INTEL·LIGÈNCIA ARTIFICIAL

I

MACHINE LEARNING

Miquel P. Baztán Grau i Alba Sirvent García

Jesuïtes El Clot
Xavier Vila Guardia
2020-2021

Índex

1. Introducció:

En el nostre treball de recerca, hem volgut centrar-nos en els aspectes més coneguts d'aquest àmbit, com són: l'aprenentatge automàtic i els diferents tipus d'algorismes que hi pertanyen (supervisat, no supervisat i per reforç), i el Minimax que no utilitza l'aprenentatge automàtic, però també forma part del camp de la intel·ligència artificial. A més, dins de cada apartat, hem inclòs les petites implementacions corresponents del nostre treball de camp.

La nostra motivació inicial per tractar aquest tema va sorgir de l'interès personal que tenim tots dos per la tecnologia i els avenços que s'estan produint avui dia en la informàtica. On hi destaca la creació i desenvolupament de les IA's. També volíem investigar amb més profunditat aquest tema i portar-lo a la pràctica nosaltres mateixos en el treball de camp. A més, tots dos estem fent el batxillerat tecnològic i volíem que el nostre treball estigués relacionat amb la modalitat que havíem escollit.

Els objectius principals que volem assolir estan enfocats en el camp pràctic de la programació, és a dir, crear models que aprenguin per ells mateixos a resoldre problemes. També ampliar el coneixement del llenguatge que s'utilitza en el seu desenvolupament (nosaltres treballarem amb Python). Tot i això, per poder arribar a entendre els programes, cal buscar informació prèvia per entendre en què es basa la intel·ligència artificial i les diferents opcions de programació que hi pot haver en cada cas. I tot això implementar-ho de forma pràctica per consolidar tota la informació teòrica adquirida, que és l'objectiu principal del nostre treball.

Els objectius de la part pràctica consisteixen a desenvolupar diferents programes pels diferents apartats. Per l'aprenentatge per reforç, crear un laberint amb un personatge que hagi d'arribar al final. En canvi, en l'aprenentatge supervisat, crear un model de regressió lineal amb una variable. Seguidament, en l'aprenentatge no supervisat, ajuntar núvols de punts en diferents agrupacions. I finalment, pel Minimax, el joc del tres en ratlla.

Per poder programar tot això hem hagut d'utilitzar diverses eines, algunes són en línia i d'altres aplicacions. El projecte està escrit usant el Google Documents i l'esquema inicial està creat amb el Lucidchart Diagrams ([6] vegeu annex 6). Tot el codi que surt en el treball està fet o bé en el Visual Studio Code o bé el Google Collaboratory, on poden treballar dues o més persones a la vegada. I per últim per poder aprendre el llenguatge Python hem utilitzat els mòduls NumPy, Matplotlib.pyplot, Pandas, Random i Pygame.

2. Introducció al tema i preguntes freqüents

La intel·ligència artificial és un mecanisme de la informàtica que ens permet que un programa es comporti de manera “intel·ligent”; com ho faria un humà. Escrivim intel·ligent entre cometes, perquè realment tot el que realitza el programa és un algorisme passat a codi, si més no, per un programador informàtic.

2.1. Què diferencia un humà d’una intel·ligència artificial?

Els humans som els animals més intel·ligents del planeta, no obstant hi ha coses que les màquines fan millor que nosaltres. Un exemple clar serien els grans càlculs que realitza un ordinador.

Aquests estan estructurats de formes diferents: les màquines amb intel·ligència artificial tenen una sèrie d’entrades i sortides de dades que es poden identificar fàcilment. En canvi, cadascuna de les parts del cervell humà pot ser receptor i emissor d'informació.

Una altra diferència és l’emmagatzematge d’aquesta informació. Els humans no tenim una gran capacitat per guardar informació detallada, per això, no som capaços de recordar les coses de la mateixa forma, sempre hi ha petites variacions. Però la intel·ligencia artificial no té aquest problema.

Una altra diferència és la capacitat humana de la multitasca, ja que som capaços de realitzar moltes tasques diferents al mateix temps. En canvi, la maquina intel·ligent necessita molt de temps per aprendre i això li dificulta aquesta capacitat de la multitasca.

