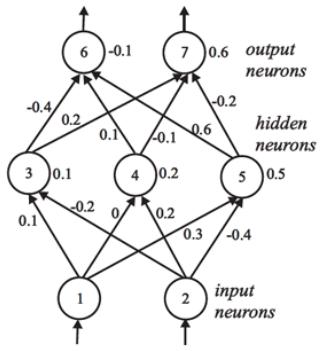
# Ejercicios Redes Neuronales:

## Ejercicio 1 febrero 2014/15:

7. Redes neuronales:

   *a*) (1 punto) Considera un perceptrón con función de activación *CUADRÁTICA* y pesos $\vec{w} = (0.1, 0.2, 0.3)$ y un conjunto de entrenamiento $D = \{E1, E2\}$ con $E1 =< (0.5, 0.5), 0.3 >$ y $E2 =< (0.4, 0.7), 0.2 >$.
   - Calcula el error cuadrático del perceptrón con pesos $\vec{w}$ sobre el conjunto de entrenamiento D.
   - Calcula en valor del gradiente de la función anterior $\vec{\nabla} E(\vec{w})$ para el vector de pesos $\vec{w}$.

   *b*) (1.5 punto) Teniendo la red neuronal siguiente, de función de activación LINEAL($\sigma(x) = x$), tasa de aprendizaje $\mu = 0.1$ y los pesos como se indican en el dibujo.
   - Realizar el cálculo hacia delante de la señal, usando como ejemplo $T1 = \{0.6, 0.1, 0, 1\}$
   - Realizar el backpropagation y calcular el cambio de los pesos.




a) Tenemos que la función de activación es Cuadrática, por lo tanto: g(x) =x^2
g'(x)=2*x
Los pesos son: w0=0.1, w1=0.2, w2=0.3, y la tabla de datos es la siguiente:

| X0 | X1 | X2 | Y |
|---|---|---|---|
| -1 | 0.5 | 0.5 | 0.3 |
| -1 | 0.4 | 0.7 | 0.2 |

La primera parte consiste en determinar el error cuadrático del perceptrón, que viene dado por la diferencia del valor de salida esperada con la salida predicha del perceptrón:

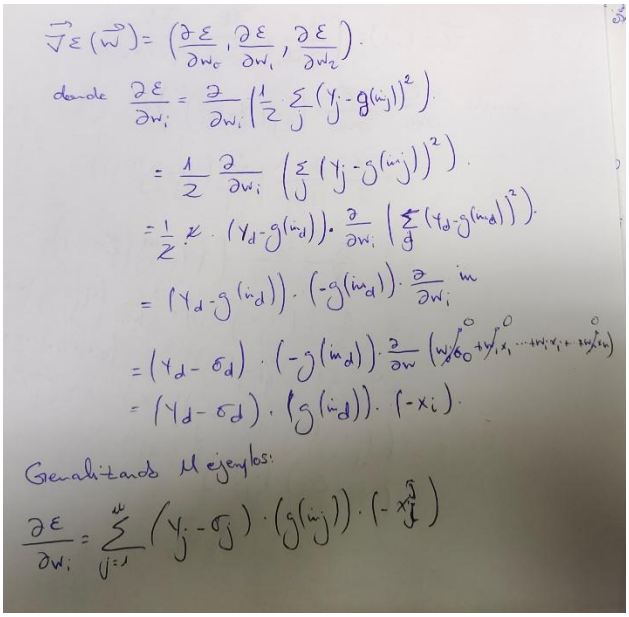
E(w)= 0.5*(sumatorio (Yj - Oj)^2) siendo j=1 hasta el número de ejemplos (m)

Para ello, calculamos la salida predicha del perceptrón Oj en los dos ejemplos:
IN1 = w0*x0 + w1 * x1 + w2*x2 = (0.1)*(-1) + (0.2)*(0.5)+(0.3)*(0.5) = 0,15
Y como Oj = g(INj) = x^2, entonces O1 = 0,0225

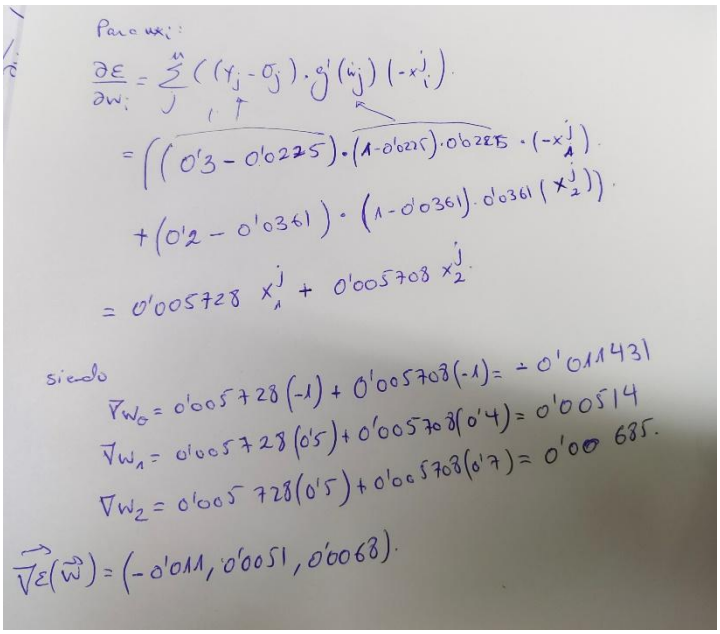
Repetimos el mismo procedimiento para el ejemplo dos:
IN2 = w0*x0 + w1 * x1 + w2*x2 = (0.1)*(-1) + (0.2)*(0.4)+(0.3)*(0.7) = 0,19
O2 = 0,0361

Ahora, calculamos el error, siendo el sumatorio de los dos ejemplos:
E(w) = 0.5 * ((0.3 – 0.0225)^2 + (0.2 – 0,0361)^2) = 0.5 *(0,0770 + 0,02686) = 0,05196

El valor de gradiente:

