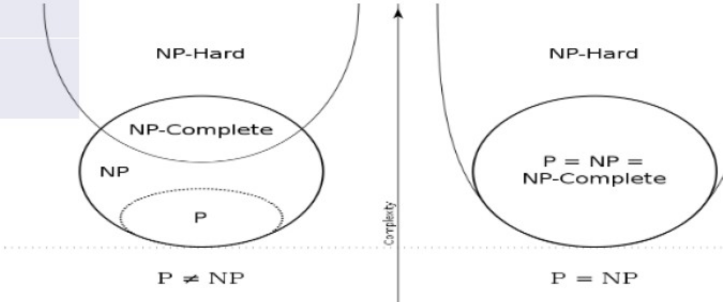
Introducción(Simplificada) a la teoría de la complejidad

- ¿P=NP? O, ¿Qué problemas se pueden resolver fácilmente con computación?

Tipo de problema	Verificable en tiempo P	Solución en tiempo P	Dificultad
P	SI	SI	-
NP	SI	Algunos	
NP-Completos	SI	Desconocido	
NP-Duros	Algunos	Desconocido	+

P= tiempo polinomial

- **Problema del millón de dólares**



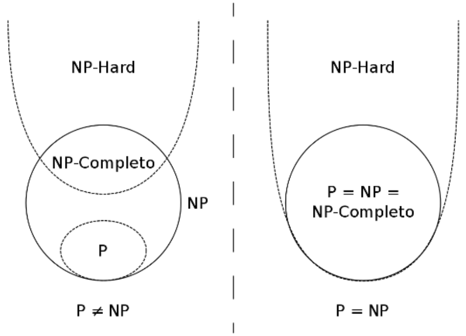
NP : no se conoce solución en tiempo polinomial
NP-Completo: si un problema NP-Completo se demuestra que es P entonces P=NP ! (El resto de NP se pueden transformar con complejidad Polinomial)
NP-Duros: Al menos tan complejo como cualquier NP pero puede que no sea NP

Introducción(Simplificada) a la teoría de la complejidad

- **Ejemplos:**

- NP-Completo : Satisfacibilidad booleana(SAT)(1971-Stephen Cook).
  * Todos los NP-Completos se “*reducen*” a SAT
  https://es.wikipedia.org/wiki/Anexo:Problemas_NP-completos

- NP-Duro
  * Problema del Agente Viajero(TSP)

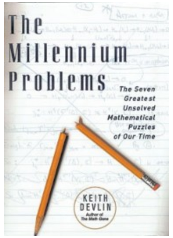


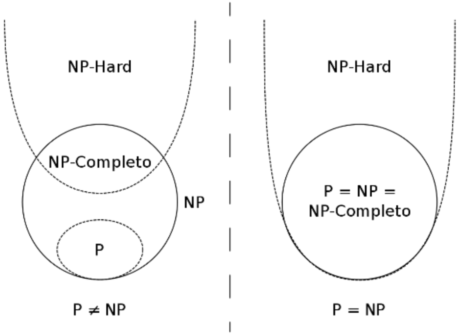
Introducción(Simplificada) a la teoría de la complejidad

- ¿P=NP?
  Resolverlo supondría encontrar “atajos” para resolver problemas NP a partir de problemas P (reducción de un problema a otro)

  Todo indica que P no es igual a NP por lo que “casi” asumimos que **no podemos** abordar estos problemas computacionalmente(*) con algoritmos exactos!

  Es muy probable que solo podamos resolver estos problemas por métodos aproximados o usando heurísticas.



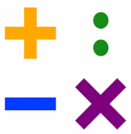


http://www.claymath.org/millennium-problems/p-vs-np-problem

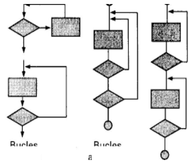
(*)Con computación clásica, pero, ¿que ocurrirá con la computación cuántica?

