

## Binary tree

Árboles numéricas > árboles cualitativas  
└ Los transformamos

Entropía recursiva → formar el árbol.

## Árboles cualitativos.

\* Revisar árboles\_cualitativos.py en github.

## Threshold splits

Dividir los datos para ver en cual partición tengo mayor G1

- Bagging (mayoría)
  - Boosting
- } Combinar modelos sencillos para obtener uno más complejo.

## BOSQUES ALEATORIOS.

Muchos árboles pequeños

## SESGOS COGNITIVOS

- Necesita patrones

## REGRESIÓN LINEAL

Métodos más antiguos de aprendizaje

- Tengo ejps. y asumo que son numéricos  $H = \text{conjunto de funciones}$

$$x \in \mathbb{R}^n \quad x(x_1, \dots, x_m)$$

$$\begin{matrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(m)} \end{matrix} \left[ \begin{matrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{matrix} \right] \left[ \begin{matrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{matrix} \right]$$

matriz

Primero construyo un extractor de características

$$x = (x_1, x_2, \dots, x_m) \in \mathbb{R}^n$$

$$\phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$\phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_m(x))$$

$$\phi(x) = (1, x_1, x_2, \dots, x_m)$$

$$h_w(x) = \omega^T \phi(x)$$

Reducir las características  
a unas nuevas

CASO 1.  $x = X$ , Lineal en datos y lineal en las características  
 $\phi(x) = (1, X_1)$   
 $\omega = (\omega_0, \omega_1)$   
 $h_w(x) = (\omega_0, \omega_1) \begin{pmatrix} 1 \\ x_1 \end{pmatrix} = \omega_0 + \omega_1 x_1$

CASO 3.  $x = (x_1, x_2, x_3)$   
 $\phi(x) = (1, x_1, x_2, x_3)$  función afín: función lineal + constante  
 $\omega = (\omega_0, \omega_1, \omega_2, \omega_3)$   
 $h_w(x) = (\omega_0, \omega_1, \omega_2, \omega_3)$   
 $h_w(x) = (\omega_0, \omega_1, \omega_2, \omega_3) \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \omega_3 x_3$

### FUNCIÓN DE PÉRDIDA

Menor error posible!

No buscamos de donde viene el error

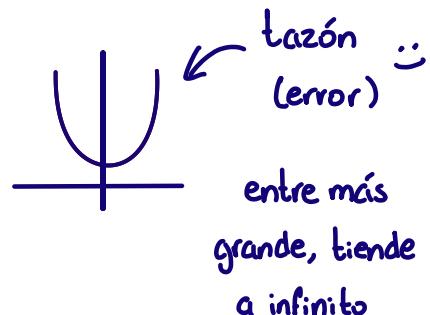
$$\text{Loss}(y, \hat{y}) = \frac{1}{2} (y - \hat{y})^2 \quad \leftarrow \text{error cuadrático}$$

$$E_{\text{in}}(h_\theta) = \frac{1}{M} \sum_{i=1}^M \text{Loss}(y^{(i)}, h_\theta(x^{(i)}))$$

$$\hookrightarrow E_{\text{in}}(h_\theta) = \frac{1}{M} \sum_{i=1}^M \frac{1}{2} (y^{(i)} - \underbrace{\omega^T \phi(x^{(i)})}_h)^2$$

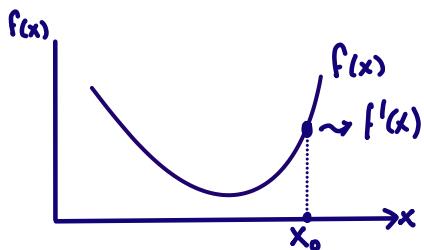
NSE (Minimum Square Error)

$$\omega^* = \arg \min_{\omega \in \mathbb{R}^n} \frac{1}{2M} \sum_{i=1}^M (y^{(i)} - \underbrace{\omega^T \phi(x^{(i)})}_h)^2$$



## DESCENSO A GRADIENTE

$\alpha$ : paso/taza de aprendizaje



Busco el punto más bajo

$$\begin{aligned} X_0 &= f(x_0) & f'(x_0) \\ X_1 &\leftarrow X_0 - \alpha f'(x_0) \\ X_2 &\leftarrow X_1 - \alpha f'(x_1) \\ &\vdots \\ X_{n-1} &\leftarrow X_n - \alpha f'(x_n) \end{aligned}$$

¿Cuándo paro?