La intel·ligència humana aprèn de les seves experiències prèvies (educació, experiència laboral o una situació que va aportar un aprenentatge) i té la capacitat de considerar múltiples factors per obtenir un resultat. Per exemple, una persona va llegir una notícia al diari sobre diferents tipus de negocis i gràcies a això va arribar a pensar en tots els possibles negocis que podria tenir. Tanmateix, el nivell d’aprenentatge d’una IA està orientat a uns objectius en concret.

La intel·ligència de la qual disposem els humans consisteix a prendre la decisió correcta en el moment indicat, sense opcions predefinides, en canvi la intel·ligència artificial de les màquines escull la decisió correcta d’entre totes les que disposa.

La velocitat de processament de les intel·ligències artificials és molt més eficient que la dels humans. Per exemple, un metge pot fer un diagnòstic en deu minuts mentre que un sistema d'IA en podria fer un milió.

2.2. Quines aplicacions tenen les intel·ligències artificials?

Les intel·ligències artificials tenen múltiples funcions en diferents camps segons les nostres necessitats. Les més destacables són al camp de la medicina, el comerç, la investigació i l'oci. Una que volem destacar és la detecció del càncer.

Si els humans haguéssim d'anar mirant cèl·lula per cèl·lula buscant algun tipus de càncer no acabaríem mai. És per això que es va crear una IA capaç de detectar el càncer, que prèviament ha d'estar programada i ha d'anar aprenent segons vagi adquirint informació. Una màquina a primera instància no pot saber si una cèl·lula és cancerígena, ja que no ho interpreta per ella mateixa, és a dir, no pensa (prèviament s'ha de programar).

Una altra utilitat important que trobem de les IA és, per exemple, l'algorisme de YouTube, una immensa xarxa neuronal que té milions i milions de dades d'informació guardada. Aquest el que fa és recomanar-te vídeos relacionats amb el que normalment es mira. No obstant això, que una IA determini el contingut del que es visualitza és molt difícil, ja que la temàtica de cada vídeo pot ser molt diferent i tenir coses comunes molt difícils de relacionar.

Per últim, però no menys important, volem destacar la importància dels mòduls d'anàlisi dels escacs, un altre exemple d'intel·ligència artificial. Anteriorment es feia molt difícil per les persones trobar les millors jugades a l'hora de jugar els escacs. És per això que es van crear els mòduls d'anàlisi, que troben les millors jugades tenint en compte totes les possibles jugades següents. Això ha estat molt útil, ja que ha fet avançar a moltes persones en el seu nivell de joc.

3. Aprenentatge automàtic

L'aprenentatge automàtic més conegut com a "machine learning" és una branca de la intel·ligència artificial que crea programes simulant l'aprenentatge d'un ésser humà. S'entén com aprenentatge la millora o tècniques que s'obtenen mitjançant l'experiència i l'ús de dades (identificant patrons o tendències i elaborant prediccions), que no estaven inicialment. Aquest aprenentatge es dona quan l'agent intel·ligent observa les dades que nosaltres li proporcionem i construeix un model, el qual ens podrà resoldre nous problemes futurs.

Els algorismes, que s'utilitzen per desenvolupar el machine learning, obtenen els seus propis càlculs segons les dades que es van recollint al sistema. Com més s'obtinguin més precises seran les accions resultants. És per això que totes les dades que se subministren es converteixen en nous algorismes els quals poden dissenyar noves respostes.

El machine learning té una infinitat d'aplicacions i situacions on es pot aplicar. Les principals se situen a la medicina (genètica, diagnòstics mèdics, detecció de malalties...) i la robòtica (conducció automàtica, robots d'ajuda personal, reconeixement de veu...).

Dins de l'àmbit del machine learning hi ha quatre tipus d'algorismes: els d'aprenentatge supervisat, els d'aprenentatge no supervisat, els d'aprenentatge per reforç i els d'aprenentatge semi supervisats. Nosaltres ens centrarem en els tres primers, ja que els algorismes semi supervisats no són tan coneguts i s'utilitzen menys.