Análisis de Algoritmos

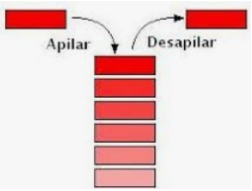
- **Contar operaciones elementales**
  - Suma, resta, multiplicación y división
  - Asignaciones
  - Condiciones , llamadas a funciones externas y retornos
  - Accesos a estructuras de datos
  - Ojo con los bucles anidados! 2 bucles => n^2



a+=b	a = a + b
a-=b	a = a - b
a*=b	a = a * b
a/=b	a = a / b
a%=b	a = a % b



- Orden de complejidad
  - ✓ Decimos que un algoritmo tiene orden de complejidad $O(f)$ si su tiempo de ejecución en operaciones elementales está acotado, salvo por una constante, por f para todos los tamaños de entrada a partir de un cierto n_0



Repaso

Análisis de Algoritmos. Sin recursividad

Contar operaciones elementales.

```
#Fuerza Bruta

def distancia_fuerza_bruta(L):                1
  mejor_distancia = 100000e10

  A,B = (),()                                 2

  for i in range(len(L)):
    for j in range(i+1, len(L)):
      D = distancia(L[i],L[j])                3
   1  if D < mejor_distancia:
            A,B=L[i],L[j]                     2
            mejor_distancia = D                         1
  return [A,B]                                1

distancia_fuerza_bruta(LISTA_2D)
```

Contar cuantas realiza la función distancia: 2 (en 1-dimensión)

Total Operaciones | **Complejidad**

$7 \cdot n \cdot (n-1) \sim n^2$

Total: $7 \cdot n^2 + 4 \Rightarrow O(n^2)$

Análisis de Algoritmos. Ecuaciones en recurrencia

El termino n-simo depende de los términos anteriores

```python
#Sucesión_de_Fibonacci
#https://es.wikipedia.org/wiki/Sucesi%C3%B3n_de_Fibonacci
#Calculo del termino n-simo de la suscesión de Fibonacci
def Fibonacci(N:int):
  if N < 2:
    return 1
  else:
    return Fibonacci(N-1)+Fibonacci(N-2)

Fibonacci(5)
```

```
8
```

T(n) = T(n-1) + T(n-2)

T(n) - T(n-1) – T(n-2) = 0

Ecuación característica:

$$x^2 - x - 1 = 0$$

$$x_1 = \frac{1+\sqrt{5}}{2} \qquad x_2 = \frac{1-\sqrt{5}}{2}$$

La solución es

$$T(n) = A \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + B \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$$

A y B dependen de los términos iniciales

Análisis de Algoritmos. Ecuaciones en recurrencia

El termino n-simo depende de una división de los términos anteriores

```
#quick_sort

A = [9187, 244, 4054, 9222, 8373, 4993, 5265, 5470, 4519, 7182, 2035, 3506, 4337, 7580, 2554, 2824, 8357, 4447, 7379]


def quick_sort(A):
  if len(A) == 1:
    return A
  if len(A) == 2:
    return [min(A), max(A)]

  IZQ=[]
  DER=[]

  #pivote = (A[0]+ A[1] + A[2])/3     #Falla en algunos casos, cuando hay valores repetidos y coinciden al principio de alguna lista
  pivote = sum(A)/len(A)              #Buena opción. La mejor opción seria la mediana pero esto pasa por ordenar la lista
                                      # que es precisamente lo que queremos hacer.


  for i in A:
    if i <= pivote :
      IZQ.append(i)
    else:
      DER.append(i)

  return quick_sort(IZQ) + quick_sort(DER)



@calcular_tiempo
def QS(A):
  return quick_sort(A)

print(QS(A))
```

Ecuaciones

$$T_n = \begin{cases} c \cdot n^k, 1 \leq n < b \\ a \cdot T\left(\frac{n}{b}\right) + c \cdot n^k, \wedge n \geq b \end{cases}$$

$$T(n) = 2 \cdot T(n/2) + n$$

a=2
b=2
c=1
k=1

Soluciones

$T(n) = O(n^k)$ si $a < b^k$

$\boxed{T(n) = O(n^k \cdot \log(n)) \text{ si } a = b^k}$

$T(n) = O(n^{\log_b(a)})$ si $a > b^k$

En el mejor caso!