Antes de volver a subir o cuando tengo el valor más bajo

Si el punto es muy grande: me paso y no aprendo

Si el punto es muy chico: aprende MUY Lento

Buscar el paso justo a prueba y error

$$h_w(x) = w^T \phi(x)$$

$$X = (x_1, \dots, x_n) \in \mathbb{R}^n$$

$$\phi(x) = (1, x_1, \dots, x_n) \in \mathbb{R}^{n+1}$$

$$w = (w_0, w_1, \dots, w_n) \in \mathbb{R}^{n+1}$$

$$\text{Loss}(h_w) = \frac{1}{M} \sum_{i=1}^M 2(y^{(i)} - h_w(x^{(i)}))^2$$

$$-\frac{1}{M} \begin{bmatrix} \sum_{i=1}^M e^{(i)} \\ \sum_{i=1}^M e^{(i)} x_1^{(i)} \\ \vdots \\ \sum_{i=1}^M e^{(i)} x_m^{(i)} \end{bmatrix} = \frac{1}{M} \phi(x)^T (y - \hat{y})$$

$\text{lr}$ : learning rate,  $\text{tol}$  := tolerancia

desc\_grado( $w_{\text{ini}}$ ,  $X$ ,  $y$ ,  $\text{lr}$ , max\_epochs, tol)

$$w \leftarrow w_{\text{ini}}$$

para epochs de 1 = max\_epochs:

$$\hat{y}^{(i)} \leftarrow w^T \phi(x^{(i)}) \quad \forall i=1,2,\dots,m$$

$$e^{(i)} \leftarrow y^{(i)} - \hat{y}^{(i)} \quad \forall i=1,\dots,m$$

$$E_{\text{in}} \leftarrow \frac{1}{M} \sum_{i=1}^M e^{(i)}^2$$

$$\nabla_w E_{\text{in}} \leftarrow$$

$$\text{si } \|\nabla_w E_{\text{in}}\| < \text{tol}$$

return  $w$

$$w \leftarrow w - n \cdot \nabla_w E_{\text{in}}$$

Nos damos cuenta

$$\phi(X) = \begin{bmatrix} X_1^{(1)} & X_2^{(1)} & \dots & X_n^{(1)} \\ X_1^{(2)} & X_2^{(2)} & \dots & X_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{(m)} & X_2^{(m)} & \dots & X_n^{(m)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\hat{y}^{(i)} = w^T \phi(x^{(i)}) = \phi(x^{(i)})^T w$$

$$\hat{y} = \phi(x) w$$

$$E = y - \hat{y}$$

$$E_{\text{in}} = \frac{1}{m} E^T E$$

$$\nabla_w E_{\text{in}} \leftarrow \frac{1}{M} \phi(x)^T E$$

$$\rightarrow \hat{y} = \phi(x) w \quad \begin{matrix} (m, n+1)(m, 1) \\ \text{Producto punto !} \\ \| \cdot \|_0 \end{matrix}$$

$$\text{si } \|\nabla_w E_{\text{in}}\| < \text{tol} \quad \text{return } w$$

$$w = w - n \cdot \nabla_w E_{\text{in}}$$

## LINEAR CLASSIFICATION FRAMEWORK

Ej.