3.1. Aprenentatge supervisat

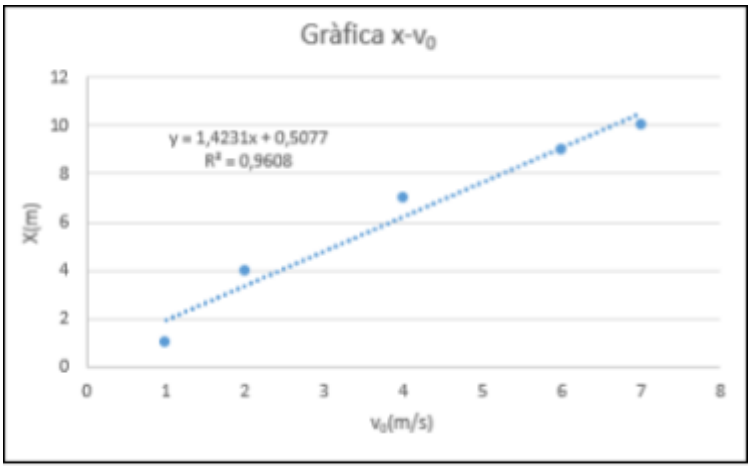
L'aprenentatge supervisat és una tècnica dins del machine learning en la qual s'utilitzen dades etiquetades per trobar una funció determinada. Aquestes dades solen distribuir-se en parelles on una és el valor d'entrada i l'altre el valor de sortida. El que farà aquesta funció és relacionar ambdues parelles (tenint en compte totes les dades) per poder trobar resposta a dades d'entrada noves on no se sàpiga el resultat. El que retorni la funció dependrà del problema que estiguem resolent. Si el problema és de regressió, la funció retornarà un número, en canvi, si es tracta de classificació, retornarà una etiqueta.

Els problemes de regressió són aquells en els quals tens dades d'entrada i dades de sortida (solen ser punts) i els has de relacionar amb la funció que vols trobar. Un exemple de regressió seria trobar la funció que relaciona graus Fahrenheit i Celsius sense saber la fórmula prèviament; només amb les parelles d'entrada i sortida.

Els problemes de classificació són aquells en els quals tens dades d'entrada (que solen ser punts) i dades de sortida (que solen ser etiquetes d'aquells punts) i el que fa la funció és trobar quina és la relació entre els punts i les etiquetes. Així quan rebi un nou punt, podrà determinar a quina etiqueta pertany. Un exemple de classificació seria que a partir d'unes fotografies de gossos i gats (etiquetades), l'algorisme sigui capaç de trobar la relació que hi ha entre ells (a partir de les característiques dels animals) i donar la resposta correcta a partir d'una nova fotografia (d'un gos o d'un gat).

3.1.1. Regressió lineal:

No us heu preguntat mai com l'Excel, a partir dels punts donats, (vegeu R.001) és capaç de generar una recta aproximada? Això es pot aconseguir utilitzant l'algorisme de regressió lineal que explicarem a continuació. Cal dir que aquest algorisme ha estat jutjat sobre si es pot incloure o no dins l'àmbit de la intel·ligència artificial a causa de la seva simplicitat, però nosaltres considerem que pot ser una bona introducció a aquest tema.



R.001

La regressió lineal és un algorisme d'aprenentatge supervisat que s'utilitza principalment en l'àmbit de l'estadística. L'objectiu d'aquest és determinar la funció lineal (d'una o més variables) que millor s'adapti al conjunt de punts proporcionats d'entrada. Per fer això s'utilitzen funcions de cost que determinen l'error de la recta als punts, com l'R2, l'error quadràtic mig o l'algorisme del gradient descendent, objectiu de les quals és minimitzar aquest error.

La forma que tenen aquestes funcions que volem obtenir és la següent: $y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 +\ldots$ i per aconseguir trobar els paràmetres 'w$_i$' es creen diverses equacions lineals per cadascuna de les dades proporcionades, quedant així:

$$y_1 = w_0 + w_1x_{11} + w_2x_{12} + w_3x_{13} +\ldots$$
$$y_2 = w_0 + w_1x_{21} + w_2x_{22} + w_3x_{23} +\ldots$$
$$y_3 = w_0 + w_1x_{31} + w_2x_{32} + w_3x_{33} +\ldots$$
$$y_4 = w_0 + w_1x_{41} + w_2x_{42} + w_3x_{43} +\ldots$$
$$\ldots$$

