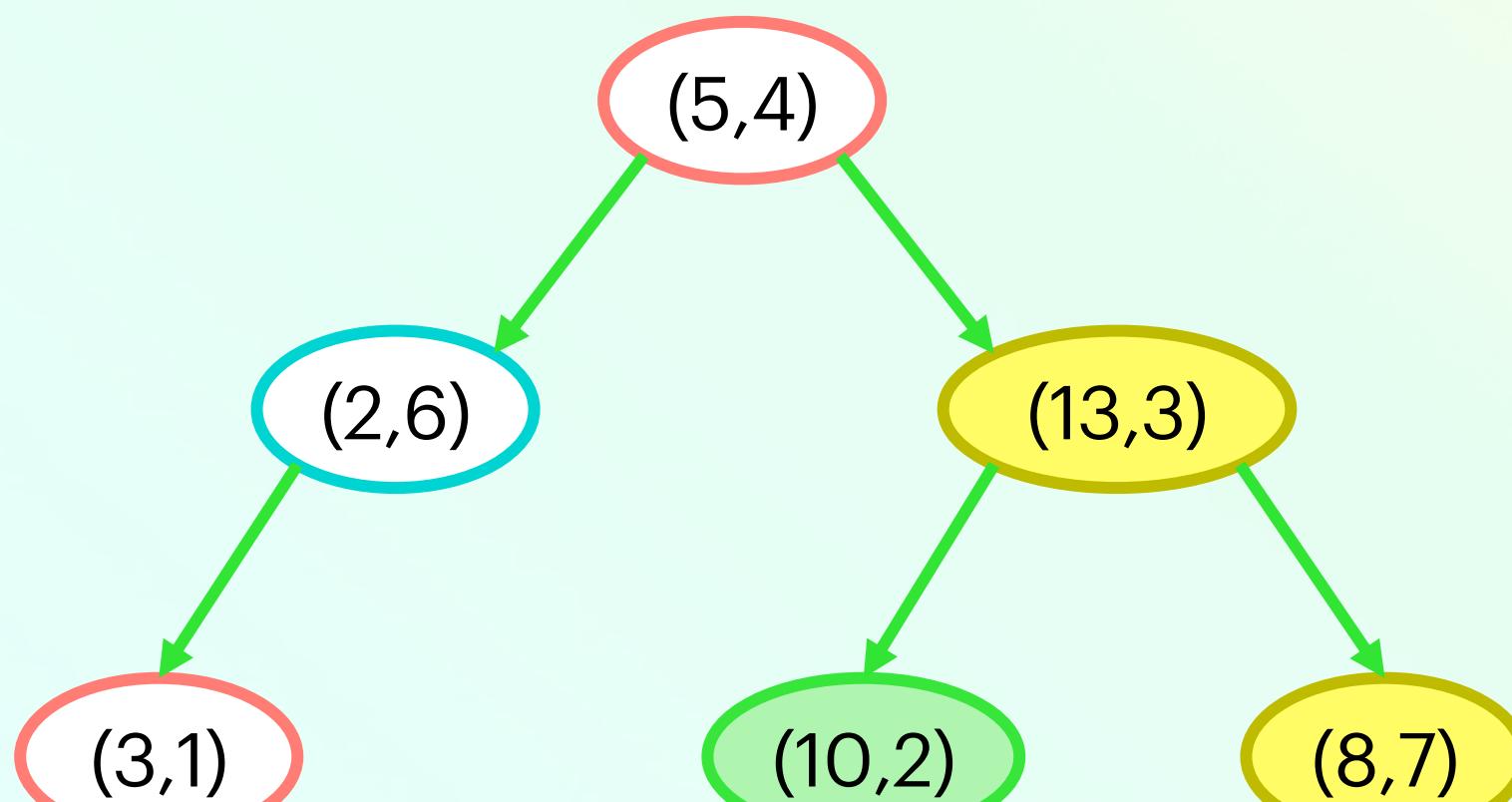
Como esta distancia es mayor a la anterior, la descartamos, y avanzamos al otro nodo, que también estamos cortando su separación...



# ENCONTRANDO A LOS VECINOS

	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

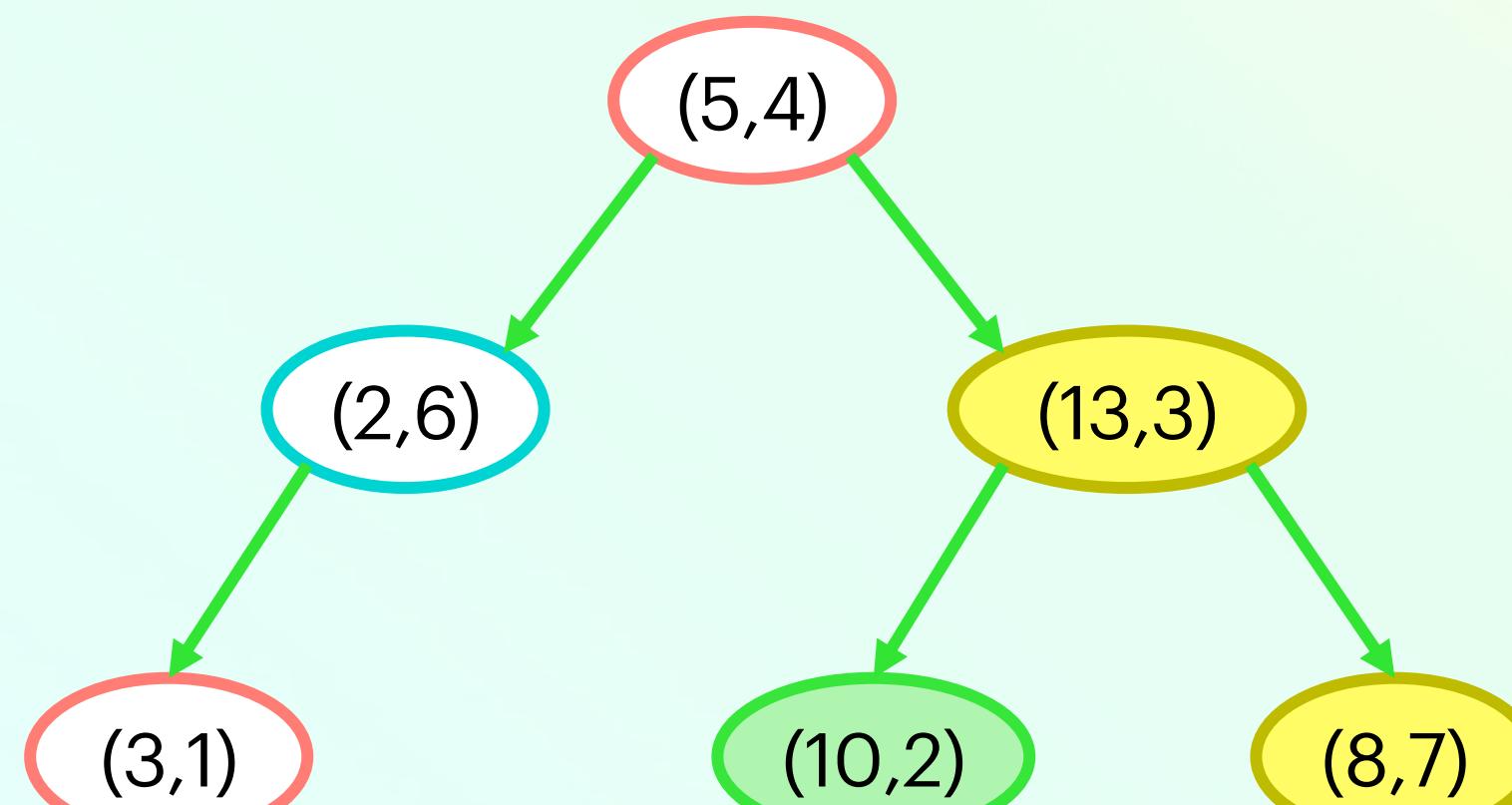
Como esta distancia es menor, la guardamos, y como no cortamos más separaciones, termina nuestra búsqueda y encontramos el más cercano solo haciendo tres cálculos.



# ENCONTRANDO A LOS VECINOS

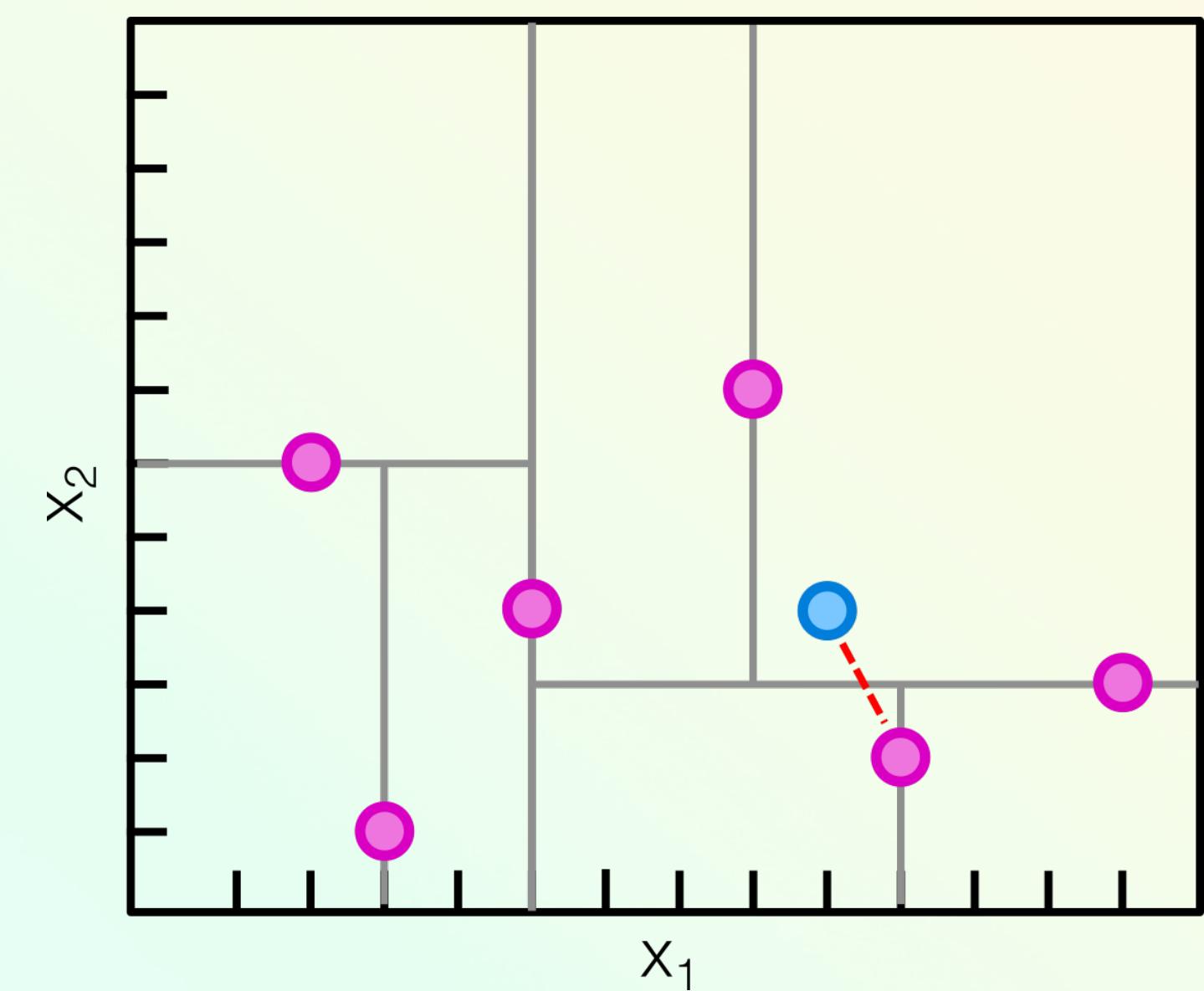
	$x_1$	$x_2$
P1	2	6
P2	3	1
P3	5	4
P4	8	7
P5	10	2
P6	13	3