$$y^{(i)} \in \{-1, 1\} \quad \text{solo tiene 1 y -1 mijo}$$

1 := E clase

-1 := E otra clase

$$x^{(i)} = (x_1^{(i)}, \dots, x_n^{(i)}) \in \mathbb{R}^n$$

$$\phi(x^{(i)}) = (1, x_1^{(i)}, \dots, x_n^{(i)}) \in \mathbb{R}^{n+1}$$

$$\omega = (\omega_0, \omega_1, \dots, \omega_n) \in \mathbb{R}^{n+1}$$

$$\hat{y}^{(i)} = h(x^{(i)}) = \text{sign}(\underbrace{\omega^\top \phi(x^{(i)})}_{\text{calcula el signo}}) = \text{sign}(\phi(x^{(i)})^\top \omega)$$

$$\omega_1 x + \omega_0 = 0$$

Relación entre  
entrada y  
parámetros  
= Lineal

$1[\dots] \leftarrow$  función indicadora

$1[\omega^\top \phi(x^{(i)}) y^{(i)} \leq 0] \leftarrow$  son distintos porque son negativos

$$E_{out}(\omega) = \frac{1}{M} \sum_{i=1}^M 1[\omega^\top \phi(x^{(i)}) y^{(i)} \leq 0]$$

$$Pr(y=1 | x; \omega, b) = \sigma(z) = \hat{y}$$

$$z = \omega_0 x_0 + \omega_1 x_1 + \omega_2 x_2 + \dots + \omega_n x_n + b$$

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial}{\partial \omega_i} \frac{1}{M} \sum_{i=1}^M [y^{(i)} - \hat{y}^{(i)}] \log(\hat{y}^{(i)}) + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

$$\frac{\partial E_{in}}{\partial \omega_i} = \frac{2}{M} \sum_{i=1}^M [y^{(i)} - \hat{y}^{(i)}] \frac{\log(\hat{y}^{(i)}) - (1-\hat{y}^{(i)}) \log(1-\hat{y}^{(i)})}{\sigma(z^{(i)})}$$

$$\nabla_\omega E_{in} = \begin{bmatrix} \frac{\partial E_{in}}{\partial \omega_0} \\ \vdots \\ \frac{\partial E_{in}}{\partial \omega_n} \end{bmatrix} = -\frac{1}{M} \begin{bmatrix} \sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)}) x_0^{(i)} \\ \vdots \\ \sum_{i=1}^M (y^{(i)} - \hat{y}^{(i)}) x_n^{(i)} \end{bmatrix}$$

$$\omega \leftarrow \omega_{-init}$$

$$b \leftarrow b_{-init}$$

para epochs de 1 a max\_epoch

$$\nabla_\omega E_{in} \leftarrow \text{Calcula_gradiente}(x, y, \omega, b)$$

$$\frac{d}{db} E_{in} \leftarrow \text{Calcula_derivada}(x, y, \omega, b)$$

hist, append(calcula.Ein(x, y, \omega, b))

$$\omega \leftarrow \omega - \eta \nabla_\omega E_{in}$$

$$b \leftarrow b - \eta \frac{d}{db} E_{in}$$

if  $\|\nabla_\omega E_{in}\|_\infty < E_{-total}$ :

break

fin para

regresa \omega, b

$$\frac{1}{M} \sum_{i=1}^M -\frac{y^{(i)}}{\hat{y}^{(i)}} \frac{\partial \sigma(z^{(i)})}{\partial \omega_j} + \frac{1-y^{(i)}}{1-\hat{y}^{(i)}} \frac{\partial \sigma(z^{(i)})}{\partial \omega_j}$$

$$\frac{\partial \sigma(z^{(i)})}{\partial \omega_j} = \frac{\partial \sigma(z^{(i)})}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial \omega_j} = \underbrace{\sigma(z^{(i)})}_{g^{(i)}} \underbrace{(1-\sigma(z^{(i)}))}_{1-g^{(i)}} x_j^{(i)}$$

$$\frac{\partial}{\partial \omega_j} (\omega_0 x_0^{(i)} + \omega_1 x_1^{(i)} + \dots + \omega_n x_n^{(i)})$$

$$\frac{\partial E_{in}}{\partial \omega_j} = \frac{1}{M} \sum_{i=1}^M -\frac{y^{(i)}}{\hat{y}^{(i)}} \hat{y}^{(i)} (1-\hat{y}^{(i)}) x_j^{(i)} + \frac{1-y^{(i)}}{1-\hat{y}^{(i)}} \hat{y}^{(i)} (1-\hat{y}^{(i)}) x_j^{(i)}$$