Utilitzar tants sistemes d'equacions no resulta eficient i és difícil de treballar, per això s'utilitzen matrius quedant aquesta equació més senzilla (R.002):

$$Y = XW^T \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} \qquad X = \begin{pmatrix} 1 & x_{11} & x_{12} & x_{13} \\ 1 & x_{21} & x_{22} & x_{23} \\ 1 & x_{31} & x_{32} & x_{33} \\ 1 & x_{41} & x_{42} & x_{43} \end{pmatrix} \qquad W = \begin{pmatrix} w_0 & w_1 & w_2 & w_3 \end{pmatrix}$$

R.002

Recordem que en un producte de matrius el nombre columnes de la primera matriu ha de coincidir amb el nombre de files de la segona, per tant la matriu 'W' s'ha de transposar (canviar files per columnes). També cal destacar que la primera columna de la matriu 'X' conté 1's per poder obtenir el terme independent.

Com que a la regressió lineal hi ha errors, aquests els calcularem amb la funció de l'error quadràtic mig, que fa la mitjana dels errors elevats al quadrat, sent la següent (en el pla): $ECM = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2$ ($\widehat{y}_i$: valor real; y_i: valor estimat), que l'extrapolarem a una vectorial: $ECM = (Y - XW^T)^T(Y - XW^T)$. Per fer el quadrat d'una matriu, una s'ha de transposar i és per això que en l'equació vectorial, una de les dues matrius ho està.

Com que nosaltres volem fer que l'error sigui el més petit possible, derivarem la funció vectorial i la igualarem a 0 (per trobar el mínim). Finalment aïllarem el vector 'W' que conté els paràmetres invariables necessaris per completar la nostra funció lineal, quedant-nos el següent (mínim error quadràtic mig): $W = (X^TX)^{-1}X^TY$. Després, només cal substituir les nostres dades (X i Y) per obtenir aquest vector i resoldre el nostre problema de regressió.

3.1.2. Petita implementació:

Com sempre farem en aquest treball, explicarem la part teòrica d'un tema i seguidament implementarem aquests conceptes a codi Python a partir d'un exemple molt simple ([1] vegeu l'annex 1).

En l'exemple de la figura R.001 mostrat anteriorment, es poden veure un conjunt de punts i una equació d'una recta que ha generat l'Excel a partir d'aquests:

x	1	2	4	6	7
y	1	4	7	9	10

y = 1,4231x + 0,5077

El que nosaltres volem fer és trobar aquesta equació a partir del que hem explicat anteriorment de l'error quadràtic mig. Per fer-ho importarem dos mòduls que són "NumPy", que ens ajuda a operar amb matrius més eficientment i "matplotlib.pyplot", que ens permet dibuixar punts, rectes i gràfics.

Seguidament crearem les nostres matrius 'x' i 'y', recordant que la matriu 'x' ha de contenir la primera columna d'1's, tal com es veu a la imatge R.003.

```
x = np.array([1,2,4,6,7])
x = np.array([np.ones(len(x)),x]).T
print('x =',x)

y = np.array([1,4,7,9,10])
print('\ny =',y)
```

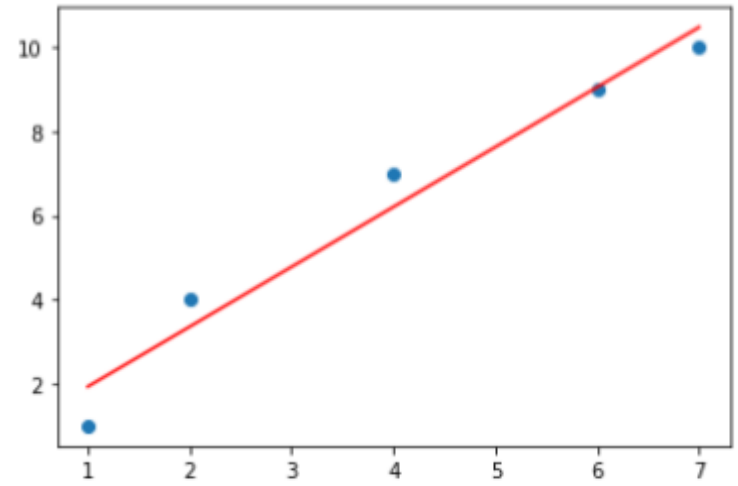
```
x = [[1. 1.]
 [1. 2.]
 [1. 4.]
 [1. 6.]
 [1. 7.]]

y = [ 1  4  7  9 10]
```