(*) El termino n-simo depende de una división de los términos anteriores

Análisis de la complejidad temporal

importante

- Ordenes de complejidad (de menor a mayor)

$O(1)$	Orden constante
$O(\log n)$	Orden logarítmico
$O(n)$	Orden lineal
$O(n \log n)$	Orden cuasi-lineal
$O(n^2)$	Orden cuadrático
$O(n^3)$	Orden cúbico
$O(n^a)$	Orden polinómico
$O(2^n)$	Orden exponencial
$O(n!)$	Orden factorial

Tabla 1

Comparación de los órdenes de complejidad según el tamaño del problema

n	$\log n$	n^2	2^n	$n!$
2	1	4	4	2
8	3	64	256	40.320
32	5	1.024	$4{,}3 \times 10^9$	$2{,}6 \times 10^{35}$
100	6	10^4	$1{,}2 \times 10^{27}$	$9{,}3 \times 10^{177}$

Repaso

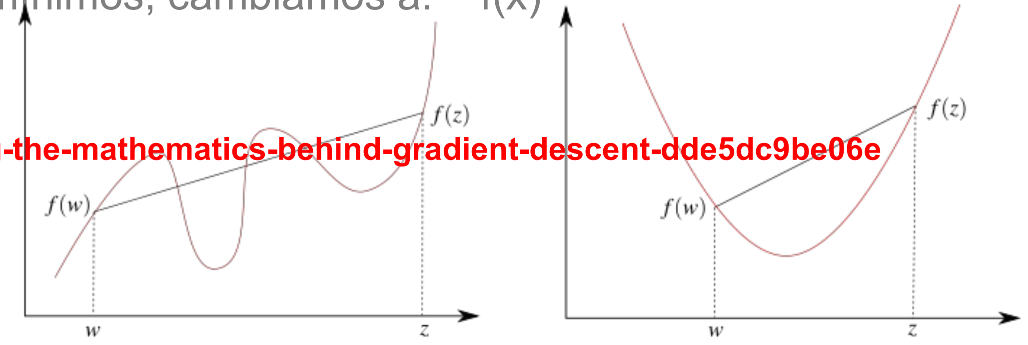
Descenso del Gradiente – AGD

importante

- Es un algoritmo **iterativo** de optimización para encontrar valores mínimos para funciones multi-varibles **convexas** y **diferenciables**(y por tanto continuas).

- Convexidad
  - Permitir asegurar que la óptimo que encontremos es global frente a que solo sea local
  - ¿El segmento que une dos puntos está siempre por encima de la función?
  - Si es cóncava y buscamos mínimos, cambiamos a: –f(x)

https://towardsdatascience.com/understanding-the-mathematics-behind-gradient-descent-dde5dc9be06e



a) Función no convexa

b) Función convexa

Descenso del Gradiente – AGD

- **Diferenciable**: derivable en n variables(dimensiones)

- **Gradiente**: Es un vector(diferente para cada punto = campo vectorial) cuyas coordenadas son las derivadas parciales:

$$\nabla f(\mathbf{r}) = \left(\frac{\partial f(\mathbf{r})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{r})}{\partial x_n} \right)$$

Ejemplo: Dada la función **f(x,y) =** $x^2 + 2x + y^2 + y^3 + xy$

el gradiente será:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \qquad (2x + y + 2, 2y + 3y^2 + x)$$

En el punto (x,y)