$$= \frac{1}{M} \sum_{i=1}^M [-y^{(i)} + y^{(i)} \hat{y}^{(i)} + \hat{y}^{(i)} - y^{(i)} \hat{y}^{(i)}] x_j^{(i)}$$

$$= -\frac{1}{M} \sum_{i=1}^M [y^{(i)} - \hat{y}^{(i)}] x_j^{(i)}$$

$$\frac{\partial E_{in}}{\partial b} = -\frac{1}{M} \sum_{i=1}^M [y^{(i)} - \hat{y}^{(i)}]$$

Optimiza  $\rightarrow$  Álgebra lineal

Programando:

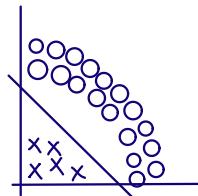
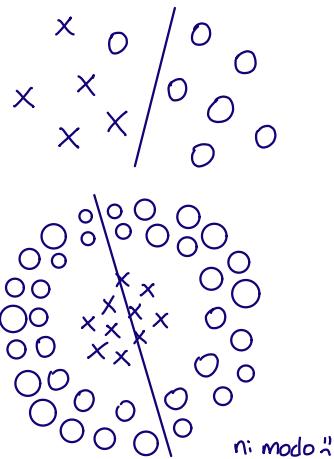
$$X = \begin{bmatrix} X_1^{(1)} & \cdots & X_n^{(1)} \\ X_1^{(2)} & \cdots & X_n^{(2)} \\ \vdots & & \vdots \\ X_1^{(m)} & \cdots & X_n^{(m)} \end{bmatrix}_{(m,n)} \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}_{(m,1)}$$

$$\hat{y} = \text{Sigmoid}(X_w + \vec{b})$$

$$\frac{dE_{in}}{db} = -\text{promedio}(y - \hat{y})$$

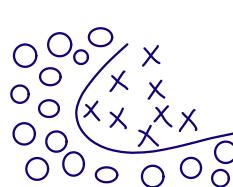
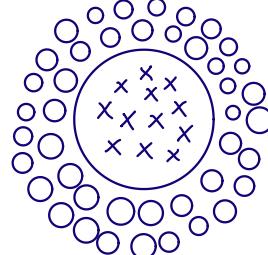
$$\nabla_w E_{in} = -\frac{1}{M} X^T (y - \hat{y})$$

REGRESIÓN LINEAL



$$y^{(i)} = (x^{(i)}, y^{(i)}) \\ \Phi = (x^{(i)}) = (x_1^{(i)}, x_2^{(i)})^T \quad \left. \right\} \text{Elevando al cuadrado}$$

\* Transformación  
Podrá ser trampa pero funciona



## MODELO BUSQUEDA

class ModeloBusqueda :

def \_\_init\_\_(self, S, A):

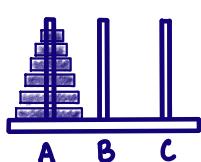
    self.S = S[:] ← desde el primero hasta el último

    self.A = A[:]

def acciones\_legales(self, s):

    return self.A[:]

## TORRE DE HANOI



S = (list\_A, list\_B, list\_C)

List\_A = {[1,2,3,4,5,6], [1,2,3,4,5], [1,2,3,4], ...}

List\_B = {\_\_\_\_\_}

List\_C = {\_\_\_\_\_}

Conj. Potencia

$$2^6 \quad 2^6 \cdot 2^6 \cdot 2^6 =$$

$$2^6 \quad 2^6$$

Cada uno puede estar o no estar

maneras de plantearlo

d → disco

lista\_A = (d<sub>1</sub>, d<sub>2</sub>, ..., d<sub>6</sub>)

List\_B = (d<sub>1</sub>, d<sub>2</sub>, ..., d<sub>6</sub>)

List\_C = (d<sub>1</sub>, d<sub>2</sub>, ..., d<sub>6</sub>)