R.003

Ara només cal aplicar la fórmula del mínim error quadràtic mig per obtenir el vector 'W', que a la primera posició tindrà el terme independent de la nostra recta i a la segona el pendent d'aquesta (imatge R.004), i finalment mostrar-ho tot per pantalla (punts i recta, imatge R.005):

```
W = np.linalg.inv(x.T @ x) @ x.T @ y

print(W)
plt.scatter(x.T[1],y)

plt.plot([1,7],[W[1] * 1 + W[0], W[1] * 7 + W[0]], c = 'red')
plt.show()

print("Funció de la recta: y =",round(W[1],5),"\bx +",round(W[0],5))
```

R.004



R.005

Com es pot observar a la imatge R.005, la recta que nosaltres hem obtingut ($y = 1,42308x + 0,50769$) és molt semblant a la que l'Excel ens ha proporcionat ($y = 1,4231x + 0,5077$); per tant, hem assolit el nostre objectiu.

3.2. Aprenentatge no supervisat

L'aprenentatge no supervisat és una tècnica dins del machine learning en la qual no es disposa de dades etiquetades, sinó només de dades d'entrada les quals seran relacionades per l'algorisme buscant patrons o relacions entre elles. A causa d'això no se sap quina és la resposta esperada i es fa difícil determinar l'eficiència dels algorismes, al contrari que passava amb els algorismes d'aprenentatge supervisat.

Hi ha dos tipus d'algorismes dins l'aprenentatge no supervisat: el clustering o agrupament i l'associació. Els algorismes de clustering tenen com a objectiu classificar en grups les dades proporcionades d'entrada (normalment són punts) segons la seva similitud. Aquesta similitud sol ser la distància entre aquests punts.

Un possible exemple seria classificar els usuaris d'una xarxa social depenent dels seus interessos per proporcionar un anunci o un altre.

Els algorismes d'associació intenten buscar certs patrons que hi pot haver en les dades d'entrada. Un exemple d'això seria que l'algorisme s'adonés que les persones que compren un ordinador solen comprar també un ratolí i llavors a aquestes persones que s'han comprat un ordinador recentment els hi surti un anunci d'un ratolí.

3.2.1 K-Mitjanes

Un dels algorismes més coneguts de clustering és l'anomenat K-Mitjanes o 'K-Means', el qual destaca per la seva rapidesa.

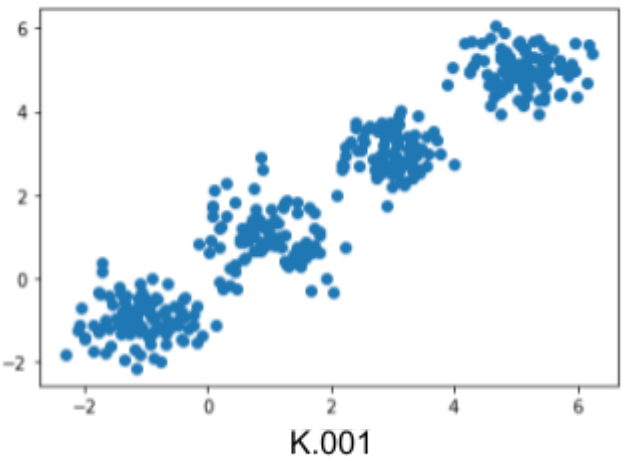
L'algorisme K-Mitjanes és un algorisme de classificació que es basa a assignar *n* punts d'entrada en *K* grups o "clústers" possibles (*K* donat com a input). Per fer-ho s'utilitzen centroides, que són punts que definiran al grup en conjunt (la mitjana dels punts) un cop aquests s'actualitzin. Això no obstant, la seva posició inicial sol ser aleatòria o generada a partir del conjunt.

Després de generar els centroides, aquests s'han d'actualitzar de la següent manera. Primer els punts s'assignen al centroide més proper. Seguidament, per determinar la nova posició dels centroides, es fa la mitjana dels seus punts assignats i es repeteix el procés fins a arribar a un cert nombre d'iteracions o que la variabilitat de la posició del centroide sigui molt petita.

En acabar l'actualització dels centroides, podem saber els punts que conformen cada grup depenent del centroide més proper que tinguin.