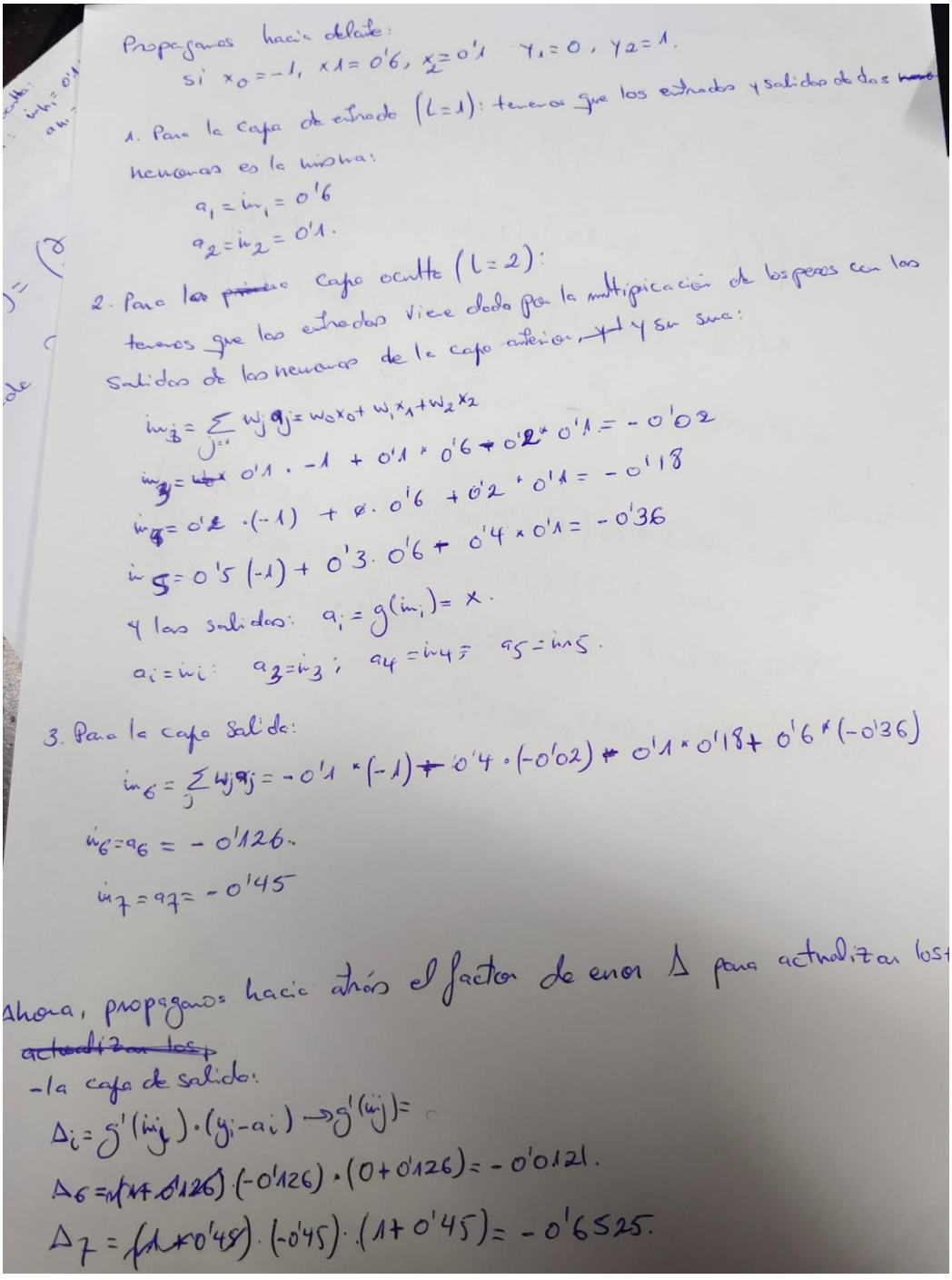



Como tenemos dos ejemplos, entonces m= 2, calculamos el valor de gradiente correspondiente a cada xi:



b) La función de activación es lineal, ósea: g(x) = x  g'(x) = 1

Realizamos la propagación de los valores hacia delante:



Propagamos hacia delante:

si $x_0 = -1$, $x_1 = 0'6$, $x_2 = 0'1$  $Y_1 = 0$, $Y_2 = 1$.

1. Para la capa de entrada ($l = 1$): tenemos que las entradas y salidas de las neuronas es la misma:

$a_1 = in_1 = 0'6$

$a_2 = in_2 = 0'1$.

2. Para la capa oculta ($l = 2$):

tenemos que las entradas viene dada por la multiplicación de los pesos con las salidas de las neuronas de la capa anterior, $x_1$ y su sma:

$in_3 = \sum_{j=1} w_j a_j = w_0 x_0 + w_1 x_1 + w_2 x_2$

$in_3 = 0'1 \cdot -1 + 0'1 * 0'6 + 0'2 * 0'1 = -0'02$

$in_4 = 0'2 \cdot (-1) + 0 \cdot 0'6 + 0'2 * 0'1 = -0'18$

$in_5 = 0'5 (-1) + 0'3 \cdot 0'6 + 0'4 * 0'1 = -0'36$

y las salidas: $a_i = g(in_i) = x$.

$a_i = in_i$:  $a_3 = in_3$;  $a_4 = in_4$;  $a_5 = in_5$.

3. Para la capa salida:

$in_6 = \sum_j w_j a_j = -0'1 * (-1) + 0'4 \cdot (-0'02) + 0'1 * 0'18 + 0'6 * (-0'36)$

$in_6 = a_6 = -0'126$.

$in_7 = a_7 = -0'45$

Ahora, propagamos hacia atrás el factor de error $\Delta$ para actualizar los pesos

-la capa de salida:

$\Delta_i = g'(in_j) \cdot (y_i - a_i) \rightarrow g'(in_j) =$

$\Delta_6 = (1 + 0'126) \cdot (-0'126) \cdot (0 + 0'126) = -0'0121$.

$\Delta_7 = (1 + 0'45) \cdot (-0'45) \cdot (1 + 0'45) = -0'6525$.

Para las capas ocultas:

$\Delta_i = g'(in_i) * \sum_j w_{ji} \Delta_j$

$\Delta_3 = -0'02 * ((+0'4 * 0'0121) + (-0'2 * 0'6525)) = 7,54 \cdot 10^{-3}$

$\Delta_4 = -0'18 * ((-0'1 * 0'0121) + (0'4 * 0'6525)) = -0'0115$

$\Delta_5 = -0'36 * ((-0'6 * 0'0121) + (0'2 * 0'6525)) = -0'044$

Actualizamos los pesos: ($\gamma = 0'1$)

$w_{ji} = w_{ji} + \gamma a_j \Delta_i$

$w_{36} = w_{36} + \gamma a_3 \cdot \Delta_6 = -0'4 + 0'1 \cdot (-0'06) \cdot (-0'0121) = -0'3999$