S = (d<sub>1</sub>, d<sub>3</sub>, d<sub>4</sub>, d<sub>5</sub>, d<sub>6</sub>)

d<sub>j</sub> ∈ {'A', 'B', 'C'}

$$|S| = 3^6$$

$$d_i^i \in \{0, 1\}, \quad d_1 < d_2 < d_3 < d_4 < d_5 < d_6$$

A = {'AB', 'AC', 'BA', 'BC', 'CA', 'CB'}

```

class TorresHanno; (ModeloBusqueda):
    def __init__(self, n):
        Self.A = ['AB', 'AC', 'BA', 'BC', 'CA', 'CB']
    def acciones_legales(self, s)
        if len(s) != self.n or intset(s).subset(['A', 'B', 'C'])
            raise ValueError(" ")
        acciones_validas = []
        for a in self.A
            de, hacia = a[0], a[1]
            comprobar que hay discos para sacar y el otro poste esté vacío o con una más grande
            if de in S and (hacia not in S or
                s.index(de) < s.index(hacia)):
                acciones_validas.append(a)
        return acciones_validas
    def transicion(self, S, a)
        costo_local = 1
        if a not in self.acciones_legales(S):
            raise ValueError(" ")
        Sn = S[:] Agarro lista de estados
        Sn[S.index(a[0])] = a[1] Buscas el estado y se cambia
        return Sn, costo_local

```

## FUNCTION BÚSQUEDA

	DFS	BFS	IDS
completa	si	si	si
óptima	si	no, $\star$	no, $\star$
tiempo	$O(b^{d_{\max}})$	$O(b^{d_{\min}})$	$O(b^{d_{\min}})$
memoria	$O(b d_{\max})$	$O(b^{d_{\min}+1})$	$O(b d_{\min})$

$\star$  si  $d^* = d_{\min}$  si (costos locales uniformes)

IDS: un híbrido entre DFS y BFS

$\star$  solo si  $d^* = d_{\min}$

$d_{\max}$ : la profundidad máxima a la que llegamos  
 $d_{\min}$ : profundidad mínima

$$d_{\min} = 0 \quad 1$$

$$d_{\min} = 1 \quad 1+b$$

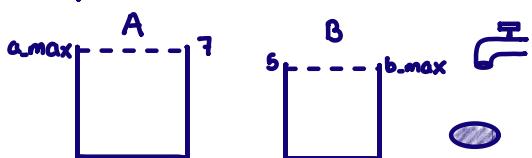
$$d_{\min} = 2 \quad 1+b+b^2$$

:

$$\begin{aligned} d_{\min} \rightarrow & 1+b+b^2+\dots+b^{d_{\min}} \\ d_{\min} + (d_{\min})b + (d_{\min}-1)b^2 + & \dots + b^{d_{\min}} \end{aligned}$$

## BÚSQUEDAS INFORMADAS

Ejemplo.



Acciones

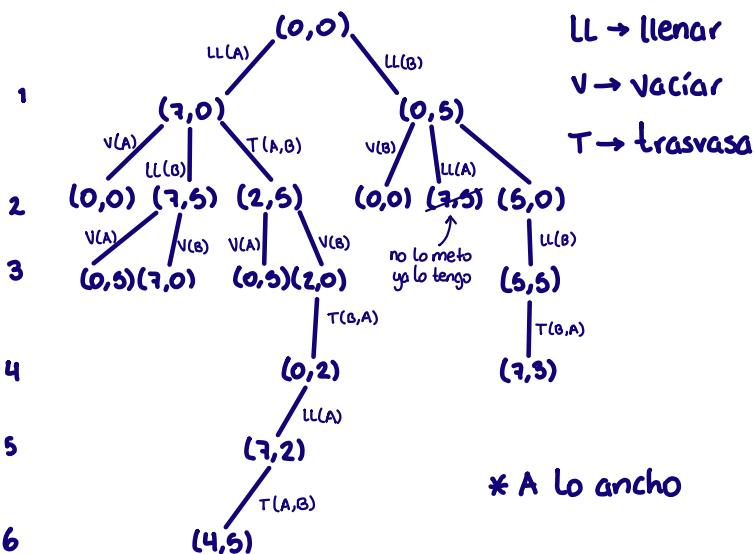
llena (A)	llena (B)	transversa (A,B)
Vacia (A)	Vacia (B)	transversa (B,A)