Como esta distancia es menor, la guardamos, y como no cortamos más separaciones, termina nuestra búsqueda y encontramos el más cercano solo haciendo tres cálculos.



Este algoritmo con pequeño cambio se puede encontrar los K-vecinos. Y el entrenamiento es el armado de este árbol.

El caso general, las regiones pueden haber más de un punto, eso lo hace menos complejo, pero en cada celda se hace una búsqueda de fuerza bruta, pero con muchos menos puntos.



# ENCONTRANDO A LOS VECINOS

Este algoritmo funciona bien para baja dimensionalidades, pero en grandes dimensiones sufre del efecto de la [maldición de dimensión](#). En grandes dimensiones y tal como se estructura el árbol, se termina en casos tan malos como la búsqueda en fuerza bruta.

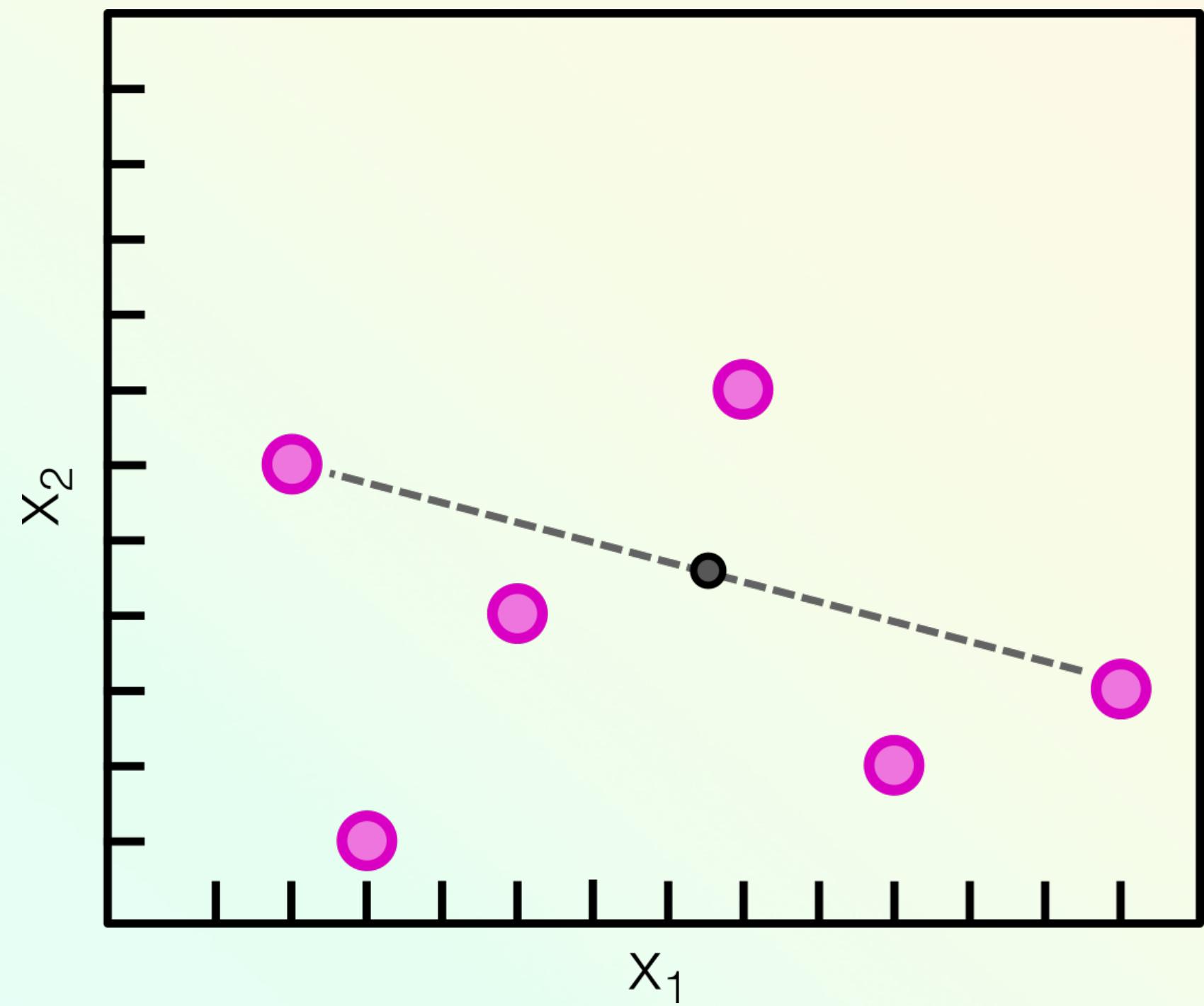
Brevemente, la maldición de dimensionalidad advierte que a medida que aumenta el número de dimensiones, los datos se vuelven más dispersos y los algoritmos basados en la distancia pueden perder precisión o eficiencia.

Para intentar resolver esto, se utiliza **Ball Tree**, el cual, en vez de usar regiones divididas a lo largo de los ejes cartesianos, los Ball Tree dividen los datos en una serie de hiperesferas anidadas. Esto hace que la construcción del árbol sea más costosa, pero resulta que puede ser muy eficiente en datos en dimensiones muy altas.

# ENCONTRANDO A LOS VECINOS

Ball Tree divide recursivamente los datos en hiperesferas.

Inicialmente se busca los dos puntos más alejados.

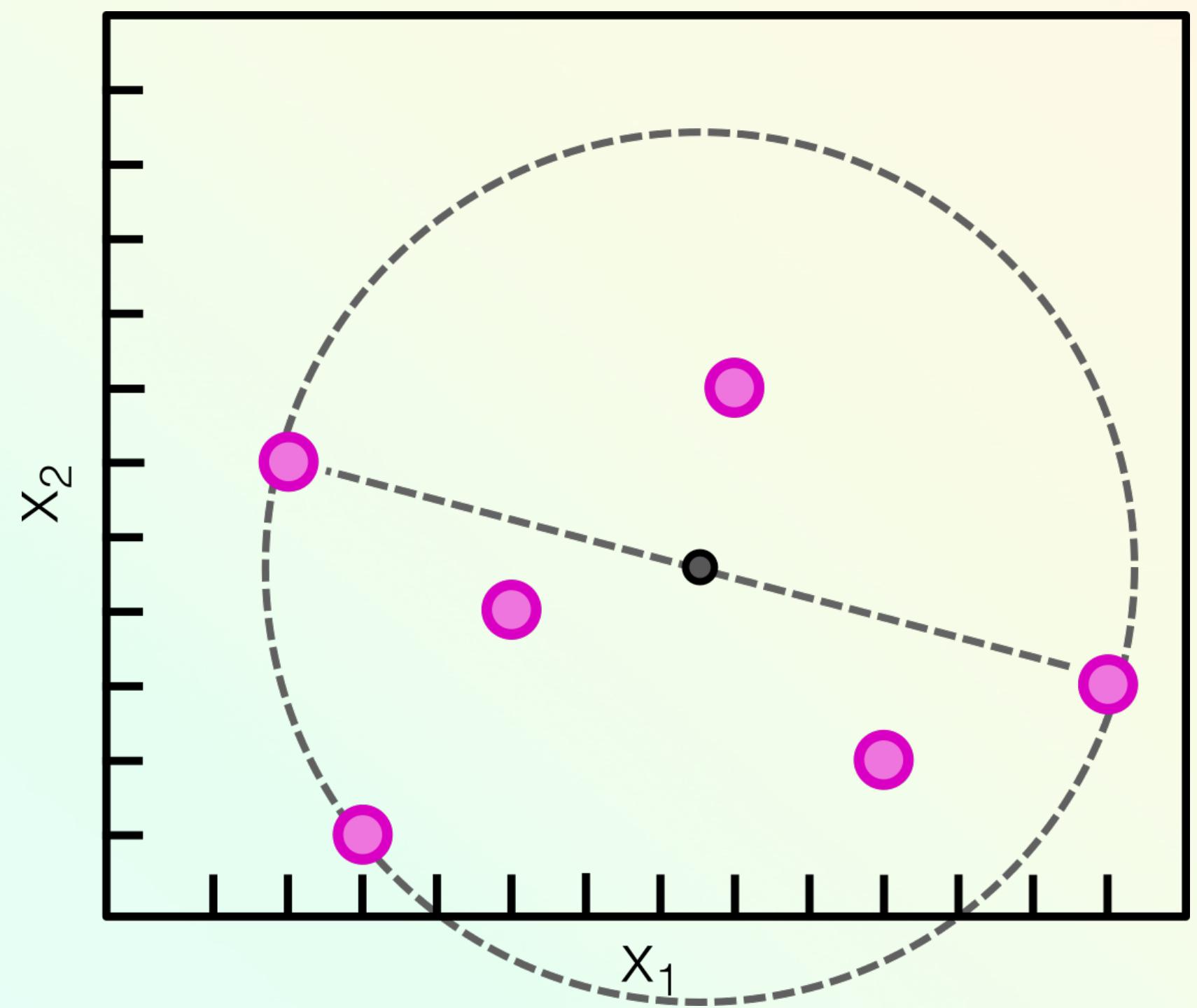


# ENCONTRANDO A LOS VECINOS

Ball Tree divide recursivamente los datos en hiperesferas.