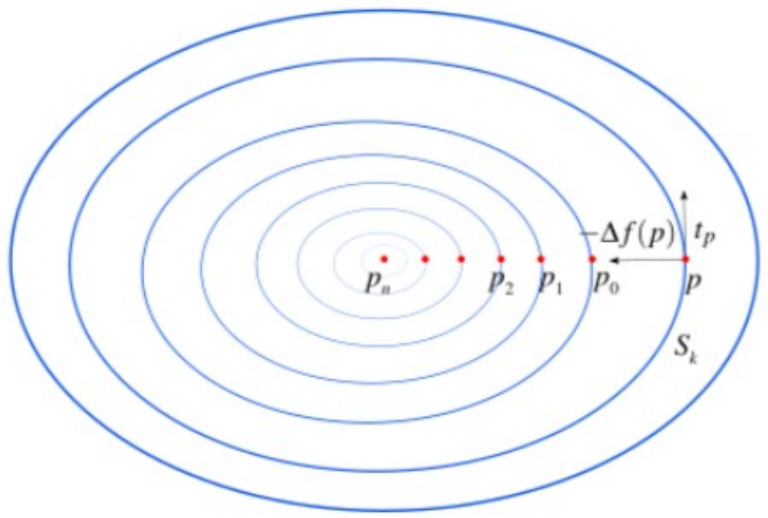
Descenso del Gradiente – AGD

El procedimiento parte de un punto **p como solución aproximada(inicial)** y da pasos en el sentido opuesto(si minimizamos) al gradiente de la función en dicho punto.

$$p_{t+1} = p_t - \alpha_t \cdot \Delta f(p_t)$$

La elección de α_t estará condicionada para que :

- p_{t+1} sea solución factible
- mejore el valor de f respecto a p_t : $f(p_{t+1}) \geq f(p_t)$



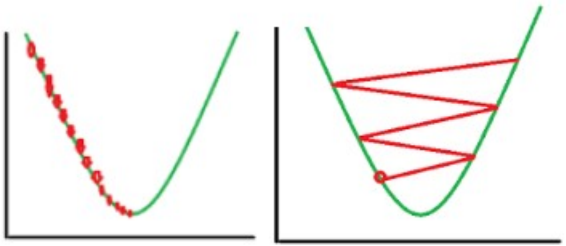
Descenso del Gradiente – AGD

La elección del parámetro α_t , llamado **tasa de aprendizaje (learning rate),** es importante para hacer efectivo el proceso de acercamiento a la solución óptima.

- Un valor demasiado pequeño puede provocar exceso de iteraciones(Figura 1)

- Un valor demasiado alto puede provocar que el proceso no se acerque los suficiente(Figura 2)

$$p_{t+1} = p_t - \alpha_t \cdot \Delta f(p_t)$$



learning rate demasiado pequeño
(Figura 1)

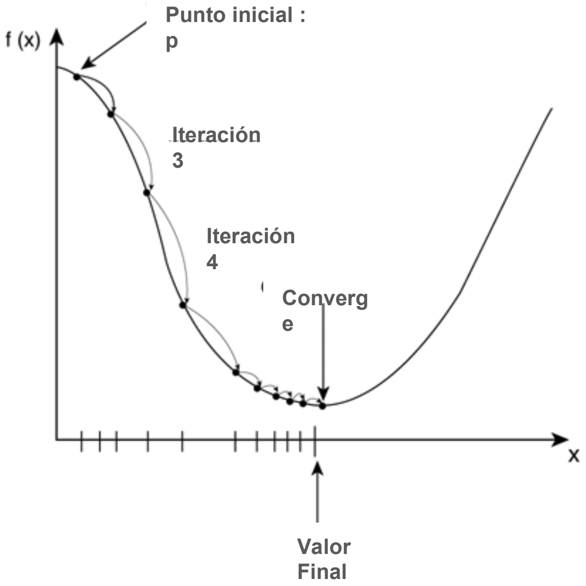
learning rate demasido grande
(Figura 2)