3.2.2. Petita implementació:

En aquesta implementació, el nostre objectiu serà agrupar els següents núvols de punts generats aleatòriament (imatge K.001) ([2] vegeu l'annex 2):



K.001

Per realitzar aquesta pràctica necessitarem la llibreria *NumPy*, que ens serveix per operar millor amb vectors i matrius i la llibreria *matplotlib.pyplot* que ens permet dibuixar els gràfics (imatge K.002).

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
```

K.002

Per generar els núvols de punts sense utilitzar cap *data set* o base de dades ho farem de la següent manera (K.003):

```
x1 = np.random.standard_normal((100,2))*0.6+np.ones((100,2))
x2 = np.random.standard_normal((100,2))*0.5-np.ones((100,2))
x3 = np.random.standard_normal((100,2))*0.4-2*np.ones((100,2))+5
x4 = np.random.standard_normal((100,2))*0.5+6*np.ones((100,2))-1

```

K.003

El que cal entendre d'aquest codi és que cada "x" és un núvol de punts que té 100 elements (punts) i cadascun és generat aleatòriament. Tanmateix, el que volem és que el programa ens sàpiga trobar aquests grups, per tant, ajuntarem tots aquests punts en un sol lloc que anomenarem "data" (imatge K.004) ("plt.scatter" serveix per dibuixar la gràfica K.001).

```
data = np.concatenate((x1,x2,x3,x4),axis=0)

plt.scatter(data[:,0],data[:,1])
plt.show()
```

K.004

Tenint ja les nostres dades completes, ara programarem l'algorisme K-mitjanes. Per començar crearem una classe anomenada "K_MEANS" que ens permetrà tenir totes les funcions que creem i utilitzem de l'algorisme ben ordenades, a la que passarem les dades dels punts, el valor "K", la variabilitat mínima de la posició dels centroides i el nombre màxim d'iteracions. Seguidament crearem les posicions dels centroides que seguiran el següent criteri: centroide 0: (0,0), centroide 1: (1,1), centroide 2: (2,2) i centroide 3: (3,3), vegeu la figura (K.005). Ho fem així, ja que d'aquesta manera els centroides estan ben repartits al llarg del pla i és menys probable que es puguin produir errors a causa de la seva proximitat.

```
class K_MEANS:
    def __init__(self, data, K, opt=0.0001, iter=500):
        self.K = K
        self.opt = opt
        self.iter = iter
        self.data = data
        self.centroids = {i : [i,i] for i in range(self.K)}
```

K.005

Abans de crear la funció d'entrenament crearem les dues funcions principals que formaran part d'aquesta. La primera, anomenada "clasify" (imatge K.006), assigna a cada punt el centroide més proper i retorna aquesta classificació.

```python
def clasify(self):
    clasifications = {i : [] for i in range(self.K)}

    for point in self.data:
        distances = [np.linalg.norm(point - centroid) for centroid in self.centroids.values()]
        clasifications[np.argmin(distances)].append(point)

    return clasifications
```

K.006

La segona, anomenada "modify_centroids" (imatge K.007), agafa com a paràmetre aquesta classificació feta per la funció anterior i fa la mitjana dels punts assignats a cada centroide, modificant la seva posició. També crea una variable anomenada "opt", que conté la modificació que es duu a cada centroide, retornant aquest valor.

```python
def modify_centroids(self, clasifications):
    opt = 0

    for index in range(self.K):
        prev_cent = self.centroids[index]
        curr_cent = np.array(np.average(clasifications[index], axis=0))
        self.centroids[index] =  curr_cent
        opt += np.linalg.norm(curr_cent - prev_cent)

    return opt
```

K.007

Ara sí, crearem la funció d'entrenament, que bàsicament consta d'un bucle que utilitza les dues funcions anteriors i comprova si la variabilitat de la posició dels centroides és prou petita per deixar d'actualitzar-los i sortir del bucle. Finalment retorna la posició dels centroides i la classificació d'aquests amb els punts (K.008).

```python
def fit(self):
    for _ in range(self.iter):

        clasifications = self.clasify()
        opt = self.modify_centroids(clasifications)

        if opt < self.opt:
            break

    return clasifications, self.centroids
```

K.008

Per últim, crearem la funció “draw” (imatge K.009) fora de la classe “K_MEANS”, ja que no té res a veure amb l’algorisme, que ens permetrà dibuixar tots els centroides i els punts de diferents colors, depenent de la classificació passada com a paràmetre (cada clúster sera d’un color diferent).