Inicialmente se busca los dos puntos más alejados.

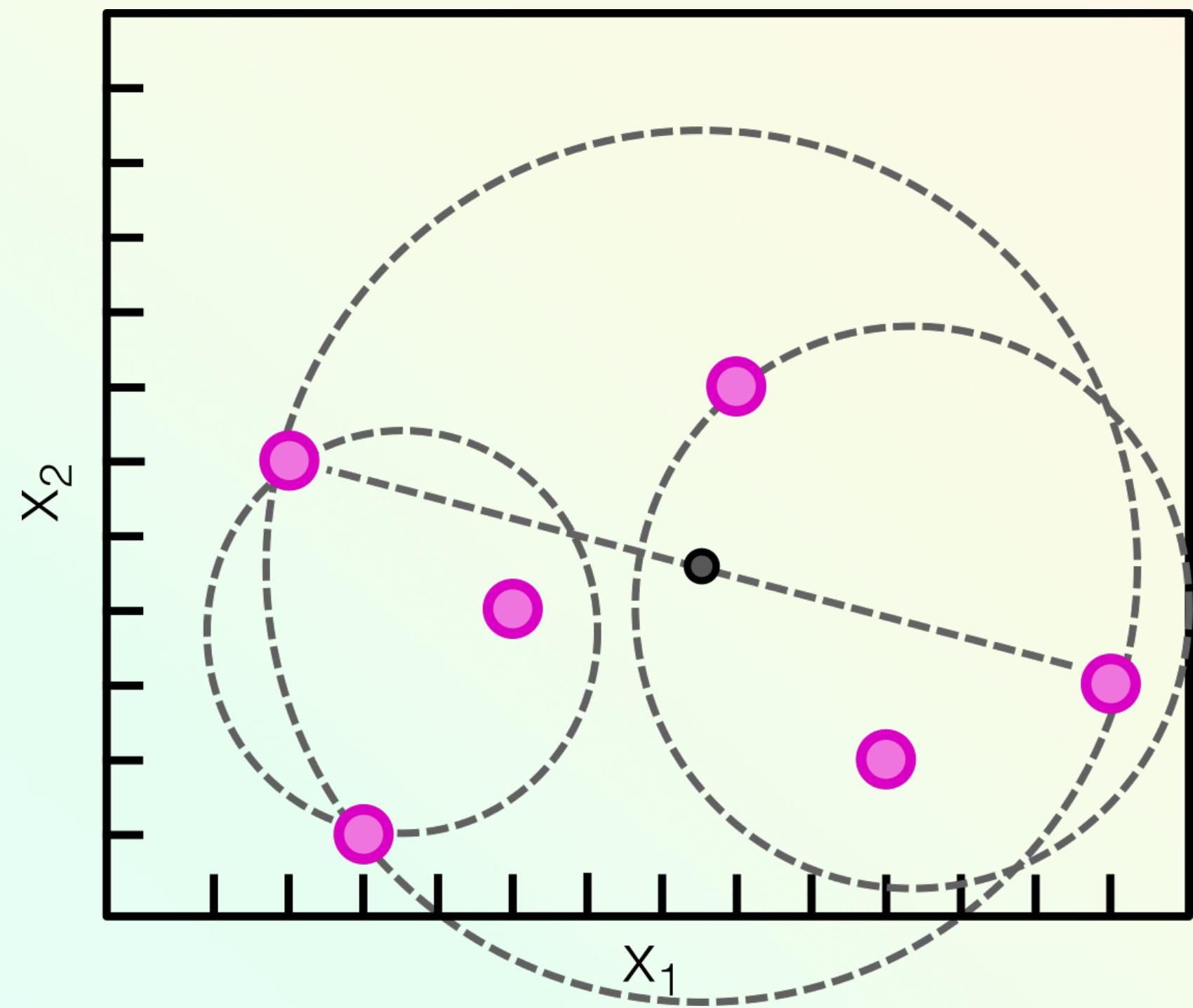
Luego se construye una circunferencia entre ellos.



# ENCONTRANDO A LOS VECINOS

Ball Tree divide recursivamente los datos en hiper-esferas.

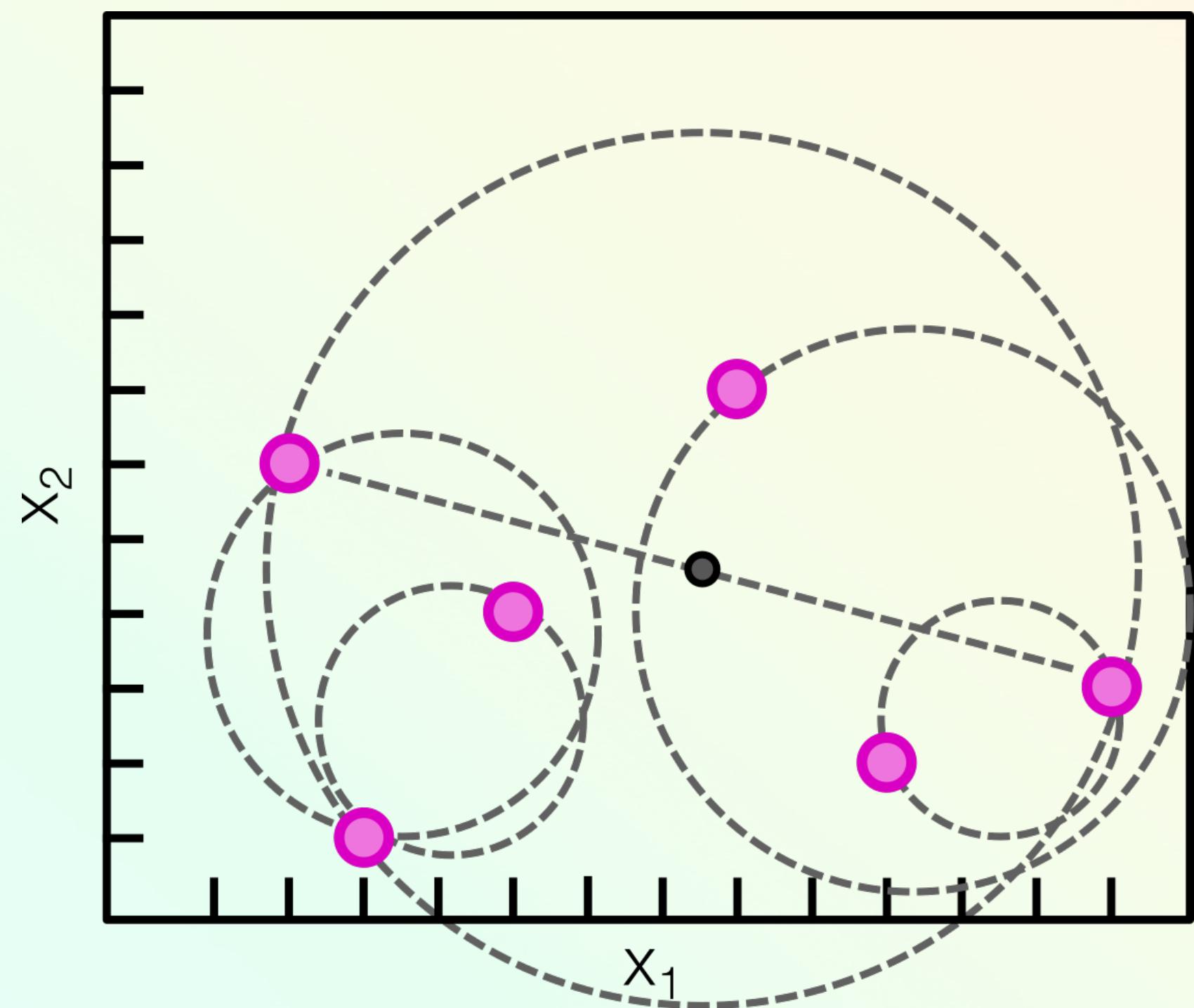
Similar a T-D Tree, se separa en mitades y mitades, pero ahora lo hacemos en esferas en vez de los ejes cartesianos.