Descenso del Gradiente – AGD

En general es buena estrategia ir reduciendo el valor de α_t dinámicamente a medida que nos aproximamos a la solución

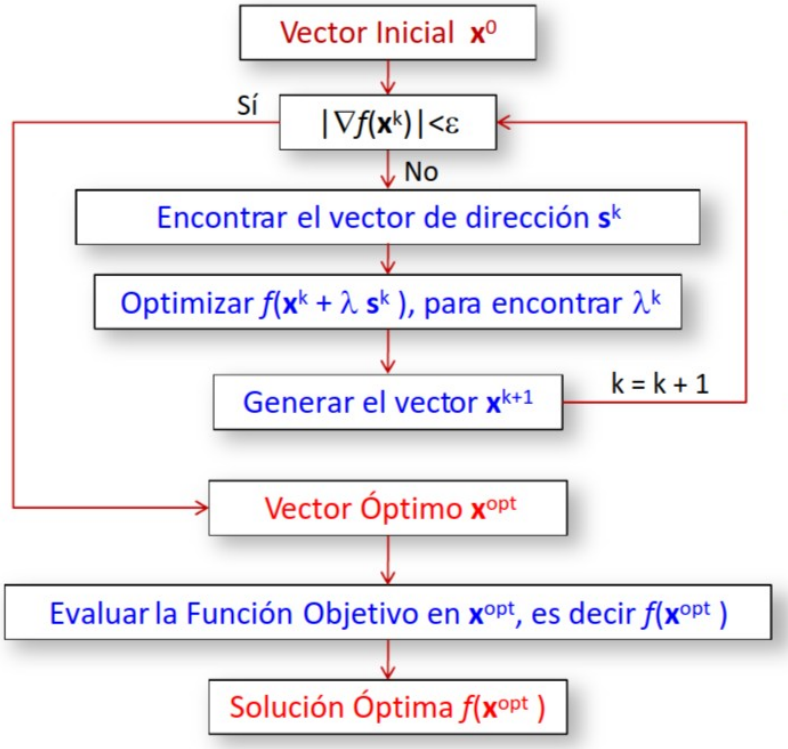
- ¿Como sabemos que nos acercamos?:
  - la magnitud del gradiente
  - cantidad de iteraciones que hemos realizado

$$p_{t+1} = p_t - \alpha_t \cdot \Delta f(p_t)$$



Descenso del Gradiente – AGD

- **Diagrama de resolución**

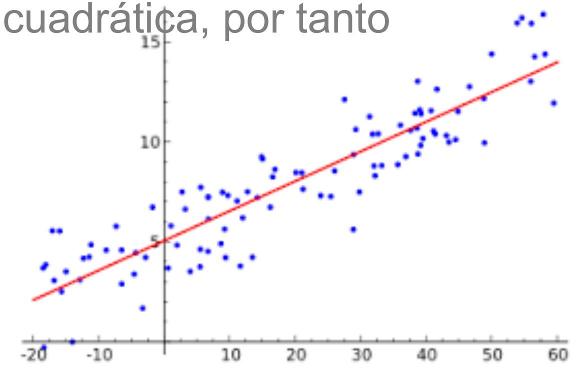


$$\mathbf{s}^k = \mp \frac{\nabla f(\mathbf{x}^k)}{|\nabla f(\mathbf{x}^k)|}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \lambda^k \mathbf{s}^k$$