$w_{37} = 0'2 + 0'1 \cdot (-0'06) \cdot (-0'6525) = 0'2039$

$w_{03} = w_{03} + \gamma x_0 \cdot \Delta_3$

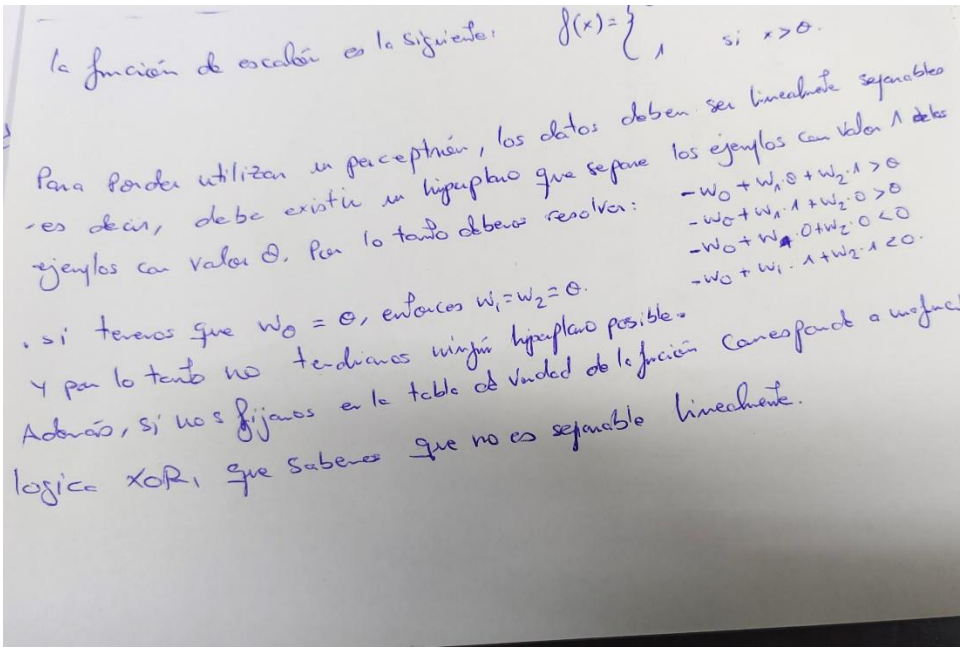
Ejercicio febrero 2017:

Dado el siguiente conjunto de datos:

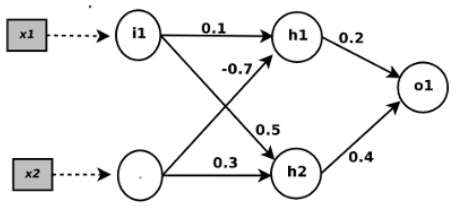
| | Entradas | | Salida |
|---|---|---|---|
| | $x_1$ | $x_2$ | $t_1$ |
| $e_1$ | 0 | 1 | 1 |
| $e_2$ | 1 | 0 | 1 |
| $e_3$ | 1 | 1 | 0 |
| $e_4$ | 0 | 0 | 0 |

Perceptrón

*a*) (0.5 puntos) Demuestra analíticamente si se puede entrenar un perceptrón con función de activación escalón. Si no es posible, explicar por qué y si es posible, dar unos posibles valores de los pesos.



Si consideramos la siguiente red multicapa con función de activación sigmoide y tasa de aprendizaje a 0.25.




*a*) (2 puntos)Describe DETALLADAMENTE el proceso de aprendizaje back-propagation para una red perceptrón multicapa

*b*) (1.5 puntos) Cómo sería 1 ciclo completo de aprendizaje para la red anterior?

El proceso de aprendizaje en una red multicapa, consiste en dado un conjunto de ejemplos, y una salida esperada, determinar los pesos de la red de manera que la función que calcula la red se ajuste lo mejor posible a los ejemplos.

El algoritmo utilizado para aprender dichos pesos se denomina retropagación, que consiste en lo siguiente:

1. Inicializar los pesos aleatoriamente
2. Repetir hasta alcanzar la convergencia:
3. Para cada ejemplo del conjunto:
4. Propagar hacia delante
5. Propagar un factor de error hacia detrás
6. Devolver los pesos

El 4 de propagar hacia delante :

1. Para la capa de entrada:
   Calcular ai siendo igual ini
2. Para las demás capas:
   Ini = sumatorio ( wi*ai)
   ai = g(Ini)

El 5 de propagar el factor de error hacia atrás:

1. Para la capa de salida:
   Para cada neurona:
   Ai = g'(Ini)*(yi – oi)
2. Para el resto de capas:

   Para cada neurona j:

   Ai = g'(Ini) * sumatorio(WjiAj)
   Para cada neurona i de la capa l+1
   Wji = Wji + n * aj * Ai

Algoritmo descenso por gradiente: es un algoritmo iterativo de aproximaciones sucesivas para encontrar los pesos de la red tal que minimicen el error total y se ajuste lo máximo a los ejemplos, para aproximar los pesos se desciende por el gradiente AE(w), para actualizar los pesos se avanza en dirección del máximo decremento.

1. Inicializar los pesos aleatoriamente
2. Repetir hasta cumplir un criterio de convergencia
3. Inicializar los Awi a 0
4. Para cada ejemplo:
5. AWi = AWi + n * g'(inj)*(yj-oj)*(-xj)
6. Para cada peso wi:
7. Wi = Wi + AWi
8. Devolvemos W

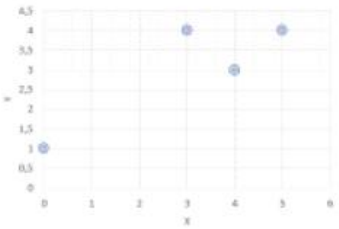
Regla delta: en lugar de tratar de modificar el error cuadrático total cometido para todos los ejemplos, se trata de proceder incrementalmente de descender el erro cuadrático sobre el ejemplo que se está tratando en cada momento.

wi = wi + n * g'(inj) )*(yj-oj)*(-xj)

## Ejercicios Regresión lineal:

1. Consideramos la siguiente tabla numérica:

| X | Y |
|---|---|
| 5 | 4 |
| 3 | 4 |
| 0 | 1 |
| 4 | 3 |



Vamos a aplicar técnicas de regresión lineal.

*a*) Describir el algoritmo de descenso por gradiente.

*b*) Aplicar una iteración del algoritmo y dar los valores nuevos, suponiendo que partimos de todos los coeficientes con valor 0.