# ENCONTRANDO A LOS VECINOS

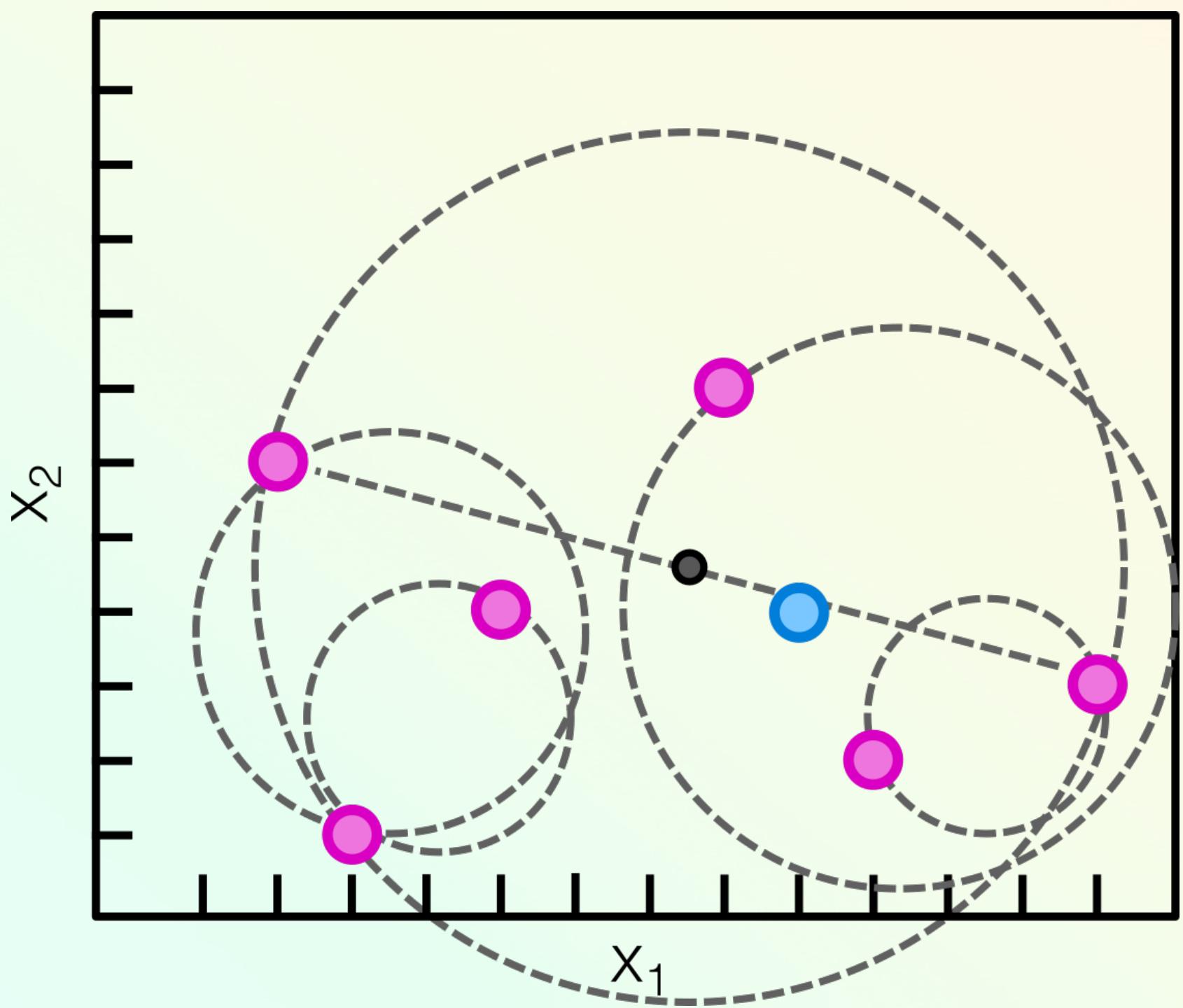
Ball Tree divide recursivamente los datos en hiper-esferas.

Similar a T-D Tree, se separa en mitades y mitades, pero ahora lo hacemos en esferas en vez de los ejes cartesianos.



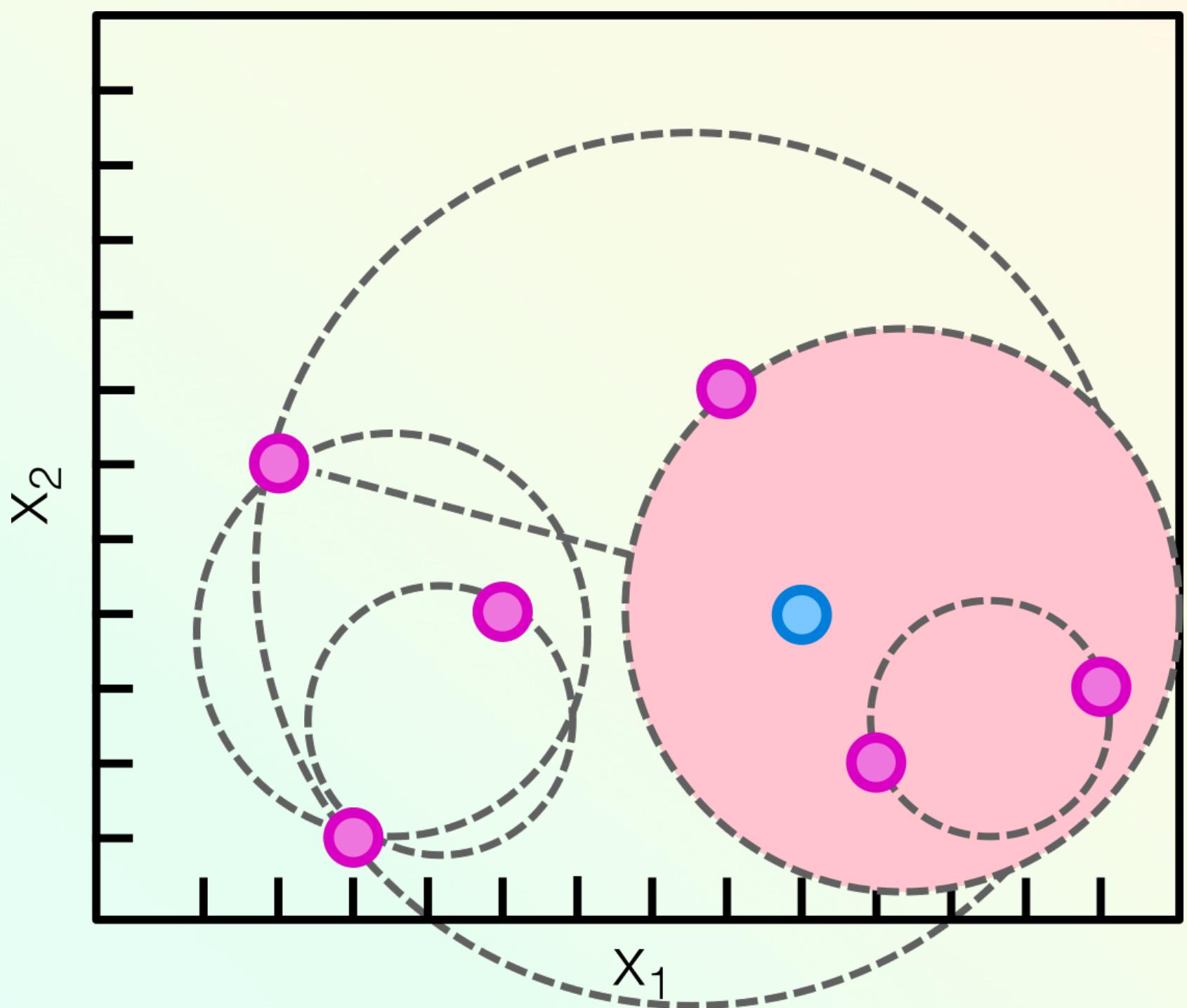
# ENCONTRANDO A LOS VECINOS

Una vez construido el árbol, cuando tenemos un nuevo dato, vemos en qué esfera está adentro (la más pequeña).



# ENCONTRANDO A LOS VECINOS

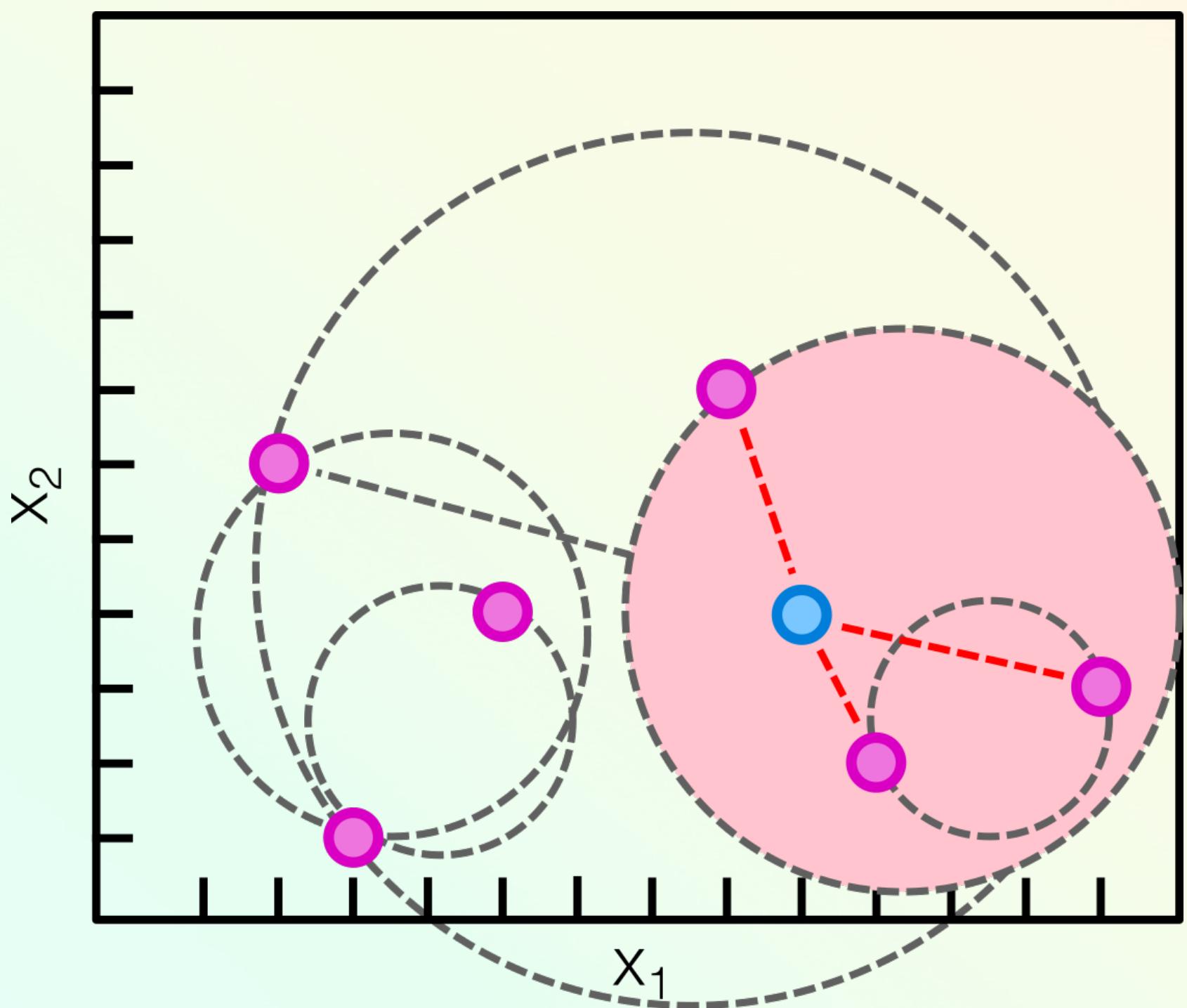
Una vez construido el árbol, cuando tenemos un nuevo dato, vemos en que esfera está adentro (la más pequeña).