```
40 def draw(clasifications, centroids):
41     colors = ["g", "r", "c", "b", "k"]
42 
43     for cent, points in enumerate(clasifications.values()):
44         points = np.array(points)
45         plt.scatter(points[:,0], points[:,1], marker='x', color = colors[cent % len(colors)])
46 
47         plt.scatter(centroids[cent][0],centroids[cent][1], marker='o', color='k')
48     plt.show()
49 
```

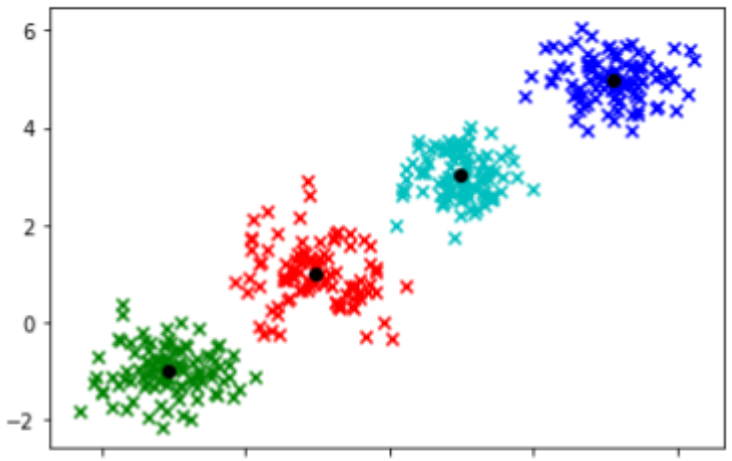
K.009

Per acabar, ens queda executar tot el que hem fet anteriorment. Per fer-ho utilitzarem la classe “K_MEANS” i executarem la funció “fit”. Seguidament, dibuixarem la classificació dels punts i els centroides amb la funció “draw” (imatge K.010).

```
50 def main():
51     k_means = K_MEANS(data, 4)
52     clas, cent = k_means.fit()
53 
54     draw(clas, cent)
55 
56 if __name__ == '__main__':
57     main()
```

K.010

Aquest és el resultat (vegeu la imatge K.011):



K.011

Com es pot observar, l'algorisme ha sigut capaç de classificar els següents punts en quatre clústers (marcats per colors). Els punts negres són els centroides de cada clúster.

Els algorismes d'agrupament tenen moltes utilitats. Algunes són en el màrqueting (per identificar les persones amb hàbits de compres similars i proporcionar els anuncis adequats) o a la medicina (per detectar diferents malalties), entre d'altres.

3.3. Aprenentatge per reforç

L'aprenentatge per reforç és una àrea del machine learning en la qual s'utilitza un agent que és sotmès a un sistema de recompenses i penalitzacions tenint en compte les seves accions en un entorn determinat. Aquestes accions modifiquen l'estat en el qual es troba, per tant, no es tracta d'obtenir la recompensa més gran a curt termini, sinó a llarg termini.

En aquest tipus d'algorismes no s'espera que l'agent realitzi les millors accions la primera vegada sinó que aquest aprengui a partir d'assaig i error amb una taula de recompenses, mitjançant experiències prèvies que li permetin optimitzar les següents respostes. És per això que l'agent ha de ser entrenat prèviament o durant l'execució del programa.

Per realitzar això s'utilitzen diversos algorismes un dels quals és el "Q-learning".

3.3.1 Q-Learning

Aquest algorisme busca una seqüència d'accions òptima maximitzant la recompensa total de cada acció. És un dels algorismes més emprats, ja que no només estima el valor de la recompensa a curt termini, sinó que també obté possibles valors futurs.

Això ho fa amb una taula d'accions-estats (matriu "q"), que determina la recompensa depenent de l'acció que es faci en cada estat i amb l'equació de Bellman, que actualitza aquesta taula: $\widehat{Q}(s,a) = Q(s,a) + \alpha[R + \lambda \cdot maxQ(s',a') - Q(s,a)]$ ($Q(s,a)$: valor actual de recompensa; α: ràtio d'aprenentatge; R: recompensa immediata; λ: factor de descompte (entre 1 i 0); $maxQ(s',a')$: màxima recompensa futura).

D'aquesta fórmula, les coses més importants són la ràtio d'aprenentatge que regula la rapidesa d'aprenentatge (si és molt gran, els canvis a $\widehat{Q}(s,a)$també seran grans i es farà difícil per l'agent aprendre, mentre que si és massa petit, l'agent aprendrà molt lent) i el factor de descompte que fa que es tinguin en compte les recompenses futures, però en menor mesura que les recompenses immediates.

3.3.2. Petita implementació:

Imagineu aquest escenari (figura Q.001) ([3] vegeu l'annex 3):

Q.001

Nosaltres ens situem a la casella 0 i volem anar a la 11. Tot això ho volem fer anant pel camí més curt i sense cremar-nos amb el foc. Així a simple vista sabem que el camí correcte seria 0-7-6-5-2-3-4-11, però això el nostre agent no ho sap, sinó que ho ha d'anar descobrint a mesura que vagi aprenent.

Per poder realitzar això, importarem tres llibreries, les quals ens ajudaran a fer el programa més eficient i fàcil de programar, les quals són *NumPy*, *pandas* i *random (Q.002)*.