El algoritmo de descenso por gradiente busca el conjunto de parámetros que minimizan la función de coste, que consiste en una serie de aproximaciones sucesivas en dirección opuesta al gradiente del error. W = w – alfa * dE/dw

Hay que tener en cuenta el valor de alfa, ya que si es muy grande los cambios de la pendiente serian grandes y la determinación de los valores de los coeficientes que minimicen la función de coste seríaseriasería difícil, en caso contrario, si es muy pequeña, la pendiente desciende de forma lenta, y el algoritmo tarda mucho en encontrar la solución.

1. Inicializamos los valores de los coeficientes aleatoriamente.
2. Repetir hasta alcanzar un criterio de convergencia:
3. Para cada teta
4. Actualizamos los valores de cada O siendo O -= alfa * Ao J(O) → dJ(O)/dOi = dJ(O)/dOi * ½ * (ho(x) – y)^2 ) = (ho(x) – y)) * dJ(O)/dOi * (ho(x) – y) = (ho(x) – y) * Xi

b)

tenemos que O0=0=O1 y alfa suponemos que es 0.001

y el vector X es: [1 5; 1 3; 1 0; 1 4], Y= [4; 4; 1; 3]

para todos los datos de entrada tenemos que: ho(X) = O0 * 1 + O1* x1 = 0

los valores de theta vienen dados como hemos comentado antes por:

O -= alfa * sumatrorio((ho(X) – Y ) * X)

O = O – alfa * [ho(1) – y1)*1 + (ho(1) – y2)*1 + (ho(1) – y3)*1 + (ho(1) – y4)* 1 ; (ho(x1) – y1)*x1 + (ho(x2) – y2)*x2 + (ho(x3) – y3)*x3 + (ho(x4) – y4)*x4]

O = [ 12*alfa; 44*alfa]

2. Regresión lineal (2 puntos).
Teniendo en cuenta la siguiente tabla:

| x | y |
|---|---|
| 1 | 12 |
| 2 | 9 |
| 3 | 4 |

1

Asume H(x) inicial con $\theta_0 = 0$ y $\theta_1 = 0$

*a*) Aplicar el algoritmo de descenso por gradiente con la primera iteración completa y recalcular todos los coeficientes con ratio de aprendizaje 0.25. **(1 punto)**.

*b*) ¿Cuál es el valor inicial del error cuadrático para la tabla?
¿Cuál será el error cuadrático final para una entrada = 25?
¿Qué ocurre si la tasa de aprendizaje es muy alta o muy baja?
Dibuja el posible efecto sobre el error.
Dibuja la regresión obtenida y los puntos de la tabla inicial. (1 punto ).

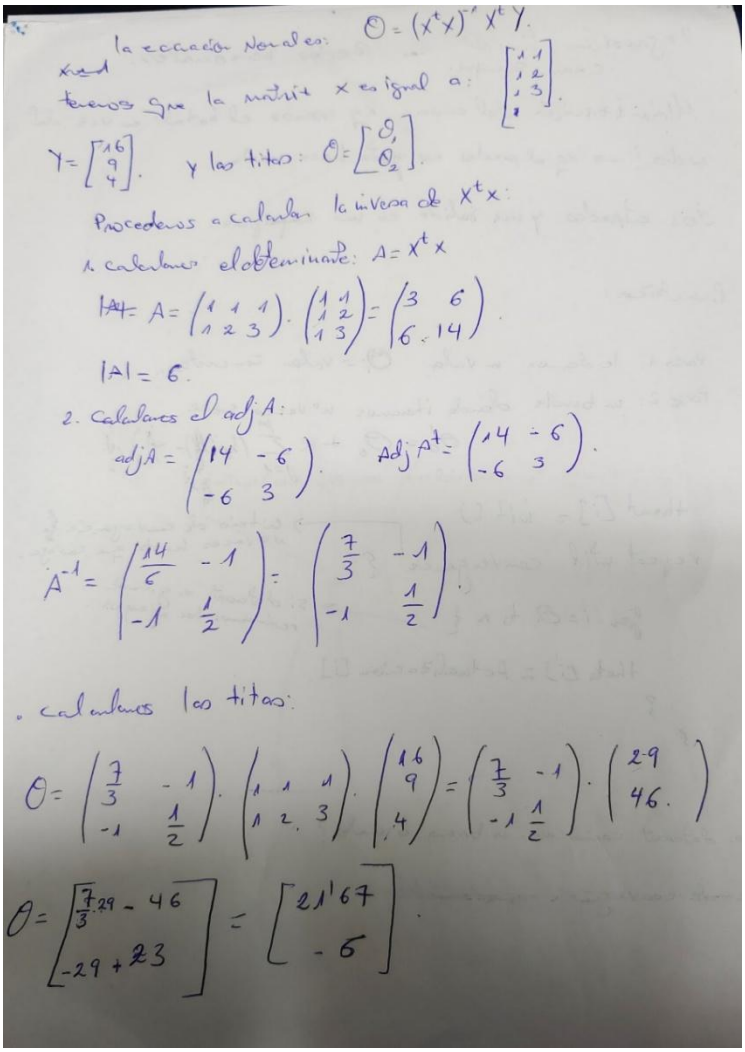
a) Igual que el apartado anterior.
b) Error Cuadrático:
    1. Medio = ½*m *(sumatorio(ho(xi) – yi)^2)
    2. Total = ½ * *(sumatorio(ho(xi) – yi)^2)
    3. Individual = (ho(xi) – yi)^2

Descendiente por gradiente estocástico: actualiza los valores de los coeficientes en cada iteración en función de un ejemplo aleatoria del dataset, no de todos los ejemplos, alcanzando la convergencia más rápido.

Ecuaciones Normales: O = (Xt X)^-1 * X^t * Y

XO = Y → X^-1 * X * O = X^-1 * Y → O = X^-1 * Y , al ser X de dimensión m*n

X^t * Xm*n = Xn*n.




**Maquias de vector soporte** se aplica en problemas de aprendizaje supervisado, tanto en regresión como en clasificación, la función kernel se utiliza en conjuntos de datos no separables linealmente, que consiste en añadir una dimensión más al conjunto de datos para poder encontrar una separación lineal (hiperplano que los divida).

Vector Normal(W) : es una recta perpendicular al hiperplano de separación y define su orientación, que sirve para determinar en q lado del hiperplano, se encuentra un punto.

Producto escalar (w.x + b = 0): sirve para indicar la posición relativo del punto respecto al hiperplano, si es mayor que 0 esta en un lado, y si es menor esta en el otro lado.

Ejemplo: w1x1 + w2x2 + b = 0 , si el hiperplano es w=[1,-1] b=-1

Case positivo: punto x=[2; 3]

Caso negativo: punto x=[4, 1]

# Adquisición de Conceptos:

Espacio de Hipotesis: el conjunto de todas las hipotesis que se pueden escribir en nuestro lenguaje.