# ENCONTRANDO A LOS VECINOS

Una vez construido el árbol, cuando tenemos un nuevo dato, vemos en qué esfera está adentro (la más pequeña).

Y en esta esfera calculamos las distancias de quienes estén en esa esfera.

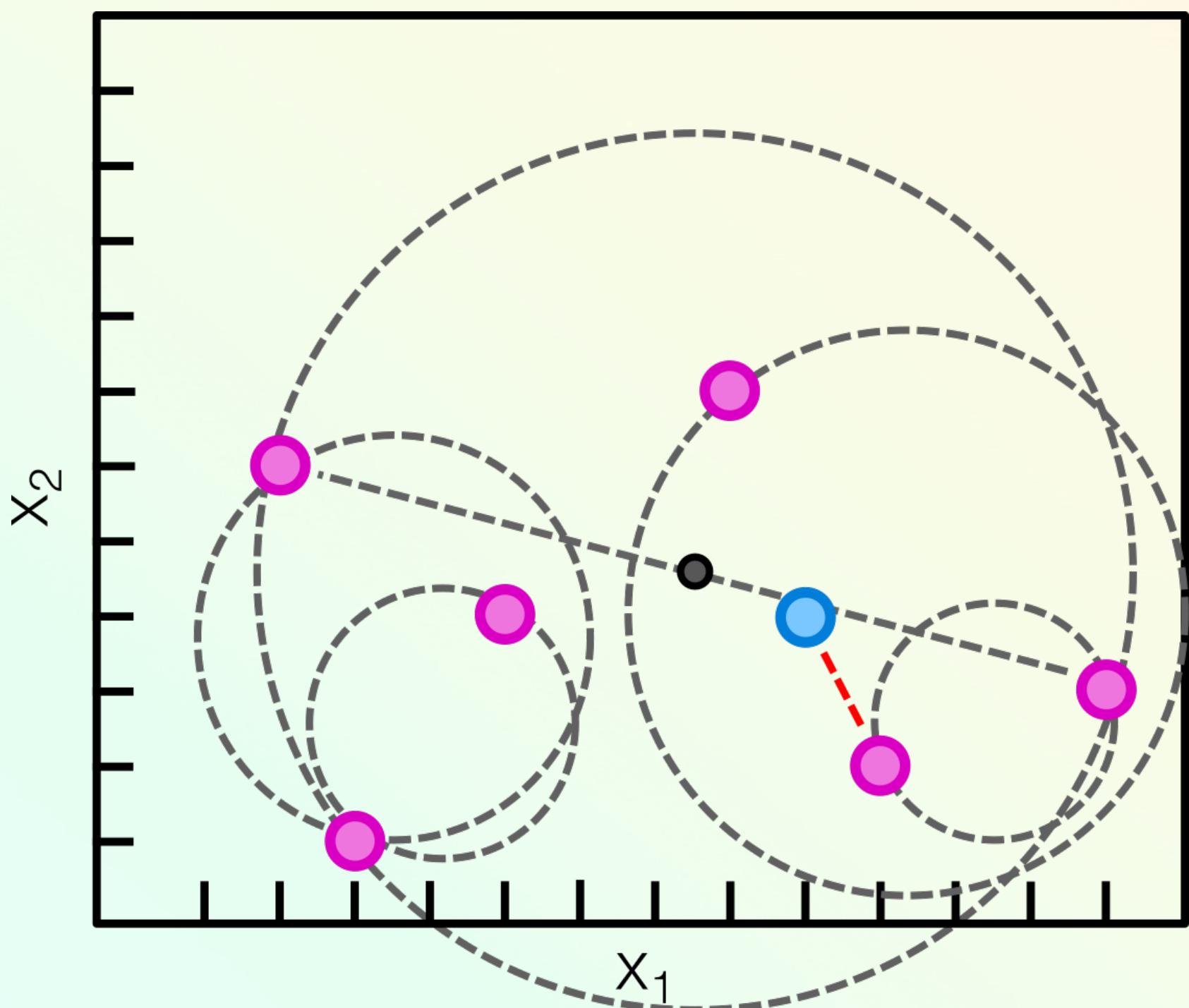


# ENCONTRANDO A LOS VECINOS

Una vez construido el árbol, cuando tenemos un nuevo dato, vemos en qué esfera está adentro (la más pequeña).

Y en esta esfera calculamos las distancias de quienes estén en esa esfera.

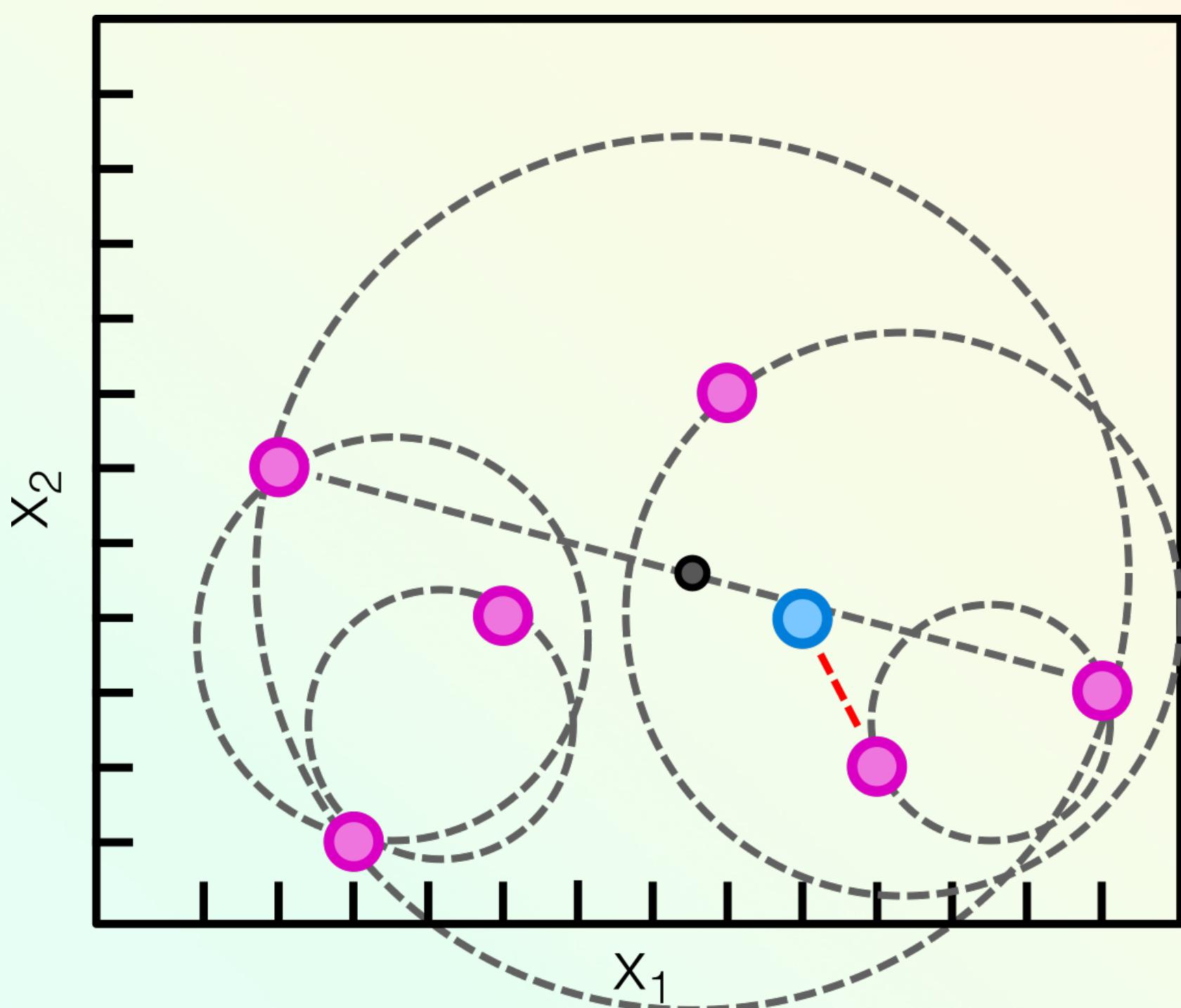
Esta estructura es más pesada y lleva más tiempo de entrenamiento para almacenamiento y para navegarse el árbol cuando buscamos un parámetro nuevo.



# ENCONTRANDO A LOS VECINOS

Scikit-Learn implementa esto tres tipos de estructuras para k-NN (y otros algoritmos como k-means), si se elige que automáticamente elija, lo que hace esta librería es:

- Si son pocos datos, usa fuerza bruta.
- Si son muchos, pero de baja dimensión, usa K-D Tree.
- Alta dimensionalidad, utiliza Ball Tree.