Descenso del Gradiente – AGD

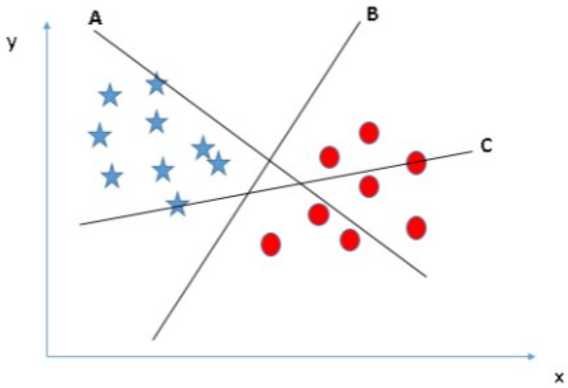
- **Aprendizaje supervisado y la regresión lineal**
  - ✓ En problemas de clasificación tratamos de clasificar los puntos en sus categorías correspondientes de tal manera que se minimice el error.
  - ✓ Para la medida de este error solemos utilizar el **error cuadrático medio**(regresión lineal) pero es posible usar otras medidas(por ejemplo: distancia Euclidea, Manhattan, ...)
  - ✓ La función que acumula los errores para los valores conocidos la llamamos función de coste. En el caso de la regresión línea es cuadrática, por tanto diferenciable y podemos obtener el gradiente.

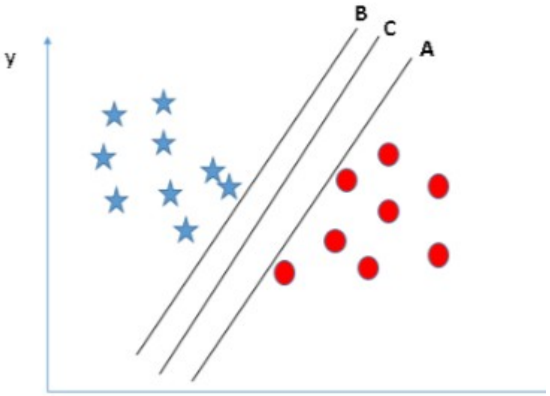


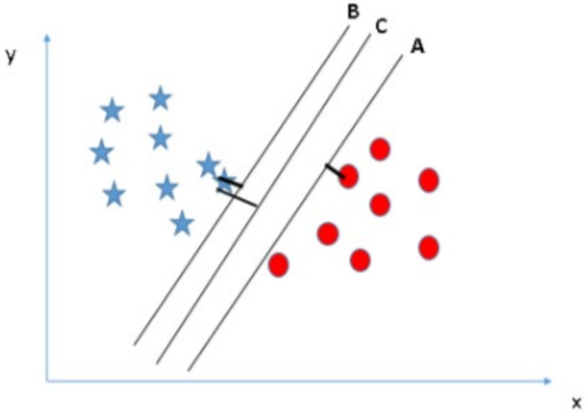
Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?





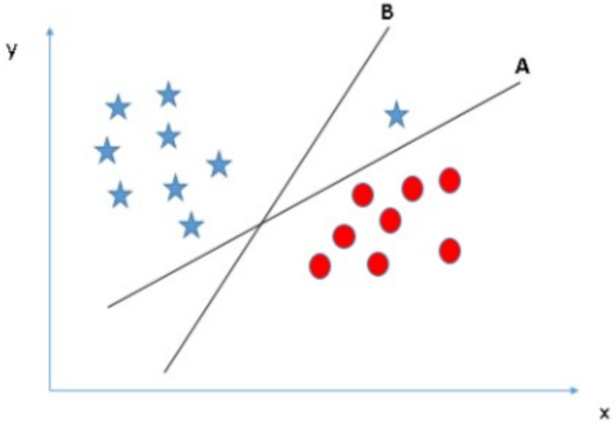


Seleccionar el que esté a mayor distancia de ambos conjuntos

Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?