Estados

$$x = \begin{pmatrix} 0-7 \\ 0-5 \end{pmatrix} \quad \text{Lo que haga en A y B}$$

$$|x| = 8 \times 6 = 48$$

Problema: tener 4 en A y 5 en B



modelo

acciones\_legales

transición

costo\_local

Problema

estado\_inicial

estados\_finales

modelo

UCS : Saca el de menor costo

- [s] [ ]
- [p, d, e] [d, e]
- [d, e, q] [e, q]
- [e, e, q] [e, q]
- [e, e, q] [e, q]
- [f, e, q] [e, q]
- [e, G, q] [G, q]

\* Siempre costos positivos mayores a 0  
→ Heurística: solución que queremos tener

$$d^* \rightarrow C^* \leftarrow \text{costo óptimo}$$

C: mínimo costo local

$$b > d^*, d_{\min}$$

	costo $c(n)$ UCS	$h(n)$ GREEDY	$c(n) + h(n)$ costo + h A*
completo	sí		
Óptimo	sí		
Tiempo		$O(b^{\lceil \frac{c_m}{\epsilon} \rceil})$	
Memoria		$O(b^{\lceil \frac{c_m}{\epsilon} \rceil})$	

### BÚSQUEDA HEURÍSTICA

Dar un estimado del costo total final.

### BÚSQUEDA GREEDY

Solo toma en cuenta  $h(n)$

El costo de la heurística siempre debe ser menor al de la solución.

→  $h$  admisible

### BÚSQUEDA A\*

Notación

$g(n)$  = Costo al nodo  $n$

$h(n)$  = costo estimado de  $n$  a la meta más cercana

$f(n) = g(n) + h(n)$  = estimado total del costo

$G^*$  = Costo más bajo

- $n$  es un plan
- $g$  es un plan ta  $g$ .estado  $\in S_f$
- $h$  es una heurística admisible si  $h(n) \leq c(n)$  tq  $n$  es un plan donde  $n$ .estado  $\forall n$
- $g^*$  es un plan óptimo
- $C^*$  es un costo óptimo
- $h(g) = 0 \quad \forall g$
- $g^*.costo = C^* \leq g.costos \quad \forall g$

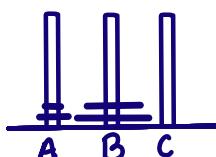
Para A\* los planes se ordenan en la frontera como  $f(n) = n.costo + h(n)$

Antes de sacar  $g^*$  de la frontera voy a revisar TODOS los planes tq  $n.costo + h(n) \leq C^*$

Supongamos un plan completo NO ÓPTIMO

$$g.costos > g^*.costo \rightarrow g.costos + h(g) = g.costos > C^*$$

### EJEMPLO TORRE DE HANOI



$$S_f = \{(C, C, C, C)\}$$

$$S_0 = \{(A, A, A, A)\}$$

Suponiendo que no hay restricciones de poner  $\rightleftarrows$  y sacar el de encima

$$h(n) = \sum (1 \text{ for } v \text{ in } n.\text{estado if } v \neq C)$$

$$h(n) = \max(0, 2N_A - 1) + \max(0, 2N_B - 1)$$

$$N_A := \# \text{discos en el poste A}$$

## EJEMPLO 8 - PUZZLE

7	2	4
5		6
8	3	1

$$S = (7, 2, 4, 5, 0, 6, 8, 3, 1)$$

$$A = (\uparrow, \downarrow, \leftarrow, \rightarrow)$$

$$S_f = (1, 2, 3, 4, 5, 6, 7, 8, 0) \neq (0, 1, 2, 3, 4, 5, 6, 7, 8)$$

$$\frac{9!}{2} = 181,440$$

$$\frac{16!}{2} = 10.46 \times 10^{12}$$

$$\frac{25!}{2} = 7.755 \times 10^{34}$$