Espacio de Versiones: el conjunto de todas las hipotesis consistente con el conjunto de ejemplos.

1. List then eliminate: algoritmo de fuerza bruta, generamos todo el espacio de hipotesis posibles, y despues descartamos las que no son consistente con los ejemplos (los que rechazan a los positivos o adimten a los negativos).
    1. Incializamos el espacioVersiones a [] y espacioHipotesis = generaH(Dataset)
    2. Para cada h en espacioHipotesis:
    3. Comprobar si es consistente con el dataset, si lo es añadir al espacioVersiones
    4. Devolver espacioVersiones

2. Find- S: establecemos la hipotesis mas pequeña que cubra a todos los positivos, a partir de una hipotesis especifica genearlizamos para cubrir a los ejemplos positivos que no estan cubiertos.
    1. H = (EMPTY, EMPTY,.....,EMPTY)
    2. Para cada ejemplo en Dataset:
    3. Si es positivo
    4. Si ¡Consistente(ejemplo,H): Generalizar(H)
    5. Devolver H
3. Dual Find-S: a lo contrario que el anterior, partimos de la hipotesis mas general, y vamos especificando por cada ejemplo negativo que sea cubierto por H:
    1. H = (ALL, ALL,....,ALL)
    2. Para cada ejemplo en Dataset:
    3. Si es negativo:
    4. Si Consistente(ejemplo, H): Especificar(H)
    5. Devolver(H)

4. Eliminación de candidatos: ya que los dos anteriores son costosos, este algoritmo mantiene un cota superior e inferior de todas las hipotesis consistente con el ejemplo, devulve como salida: espacio de versiones que contiene (un conjunto de hipotesis genericas minimales y otros especificas maximales).
    1. Inicializar G=(ALL, ALL, .....,ALL), S=(EMPTY, EMPTY, ....., EMPTY)
    2. Para cada ejemplo en Dataset:

```
3.      Si es Positivo(ejemplo) :
4.          Eliminar de G todas las hipotesis incosistente con el ejemplo
5.          Para hipotesis s en S:
6.              Si no es Consistente(ejemplo):
7.                  Eliminar s de S  → S.remove(s)
8.                  Generalizar s  → gms =generalizar(s)
9.                  Para cada hipotesis g en gms:
10.                     Si es consistente(ejemplo) && existe una mas general(G,g)
11.                         S añadir g
12.                 Para cada s1,s2 en S:
13.                     Comprobar que ninguna sea mas general que otra
14.                     If general(s2,s1): S.remove(s2)
15.     Sino es Negativo(ejemplo)
16.         Eliminar de S todas las hipotesis  incosistente con el ejemplo
17.         Para hipotesis g in G:
18.             Si no es Consistente(g,ejemplo):
19.                 G.remove(g)
20.                 Especificar g →  sms= especificar(g)
21.                 Para cada hipotesis s in sms:
22.                     Si no consistente(ejemplo) && existe mas especifica en S
23.                         G.añadir(g)
24.                 Para cada g1,g2 en G:
25.                     Comprobar que no haya ninguna menos general que
    otra
26. Devolver V = (G,S)
```

El proceso de generalización consiste en convertir una hipotesis h en un g de forma que g >g h, para incorporar los ejemplos positivos.

El proceso de especificación consiste en convertir una hipotesis h en un g de forma que g >s h, para excluir los ejemplos negativos.

## Ejemplos de resolucion:

## Find-S:

| Origen | A(frica) | A(frica) | A(frica) | E(uropa) | A(frica) |
|---|---|---|---|---|---|
| Clase | Mamífero | Reptil | Reptil | Mamífero | Mamífero |
| Alimentación | Carnívoro | Herbívoro | Herbívoro | Herbívoro | Carnívoro |
| Valor | Alto | Bajo | Alto | Bajo | Normal |
| Situación | Peligro | Normal | Peligro | Peligro | Peligro |
| Ejemplo | + | - | + | - | + |

**Conjunto de entrenamiento**

En el espacio de hipótesis $H$ cada hipótesis está formada por una conjunción de restricciones sobre los valores de sus atributos.

**Inicio**
Se inicializa $h$ a la hipótesis más específica $\Rightarrow$ $\boldsymbol{h_0 = (\varnothing, \varnothing, \varnothing, \varnothing, \varnothing)}$

**Primer ejemplo** (A, M, C, A, P; +) $\Rightarrow$ $\boldsymbol{h_1 = (A, M, C, A, P)}$

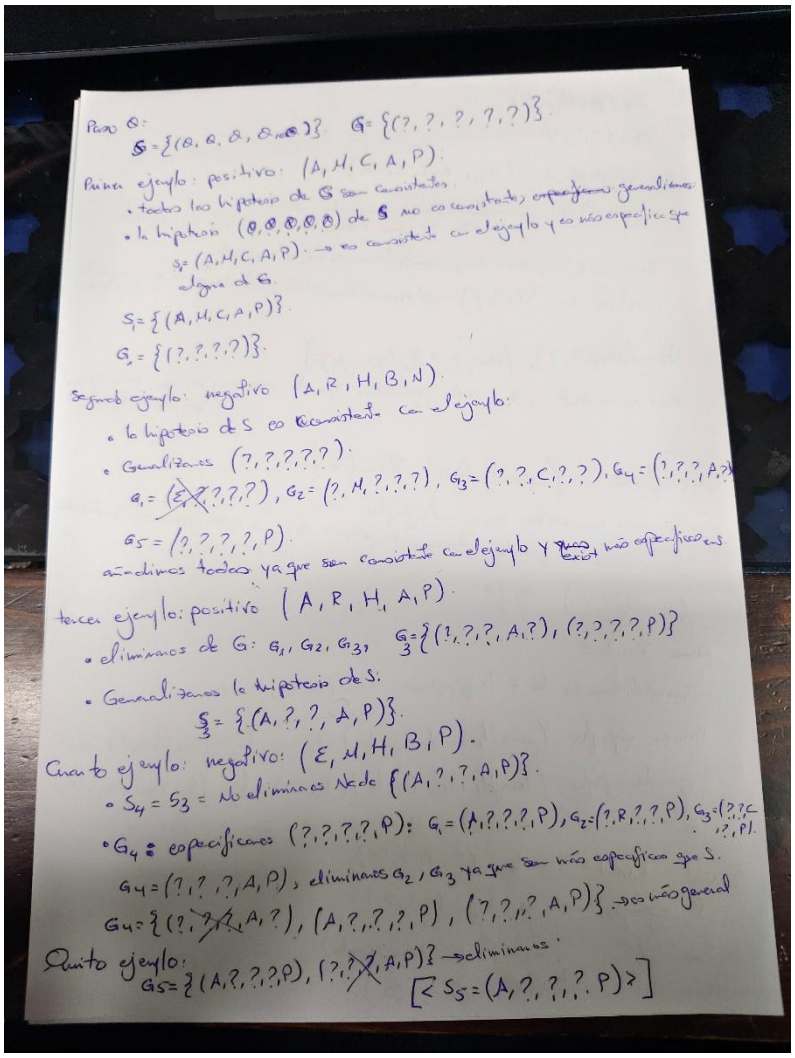
**Segundo ejemplo** (A, R, H, B, N; -) $\Rightarrow$ $\boldsymbol{h_2 = h_1 = (A, M, C, A, P)}$