1º . Seleccionar el que mejor clasifique

2º . Seleccionar el que está a mas distancia

Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

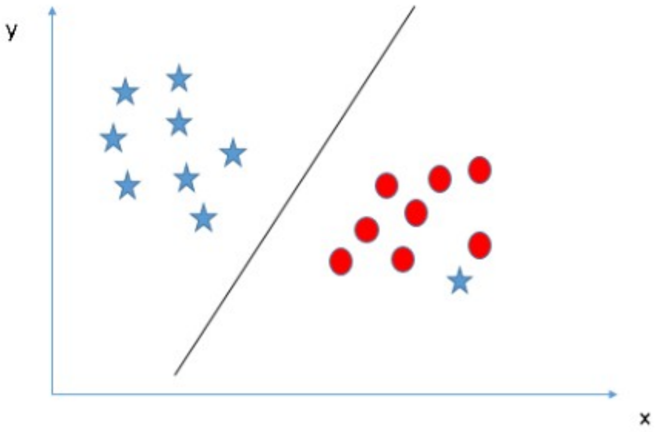
¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

- *No siempre es posible clasificar 100%*
- *¿valores atípicos?*

Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

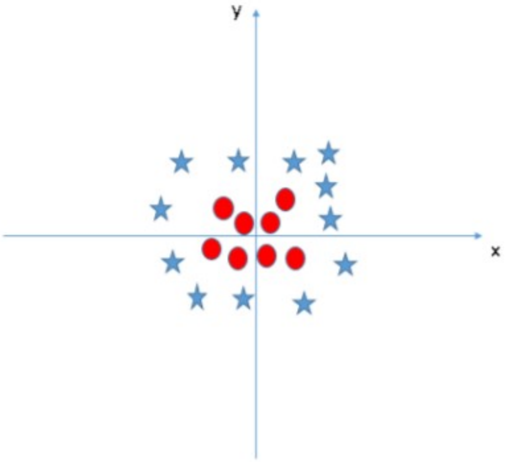


- ***Ignorar valores atípicos***

Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

¿Cómo identificar la mejor recta (hiperplano en N-dimensión) que separa los conjuntos de datos?

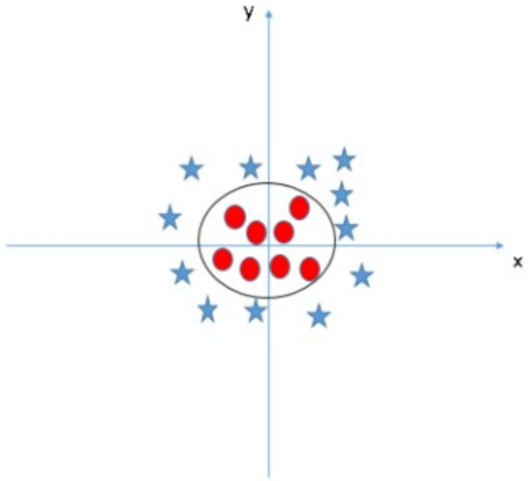


- *No parecen valores atípicos*
- *Parece que si hay una clasificación*
- *Es imposible con hiperplanos lineales*

Descenso del Gradiente – AGD

- **Aprendizaje supervisado y la regresión lineal**

Podemos extender el procedimiento a funciones no lineales!



Una función como $f(x, y) = x^2 + y^2$ *lo resuelve*

Descenso del Gradiente – AGD

- **Otra opción para función de coste(no cuadrática)**

- **Entropía cruzada(cross-entropy)**
  - Para problemas con soluciones binarias
  - **Definición**: Número de bits diferentes. Mide como de diferentes son dos elementos.
  - Usada en algoritmos de clasificación de imágenes