# **REPASO DE MÉTRICAS DE EVALUACIÓN**

# MÉTRICAS DE EVALUACIÓN

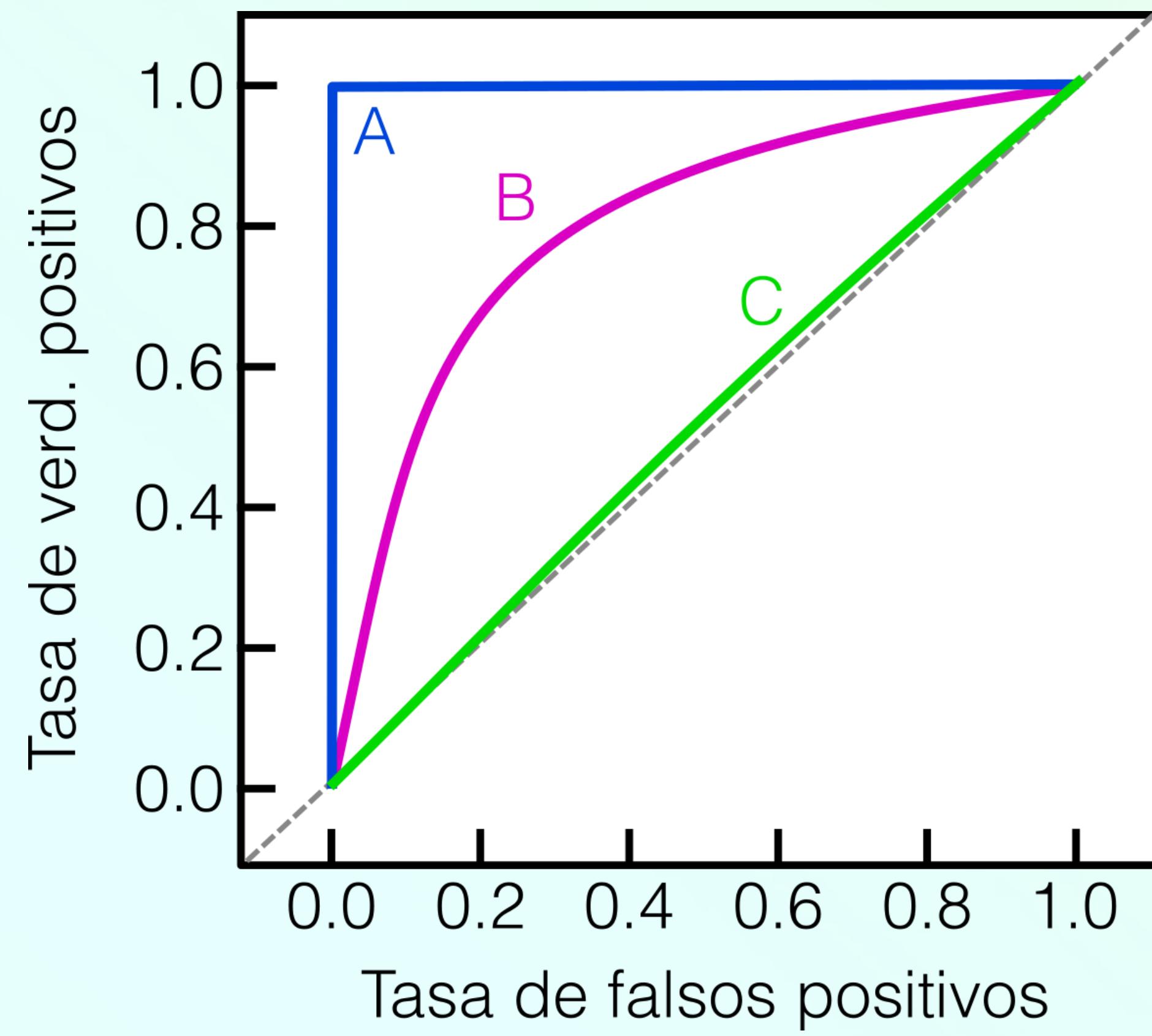
## MATRIZ DE CONFUSIÓN

		Valores actuales	
		1	0
Predicción	1	<b>Verdadero positivo (TP)</b>	<b>Falso positivo (FP)</b>
	0	<b>Falso negativo (FN)</b>	<b>Verdadero negativo (TN)</b>

# MÉTRICAS DE EVALUACIÓN

- **Sensibilidad:**  $TPR = \frac{TP}{P} = \frac{TP}{TP + FN} = 1 - FNR$
- **Especificidad:**  $TNR = \frac{TN}{N} = \frac{TN}{TN + FP} = 1 - FPR$
- **Exactitud:**  $ACC = \frac{TP + TN}{P + N}$
- **Exactitud balanceada:**  $BA = \frac{TPR + TNR}{2}$
- **Precisión:**  $\text{Precision} = \frac{TP}{TP + FP}$
- **Recuperación:**  $\text{Recall} = \frac{TP}{TP + FN}$
- **F1-score o F $\beta$ -score:**  $F_{\beta} = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$

# CURVA ROC



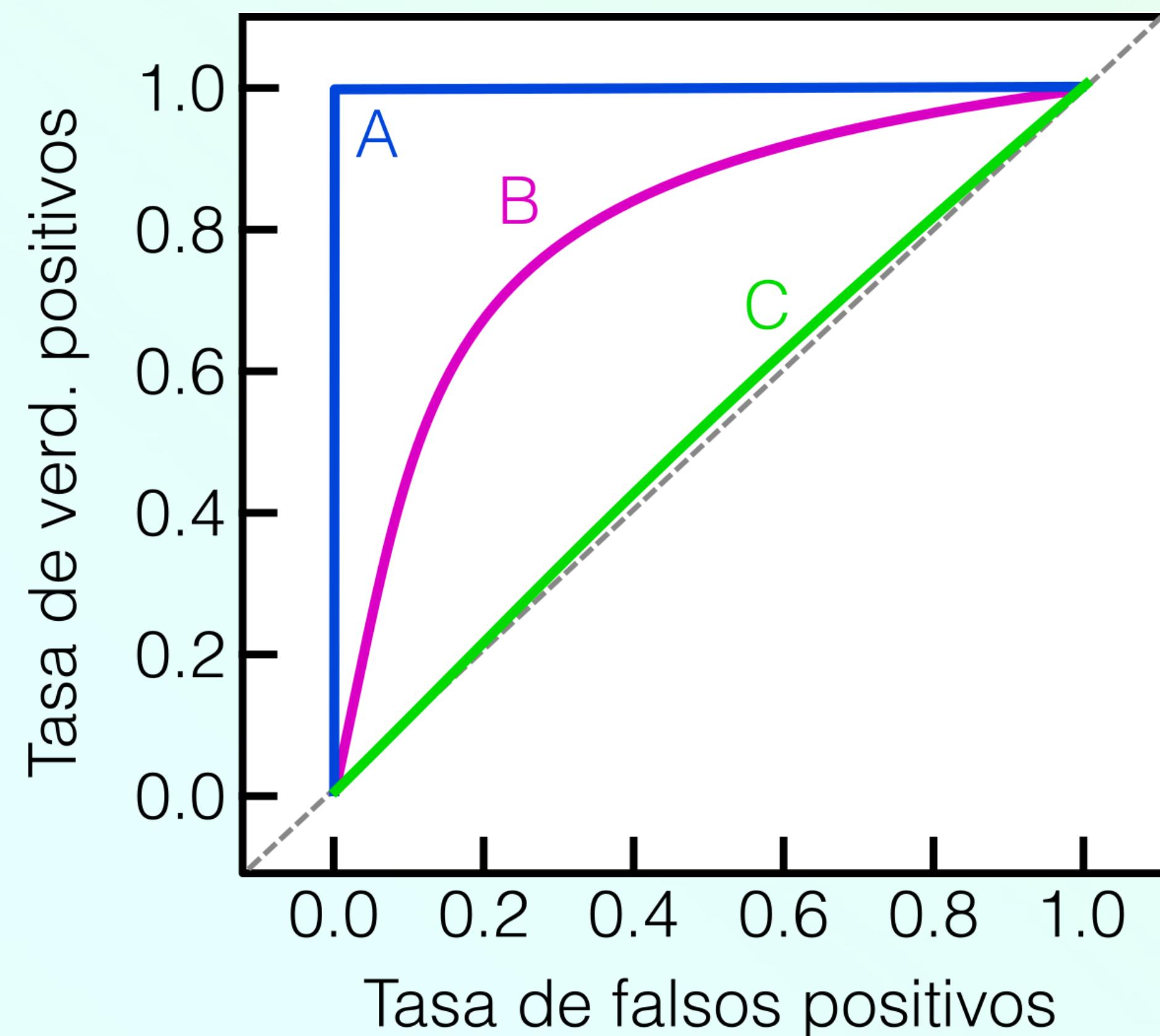
Siempre se arranca de umbral 1, donde la TPR es 0 y TFP es 0 y termina en 0 donde TFP es 1 y TPR es 1.

- **A** es la curva de un clasificador perfecto
- **B** es la curva de un clasificador estándar.
- **C** es la curva de un clasificador que adivina (el peor caso).

La curva ROC me permite encontrar el valor umbral que mejor resultado me dé.

Además, me permite comparar clasificadores sin preocuparme del valor umbral elegido.

# CURVA ROC



Si quiero bajar a una métrica a esta curva, podemos calcular el área bajo la curva (AUC).

- **A** tendrá un  $AUC = 1$
- **B** tendrá un  $0.5 < AUC < 1$
- **C** tendrá un  $AUC = 0.5$

**k-NN no tiene curva ROC, dado que no puede darnos salida probabilística, solo nos da un punto.**

**VAMOS A PRÁCTICAR UN POCO...**

# **MÉTODOS DE AJUSTE DE LOS HIPER-PARAMETROS**

# MÉTODOS DE AJUSTE DE LOS HIPER-PARÁMETROS

En Inteligencia Artificial se vió que validación cruzada nos sirve para ayudarnos a buscar los hiper-parámetros que mejor se nos ajustan a nuestros modelos, permitiendo mantener la **generalidad**.

Pero por sí solo, no alcanza, necesitamos de alguna forma *movernos* por el espacio de búsqueda.

Una búsqueda consiste en:

- Un modelo (de regresión o clasificación)
- Un espacio de parámetros
- Un método de búsqueda o de muestreo de candidatos
- Un esquema de validación cruzada
- Una función de puntaje

# MÉTODOS DE AJUSTE DE LOS HIPER-PARÁMETROS

En Inteligencia Artificial se vió que validación cruzada nos sirve para ayudarnos a buscar los hiper-parámetros que mejor se nos ajustan a nuestros modelos, permitiendo mantener la **generalidad**.

Pero por sí solo, no alcanza, necesitamos de alguna forma *movernos* por el espacio de búsqueda.

Una búsqueda consiste en:

- Un modelo (de regresión o clasificación)
- Un espacio de parámetros
- **Un método de búsqueda o de muestreo de candidatos**
- Un esquema de validación cruzada
- Una función de puntaje

# MÉTODOS DE AJUSTE DE LOS HIPER-PARÁMETROS

Dos métodos de búsqueda típicos:

**Búsqueda de grilla:** Busca exhaustiva de todas las combinaciones de los parámetros. Es el más completo pero el más ineficiente.

**Búsqueda aleatoria:** Busca aleatoriamente tomando datos del espacio de combinaciones de los parámetros, bajo una distribución aleatoria dada. Termina dado una cierta cantidad arbitraria de iteraciones.

# MÉTODOS DE AJUSTE DE LOS HIPER-PARÁMETROS

Una mejora que se puede aplicar a ambos métodos, a expensas de una mayor duración, es incorporar la reducción a la mitad sucesiva (successive halving).

La idea es tener una especie de torneo. En el cual, se arranca buscando los hiper-parámetros pero usando un subset de entrenamiento chico.