**Tercer ejemplo** (A, R, H, A, P; +) $\Rightarrow$ $\boldsymbol{h_3 = (A, x2, x3, A, P)}$

**Cuarto ejemplo** (E, M, H, B, P; -) $\Rightarrow$ $\boldsymbol{h_4 = h_3 = (A, x2, x3, A, P)}$

**Quinto ejemplo** (A, M, C, N, P; +) $\Rightarrow$ $\boldsymbol{h_5 = (A, x2, x3, x4, P)}$

## Eliminacion de candidatos:




## Programación lógica inductiva

FBF (Formulas bien reformadas): son exprersiones lógicas que se construyen combinando proposiciones simples y conectivas lógicas, una FBF puede ser:

1. Una proposición (p)
2. La negación de una FBF (¬p)
3. La conjunción de dos FBF (p ^ q)
4. La disyunción de dos FBF (p  or q)
5. La implicación entre dos FBF (p → q)
6. La doble implicación entre dos FBF (p ←→ q)

| Q → P |
|---|
| V |
| V |
| F |

| P | Q | P ∨ Q | P ∧ Q | ¬ P | P→Q | P↔Q |
|---|---|---|---|---|---|---|
| V | V | V | V | F | V | V |
| V | F | V | F | F | F | F |
| F | V | V | F | V | V | F |
| F | F | F | F | V | V | V |

V

FNC (Forma normal conjuntiva): es posible transformar de una forma lógica aplicando transformaciones, las características de una FNC son:

1. No contiene implicaciones ni doble implicaciones.
2. Las negaciones se aplican solo a las proposiciones simples.
3. Las expresiones están formadas por conjunción de disyunciones.

I. Sustituir toda implicación $(a \implies b)$ por $(-a \vee b)$.

II. Sustituir toda doble implicación $(a \leftrightarrow b)$ por $(a \wedge b) \vee (\neg a \wedge \neg b)$.

III. Sustituir las expresiones de tipo $\neg(a \vee b)$ por $(\neg a \wedge \neg b)$

IV. Sustituir las expresiones de tipo $\neg(a \wedge b)$ por $(\neg a \vee \neg b)$

V. Aplicar la propiedad distributiva para expresar la fórmula como conjunción de disyunciones.

Existen dos estrategias para la obtención de conocimiento:

- Deducción (razonamiento hacia delante): consiste en obtener nuevas afirmaciones a partir de las que ya tenemos mediante un motor de inferencia.
- Demostración (razonamiento hacia detrás): consiste en explicitar nuestro conocimiento que deseamos obtener, y comprobar si es consecuente con nuestra base de conocimientos

Una clausula de horn: es una disyunción de literales en lo que hay a lo sumo un único literal positivo (cabeza, y cuerpo serian el conjunto de los literales negativos):

$$\neg p1 \vee \neg p2 \vee \neg p3 \vee \cdots \vee \neg pn \vee q$$

Programa lógico: es un conjunto de cláusulas de horn.

- es consistente con los ejemplos negativos si cada ejemplo negativo no puede ser deducido a partir del programa
- es completo con los ejemplos positivos, si todos los ejemplos se pueden deducir a partir del programa.

Componentes de un Programa Lógico indutivo:

1. conjunto de ejemplos positivos e+
2. conjunto de ejemplos positivos e-

3. Dominio del conocimiento: un programa lógico consistente T, a partir de la cual no se pueda deducir al menos uno de los ejemplos positivos.

El Objetivo es encontrar un programa lógico que sea completo con e+ y consistente con e-.

FOIL: es una estrategia de la programación lógica inductiva que obtiene una cláusula de horn que describan los predicados a que se refiere los ejemplos del problema, parte de una cláusula más general (que no es consistente con todos los ejemplos negativos) e ir especificando hasta hallar la consistencia.

1. listaReglas = []
2. mientras el conjunto de e+ no esta vacio:
3. regla←creaRegla(predicado)
4. Mientras (regla cubre e-)
5. Literales ← generarLiterales(BC, regla)
6. Mejor ← mejor(Literales)
7. Regla.add(L)
8. listaReglas.añadir(regla)
9. eliminarCubiertos(E, regla)
10. Devolver listaReglas

$$ganancia = t \cdot (log_2(\frac{p_{R'}}{p_{R'} + n_{R'}}) - log_2(\frac{p_R}{p_R + n_R}))$$

Ejercicio:

3. Considérese el siguiente problema de programación lógica inductiva:

   - Ejemplos positivos: q(a,d) q(a,c) q(c,b) q(b,d) q(d,b)
   - Ejemplos negativos: q(a,b) q(c,d) q(d,d)
   - Conocimiento base: h(a) h(b) m(c) m(d) r(a,b) r(c,d)

   Supongamos que el algoritmo FOIL aplicado a este problema se encuentra en el bucle interno, con la regla $q(X, Y) : -h(X).$, parcialmente construida. Incluirá esta regla en el conjunto de reglas a devolver o continuará añadiendo condiciones? Si es así: ¿Cuáles son los posibles literales candidatos a ser añadidos y cuál se elegirá?. Una vez completada esta regla ¿será la única regla que devuelva el algoritmo como salida? Justificar todas las respuestas.