Descenso del Gradiente – AGD

- **Dependiendo del Volumen de Datos**

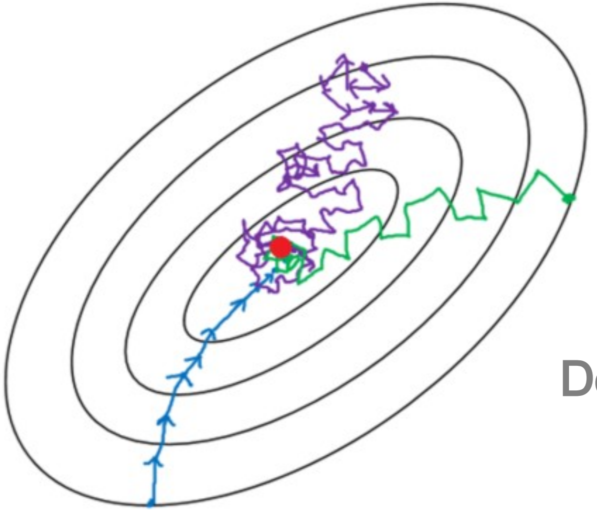
  ✓ Descenso del gradiente por lotes (**batch gradient descent**)
  Calcula la desviación para todos los puntos en cada iteración!!!

  ✓ Descenso del gradiente estocástico(**stochastic gradient descent**)
Calcula la desviación para un punto en cada iteración!!!

  ✓ Descenso del gradiente por lotes reducido(**mini-batch gradient descent**)
  Mezcla de ambos conceptos

Descenso del Gradiente – AGD

- Dependiendo del Volumen de Datos



Batch gradient descent (directo pero muy lento)

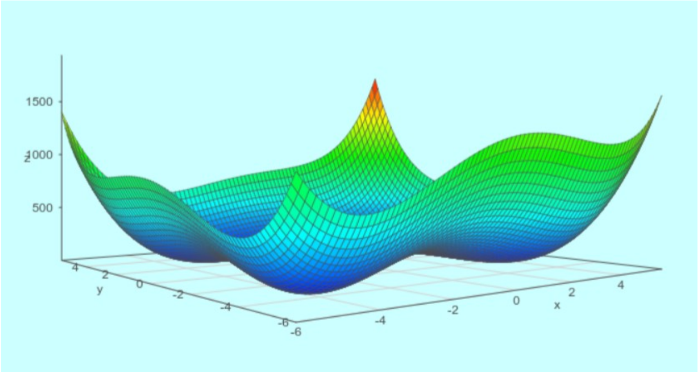
Stochastic gradient descent(rápido pero muy disperso)

Mini-batch gradient descent(equilibrio)

Decisión: Elección de mini-batch size

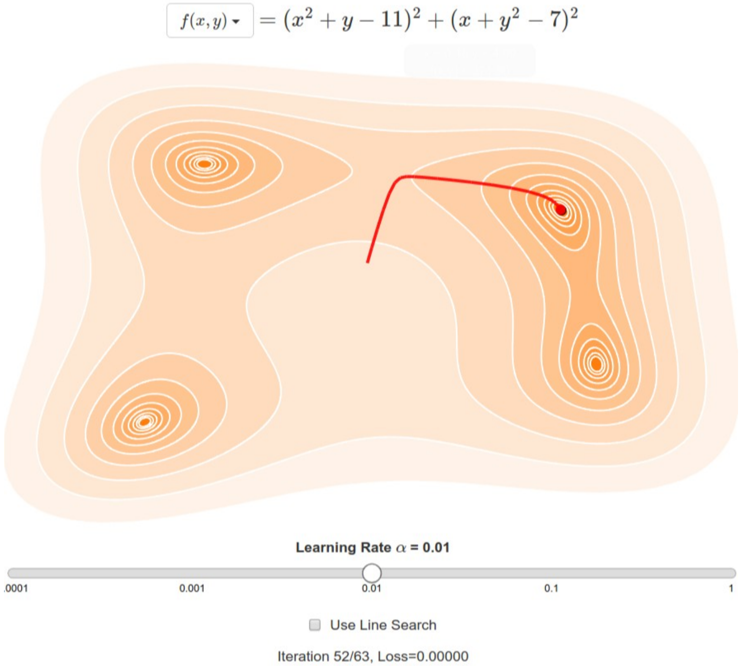
Descenso del Gradiente – AGD

Visualización



(x^2+y-11)^2+(x+y^2-7)^2

http://al-roomi.org/3DPlot/index.html



https://www.benfrederickson.com/numerical-optimization/