Una vez que termina la búsqueda, se elige la mitad de las combinaciones que mejores parámetros dieron. Con esa mitad, se aumenta el set de entrenamiento y este proceso se repite, hasta que quede uno.

**VAMOS A PRÁCTICAR UN POCO...**

# **MÉTODOS MÁS AVANZADOS**

# MÉTODOS AVANZADOS

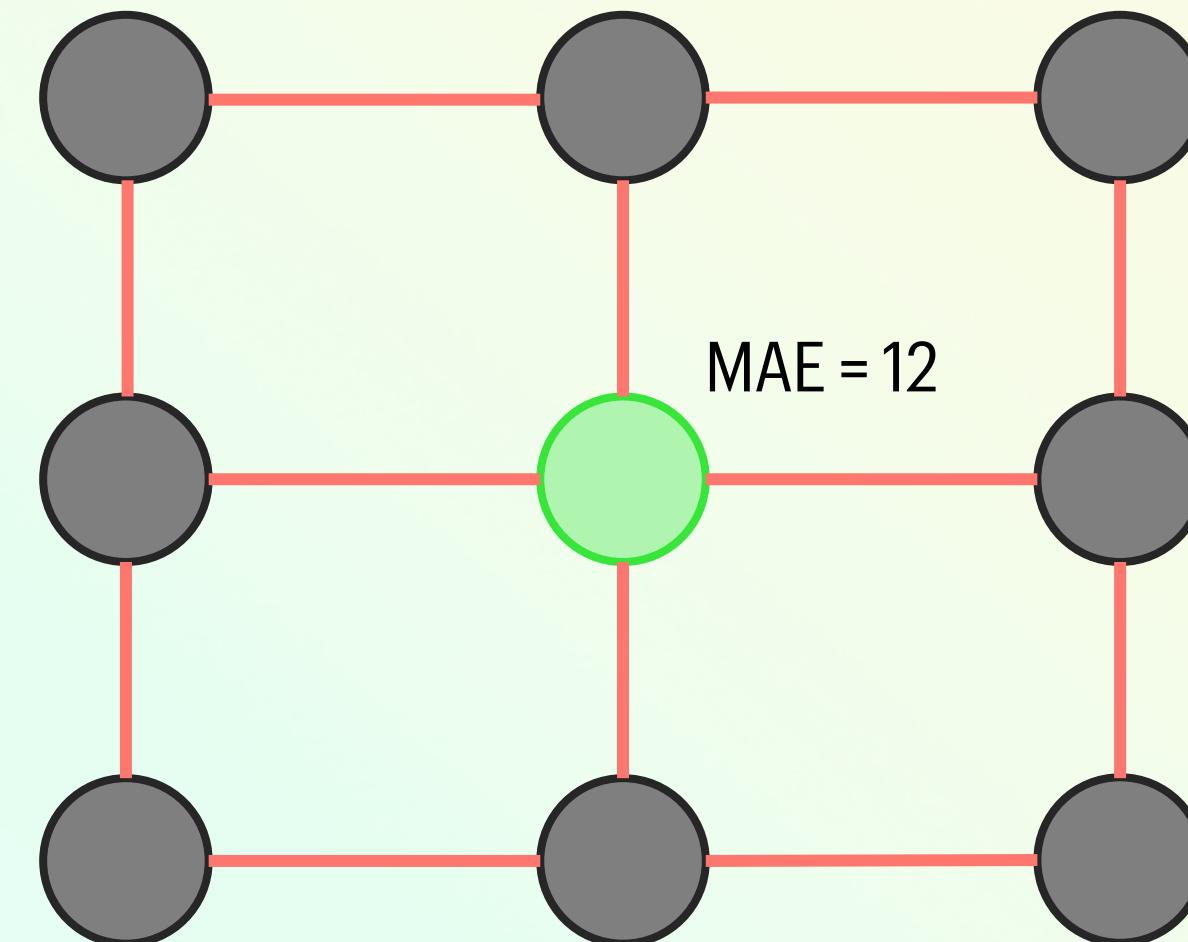
Podemos crear nuestros métodos que sean más avanzados. En inteligencia artificial habían visto algoritmos de búsquedas. Podemos usar estos para buscar hiper-parámetros:

- Gradiente descendente o Ascendente
- Simulated annealing
- Búsqueda Local Beam
- Algoritmos Genéticos

# MÉTODOS AVANZADOS

Podemos crear nuestros métodos que sean más avanzados. En inteligencia artificial habían visto algoritmos de búsquedas. Podemos usar estos para buscar hiper-parámetros:

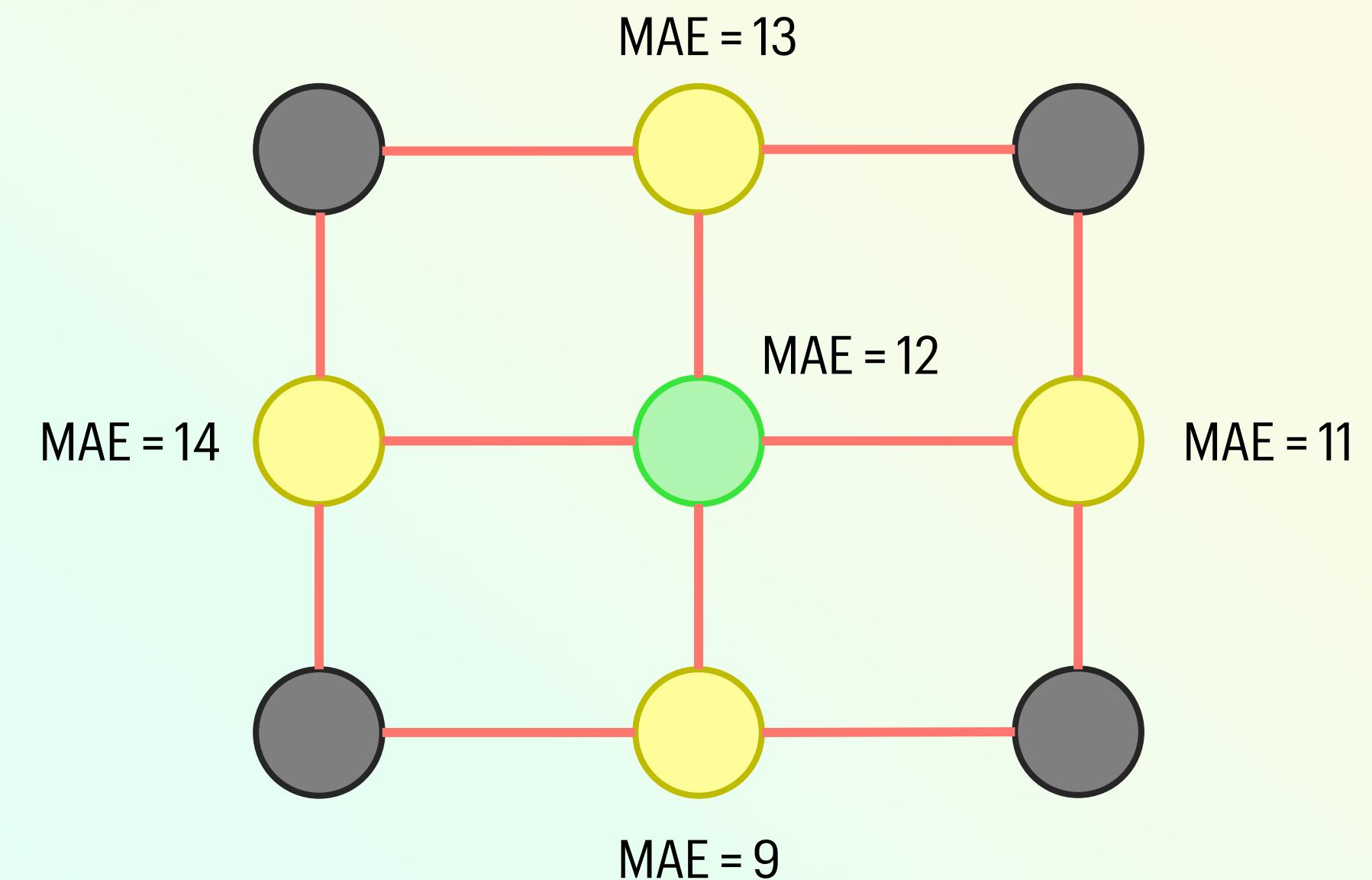
- **Gradiente Descendiente o Ascendente**
- Simulated annealing
- Búsqueda Local Beam
- Algoritmos Genéticos



# MÉTODOS AVANZADOS

Podemos crear nuestros métodos que sean más avanzados. En inteligencia artificial habían visto algoritmos de búsquedas. Podemos usar estos para buscar hiper-parámetros:

- **Gradiente Descendiente o Ascendente**
- Simulated annealing
- Búsqueda Local Beam
- Algoritmos Genéticos