Ejercicio 5:

Primer Paso: pintamos de la regla más general (que cubre todos los ejemplos)

$q(A,X):-.$ número de ejemplos Positivos = 5
número de ejemplos negativos = 3

Conjunto de literales candidatos a añadir a esa regla para:

$q(A,X):-h(A).$ $q(A,X):-h(X).$ $q(A,X):-m(A).$

$q(A,X):-m(X).$ $q(A,X):-r(A,X).$ $q(A,X):-r(A,A).$

$q(A,X):-r(X,A).$ $q(A,X):-r(A,Z).$ $q(A,X):-r(Z,A).$

$q(A,X):-r(X,X).$ $q(A,X):-r(Z,X).$ $q(A,X):-r(X,Z).$

- si añadimos $h(A)$: calculamos la ganancia:

$q(A,X):-h(A).$

| t | Pp | Np | Ganancia |
|---|---|---|---|
| 3 | 3 | 1 | $3*(\log(\frac{3}{4})-\log(\frac{5}{8}))=$ |

- Se continúa añadiendo más condiciones a esa regla, ya que cubre a un ejemplo negativo del dataset. $\begin{cases} Np' = 1 \\ Pp' = 3 \end{cases}$

Los posibles literales son los anteriores.

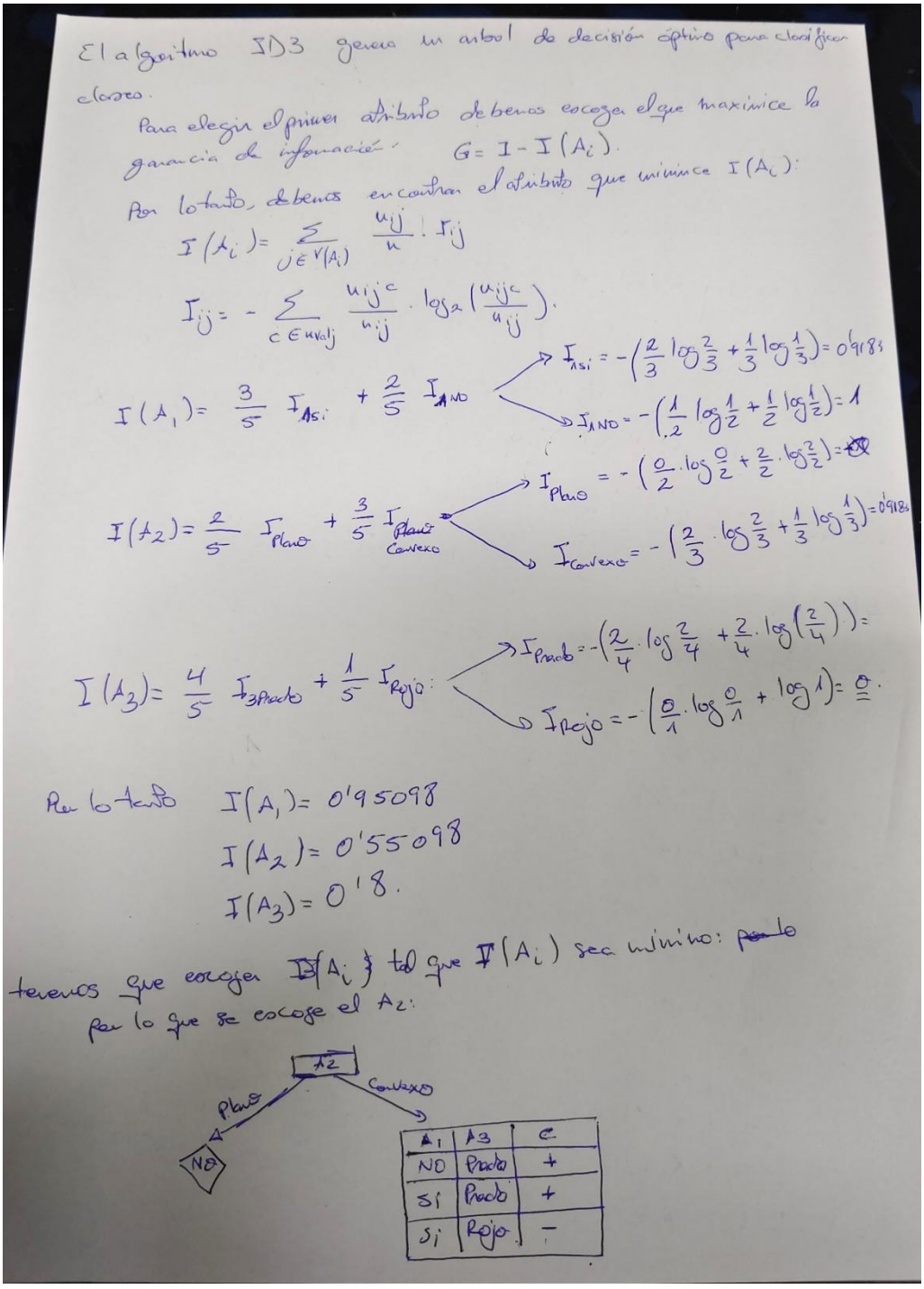
| t | Pp | Np | Ganancia |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 3 | 3 | 1 | $3*(\log\frac{3}{3}-\log\frac{3}{4})=$ |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | $2*\log\frac{2}{2}-\log\frac{3}{4}$ |
| 0 | 2 | 0 | $1*\log\frac{1}{1}-\log\frac{3}{4}$ |
| 2 | 1 | 0 | |
| 1 | 0 | 0 | $1*\log\frac{1}{1}-\log\frac{3}{4}$ |
| 0 | 1 | 0 | |
| 1 | 2 | 0 | $2*\log\frac{2}{2}-\log\frac{3}{4}=$ |
| 2 | | | |

$q(A,X):-h(A),h(X).$
$q(A,X):-h(A),m(A).$
$q(A,X):-h(A),m(X).$
$q(A,X):-h(A),r(A,X).$
$q(A,X):-h(A),r(X,A).$
$q(A,X):-h(A),r(A,A).$
$q(A,X):-h(A),r(A,Z).$
$q(A,X):-h(A),r(Z,A).$
$q(A,X):-h(A),r(X,X).$
$q(A,X):-h(A),r(X,Z).$
$q(A,X):-h(A),r(Z,X).$

Se elegirá la regla $q(A,X):-h(A),m(X).$ por que es la regla con mayor ganancia, al ~~ser~~ no cubrir ningún ejemplo negativo. Se ~~debe~~ añadirá esa regla al conjunto de reglas. Además, hay que buscar nuevas reglas que cubran a los ejemplos positivos que quedan: $q(a,c).$ $q(c,b).$

# Extracción de reglas:

## ID3:




Poda de Árboles de decisión: los arboles de decisión se podan para evitar sobreajuste, simplificar el modelo y mejorar la eficiencia, existen dos formas para podar:

1. Pre-Poda: establecer un límite de ramas para evitar que crezca demasiado.
2. Post-Poda: construir el árbol por completo y eliminar las ramas que no contribuyen significativamente en el rendimiento.

## AQ:

El algoritmo AQ es una estrategia de extracción de reglas, que permite generar una lista de reglas que cubren a todos los ejemplos positivos y excluyen a los negativos del conjunto de entrenamientos, es decir, que sean consistentes.

Está compuesto por:

1. Un conjunto de atributos
2. Un conjunto de valores de los atributos.
3. Un conjunto de entrenamiento.
4. Un conjunto de clases.
5. Un criterio de selección de regla (mediante evaluación de función lexicográfica LEF), que puede ser:
    5.1. Cobertura: numero de ejemplos positivos cubiertos por la regla.
    5.2. Coste: coste de evaluar el antecedente.
    5.3. Simplicidad: numero de permisas del antecedente.
    5.4. Generalidad: numero de ejemplos observados entre ejemplos totales.

Elementos son:

- Selectores: una condición que esta compuesta por un valor, un operador y un atributo.
- Complejos: una conjunción de un conjunto de selectores.
- Recubrimientos: una disyunción de un conjunto de complejos.

Pseudocódigo es el siguiente:

1. Inicialmente el recubrimiento está a vacío.
2. Sea P el conjunto de ejemplos + y N el conjunto de ejemplos -.
3. Mientras P no está vacio
4. Escoger un ejemplo de P que será nuestra semilla
5. Sea E = [] y L =[{}]
6. Mientras L no este vacia:
7. E’ es igual al conjunto de complejos resultados de la combinación de L con la semilla
8. Eliminar de E’ los que están en E
9. Para cada complejo en E’
10. Si es consistente con N añadir a E y eliminar de E’
11. Actualizar L a E’

12. Seleccionar una regla de E según el criterio de LEF
13. Eliminar de P todos los ejemplos que cubre la regla
14. Añadir al Recubrimiento
15. Devolver Recubrimiento

## Aprendizaje no supervisado:

Clustering: es la organización de datos sin etiquetas mediante la similtud de grupos, o sea, la formación de elementos de grupos similares, separados de otros distintos.

Cluster: colección de elementos de datos que son similares entre si y diferentes a los elementos de otros grupos.

Para hacer clustering necesitamos una medida de similaridad, que será la distancia entre puntos para saber si están cerca o no, lo relacionamos así:

S(xi,xj)=1/1+d(xi,xj), si están iguales la distancia será 0 y la similtud 1, en caso contrario será infinito y la similtud 0, de forma que s pertenece [0,1].

Para evaluar los clústers existen las dos siguientes funciones de evaluación:

1. Inter-Cluster: las relaciones que ocurren en un mismo grupo, mide que tan cerca están los puntos de datos de un grupo al centroide.
2. Intra_Cluster: define las interacciones entre distintos grupos, la separación significa que los diferentes clústeres deben estar muy alejados.

Algoritmos:

1. Particionales: determinar todos los clústeres a la vez.
2. Jerárquicos: encontrar agrupaciones sucesivas utilizando agrupaciones previamente establecidas.
   2.1. Aglomerativos: comienzan con cada cluster en un grupo separado y los fusionan en grupos más grandes, el dendograma comienza desde el nivel inferior fusionando los clusters más similares o cercanos.
   2.2. Divisivos: comienzan con todos los puntos de datos en un cluster (nodo raíz) y proceden a dividirlo en clusters secundarios más pequeños recursivamente.

K-means: es un algoritmo de agrupamiento particional, que divide los datos en k grupos, donde cada grupo tiene un centro q se denomina centroide, y el objetivo es minimizar la suma de las distancias entre los puntos y el centroide al que pertenecen.

Fortalezas:

1. Es simple y fácil de entender.
2. Es eficiente, complejidad temporal O(tkn) n:número de datos, t: número de iteraciones.
3. Al ser t y k pequeñas, se puede considerar un algoritmo lineal.

Debilidades:

1. Es aplicable si se defina la media, el usuario debe especificar k, es sensible a los outliers.

Outliers: puntos que están demasiado lejos de los otros puntos, que pueden ser errores en el registro de datos o algunos datos espaciales que son diferentes.

1. Eliminar los puntos de datos que estén mucho más lejos del centroide que otros puntos de datos.
2. Realizar un muestreo aleatorio, elegimos un subconjunto pequeño de los puntos de datos y así posibilidad de seleccionar un valor atípico sea menor.

Pseudocodigo:

1. Establecer el numero de k (cuantos clústeres queremos).
2. Escogemos los centroides según k de manera aleatoria
3. Repetir hasta llegar un criterio de convergencia (los centroides dejan de cambiar, los puntos dejan de cambiar de cluster, una disminución mínima en la suma del error cuadtractico..etc).

$$SSE = \sum_{j=1}^{k} \sum_{x \in C_j} d(x, m_j)^2$$

donde:

- $C_j$: es el clúster j.
- $m_j$: es el centroide del clúster j
- $d(x, m_j)$: es la distancia entre el punto x y el centroide $m_j$.

4. Asignamos cada punto a cada cluster según la distancia al centroide.
5. Calculamos los nuevos centroides de cada clúster.

Dendograma: es un árbol binario que muestra típicamente la similitud de grupos creados, donde el nivel k corresponde a la partición de n – k + 1 clusters, si se necesita k clusters toamamos el n-k+1.

Pseudocogido del aglometrativos:

1. Inicializar con cada ejemplo en un cluster separado
2. Mientras haya más de un cluster:
3. Encontrar los clusters más cercanos
4. Fusionarlos

## Otros:

Proceso de un Data Sciene es :

1. Extraer los datos.
2. Limpiar los datos.
3. Procesar los datos.
4. Diseñar nuevos experimentos.
5. Visualizar gráficamente los datos.

Dos etapas de aprendizaje:

1. Adquisición: que tiene como entrada un conjunto de ejemplos con la salida esperada y un objetivo de mejora.
2. Visualización: en la que se el pasa el modelo entrenado, un objetivo diferente y se evalúa el aprendizaje.

Problemas que se resuelven mediante ML:

1. Regresión o Predicción: tratar de encontrar el valor de una función objetivo a partir de un conjunto de ejemplos con el resultado.
2. Clasificación: Agrupar objetos en función a un objetivo, en función de su valor.

Aprendizaje Supervisado: se aprende a partir del conjunto de ejemplos, de los cuales sabemos el comportamiento ideal.

Aprendizaje no Supervisado: no tenemos medidas de cual es el comportamiento idea, pero la medida se basa en la similitud de grupos de datos.

Aprendizaje por refuerzo: el comportamiento deseado se representa mediante una cierta evaluación de los resultados que genera el sistema en su entorno.