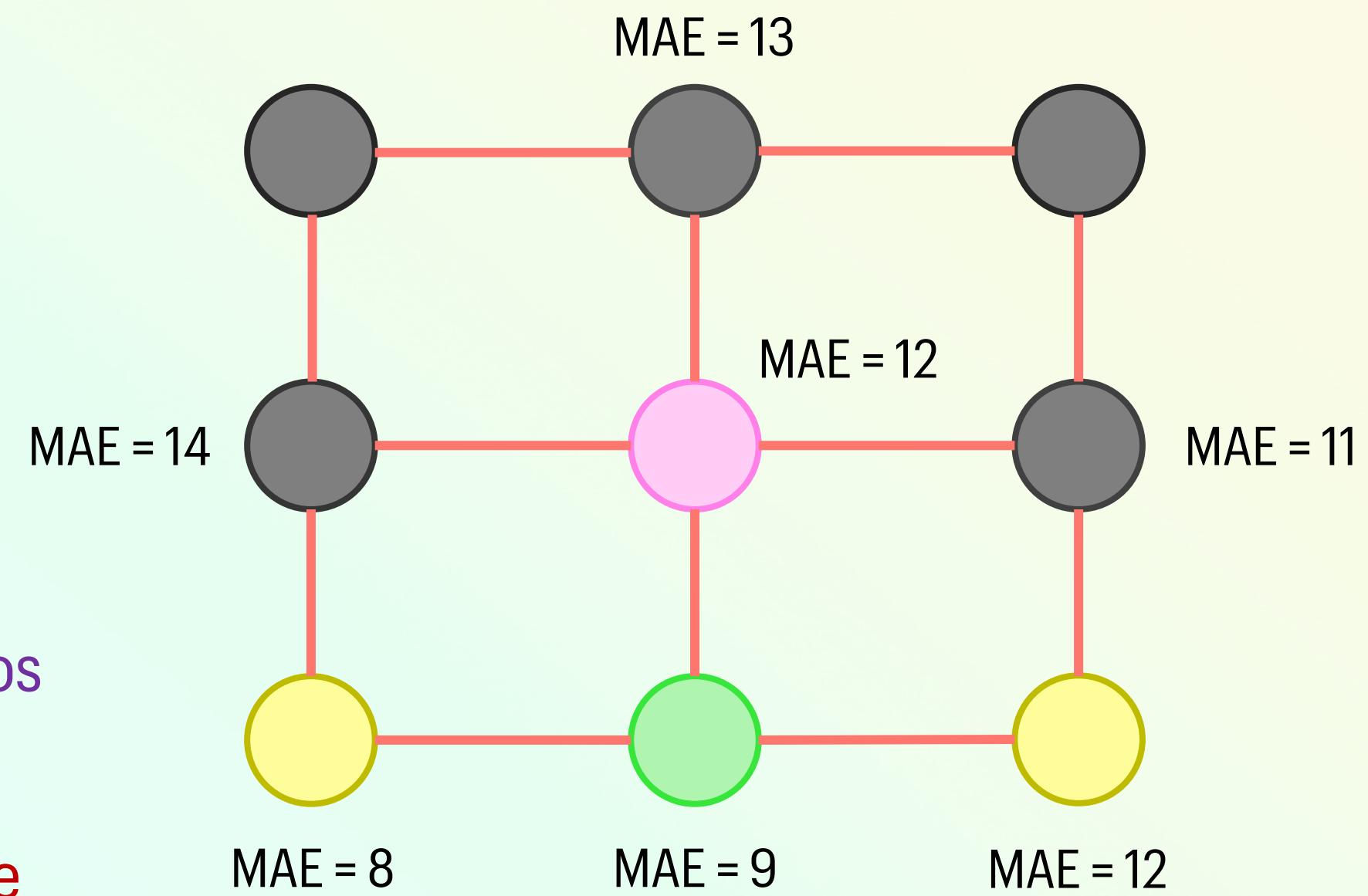
# MÉTODOS AVANZADOS

Podemos crear nuestros métodos que sean más avanzados. En inteligencia artificial habían visto algoritmos de búsquedas. Podemos usar estos para buscar hiper-parámetros:

- **Gradiente Descendiente o Ascendente**
- Simulated annealing
- Búsqueda Local Beam
- Algoritmos Genéticos

De esta forma podemos explorar le espacio sin tener que ir todos los parámetros exhaustivamente, sino con algo de inteligencia.

El problema de este método es que se depende de donde se arranca, es fácil de caer en mínimos locales dado la complejidad del espacio de hiper-parámetros (se puede usar gradiente estocástico para remediar), y que es difícil de parallelizar (aunque es posible múltiples comienzos).



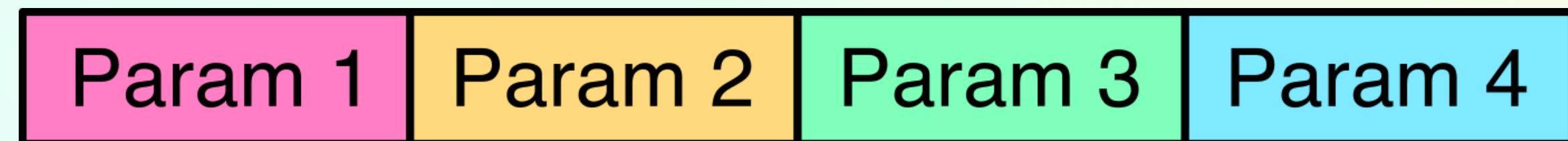
# MÉTODOS AVANZADOS

Podemos crear nuestros métodos que sean más avanzados. En inteligencia artificial habían visto algoritmos de búsquedas. Podemos usar estos para buscar hiper-parámetros:

- Gradiente Descendiente o Ascendente
- Simulated annealing
- Búsqueda Local Beam
- **Algoritmos Genéticos**

# MÉTODOS AVANZADOS

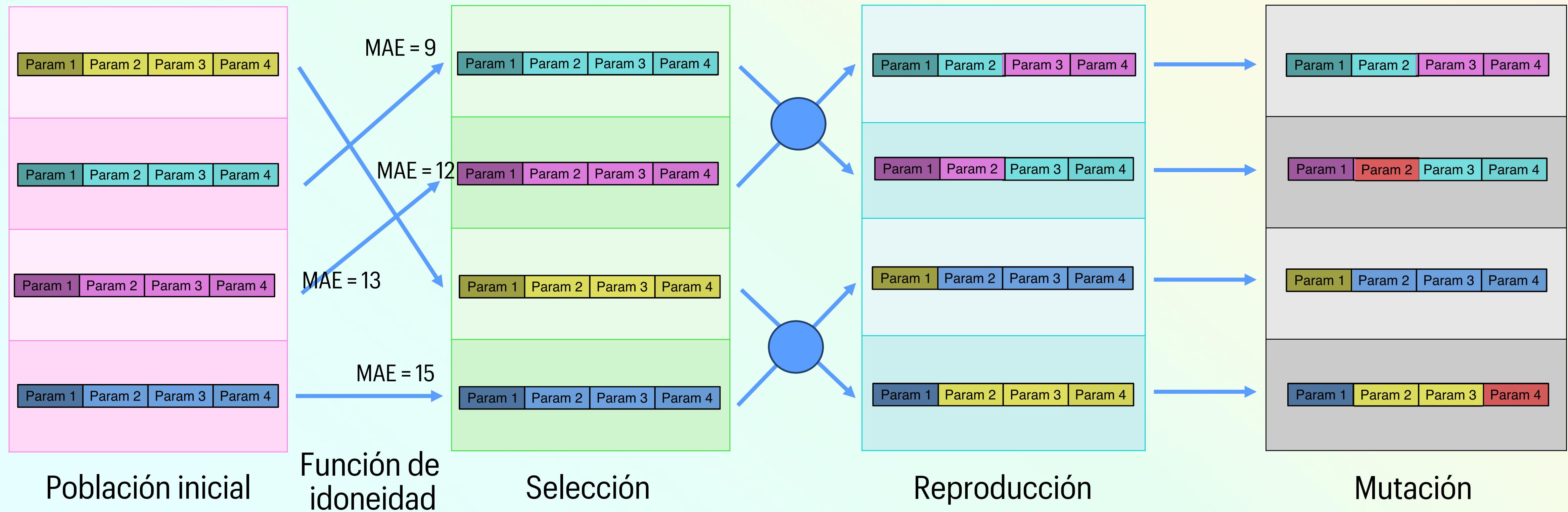
Esta problemática es apropiada para algoritmos genéticos. Podemos codificar el cromosoma con los valores de los hiper-parámetros:



Una manifestación de este cromosoma es un set de hiper-parámetros.

Entonces como función de idoneidad es el resultado de la función de puntaje del esquema de validación cruzada.

# MÉTODOS AVANZADOS



# MÉTODOS AVANZADOS

Este método es fácil de hacerlo más eficiente computacionalmente aprovechando multiproceso y una buena definición de una generación inicial apropiada y una tasa de mutación correcta, puede llevarnos a buenos resultados sin explorar tanto.

La principal desventaja es difícil de definir y configurar, hay herramientas más poderosas y más seguras.

**VAMOS A PRÁCTICAR UN POCO...**

# **FRAMEWORK DE BÚSQUEDA**

# FRAMEWORK DE BÚSQUEDA

Hay muchos estudios que aplican algoritmos que buscan encontrar selecciones de hiperparámetros eficientes, y cortar evaluaciones de combinaciones no tan atractivas.

Un framework que nos ofrece técnicas más avanzadas de búsqueda de hiperparámetros es **Optuna**. Para buscar hiper-parámetros realiza dos acciones que ayudan a ser más eficiente en su búsqueda:

- Selección de hiper-parámetros que pueden dar buenos resultados.
- Podado de hiper-parámetros que es innecesario buscar por malos resultados.

# FRAMEWORK DE BÚSQUEDA

## Selección de hiper-parámetros

Hay muchos tipos de búsqueda para este problema. **Optuna** usa una combinación de TPE+ CMA-ES:

**Tree-structured Parzen Estimator (TPE)**: Es una técnica de optimización bayesiana. Se basa en el teorema de Bayes para actualizar la probabilidad de que una configuración de hiper-parámetros sea la óptima, dadas las observaciones de su rendimiento en el proceso de optimización.

En lugar de asignar una probabilidad uniforme a todas las configuraciones posibles de hiper-parámetros, TPE utiliza dos modelos de densidad: uno para las configuraciones que han producido un mejor rendimiento y otro para las configuraciones que han producido un peor rendimiento. Se utiliza esta información para seleccionar de manera más inteligente las próximas configuraciones de hiper-parámetros a evaluar, priorizando aquellas que se espera que mejoren el rendimiento del modelo.

# FRAMEWORK DE BÚSQUEDA

## Selección de hiper-parámetros

Hay muchos tipos de búsqueda para este problema. **Optuna** usa una combinación de TPE+ CMA-ES:

**Covariance Matrix Adaptation - Evolution Strategy (CMA-ES):** Es un algoritmo de optimización basado en evolución que busca encontrar los óptimos en espacios de alta dimensionalidad.

Utiliza una estrategia de evolución para ajustar una distribución multivariada de probabilidad que representa la población de soluciones candidatas. La matriz de covarianza se adapta durante el proceso de optimización para guiar la búsqueda hacia las regiones más prometedoras del espacio de búsqueda.

# MÉTODOS DE AJUSTE DE LOS HIPER-PARÁMETROS

## Podado de hiper-parámetros

Por otro lado, el podado permite reducir el espacio de búsqueda, determinando anticipadamente que combinación de hiper-parámetros no va a dar buenos resultados. Optuna implementa esto usando una versión de reducción a la mitad sucesiva (halving).

**VAMOS A PRÁCTICAR UN POCO...**