```
[1]   1 import numpy as np
      2 import pandas as pd
      3 import random
```

Q.002

No ens estendrem massa a explicar el que fan, però el que cal dir és que *NumPy* ens ajuda amb les matrius, *pandas* ens facilita la visualització d'aquestes i *random* ens genera nombres pseudoaleatoris amb els quals podem treballar.

Seguidament generarem una matriu de recompenses (Q.003), que serà el nostre entorn, la qual funcionarà de la següent manera:

```
1 entorn = np.array([
2     [ 0,-10,-1,-1,-1,-1,-1, 0,-1,-1,-1,-1],
3     [ 0,-10, 0,-1,-1,-1, 0,-1,-1,-1,-1,-1],
4     [-1,-10, 0, 0,-1, 0,-1,-1,-1,-1,-1,-1],
5     [-1, -1, 0, 0, 0,-1,-1,-1,-1,-1,-1,-1],
6     [-1,-1,-1, 0, 0,-1,-1,-1,-1,-1,-1,10],
7     [-1,-1, 0,-1,-1, 0, 0,-1,-1,-1, 0, -1],
8     [-1,-10,-1,-1,-1, 0, 0, 0,-1, 0,-1,-1],
9     [ 0, -1,-1,-1,-1,-1, 0, 0,-1,-1,-1,-1],
10    [-1,-1,-1,-1,-1,-1,-1,-1,-10, 0,-1,-1],
11    [-1,-1,-1,-1,-1,-1, 0,-1,-10, 0, 0,-1],
12    [-1,-1,-1,-1,-1, 0,-1,-1, -1, 0, 0,-1],
13    [-1,-1,-1,-1, 0,-1,-1,-1,-1,-1,-1,10]
14 ])
15 pd.DataFrame(entorn)
```

Q.003

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	-10	-1	-1	-1	-1	-1	0	-1	-1	-1	-1
1	0	-10	0	-1	-1	-1	0	-1	-1	-1	-1	-1
2	-1	-10	0	0	-1	0	-1	-1	-1	-1	-1	-1
3	-1	-1	0	0	0	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	10
5	-1	-1	0	-1	-1	0	0	-1	-1	-1	0	-1
6	-1	-10	-1	-1	-1	0	0	0	-1	0	-1	-1
7	0	-1	-1	-1	-1	-1	0	0	-1	-1	-1	-1
8	-1	-1	-1	-1	-1	-1	-1	-1	-10	0	-1	-1
9	-1	-1	-1	-1	-1	-1	0	-1	-10	0	0	-1
10	-1	-1	-1	-1	-1	0	-1	-1	-1	0	0	-1
11	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	10

Les files d'aquesta matriu representen l'habitació on el nostre agent es pot trobar i les columnes les habitacions on anar. Els valors que donem a la matriu van relacionats amb la validesa dels moviments (-1 si el moviment és invàlid i 0 si és vàlid) i amb l'arribada de les habitacions especials (-10 si s'arriba al foc i 10 si s'arriba al final).

No obstant això, cal dir que aquesta manera no és l'única que existeix a l'hora de crear la matriu de recompenses, però creiem que és la més adequada i fàcil d'entendre per l'exemple que volem resoldre.

A continuació cal crear la matriu "q" que s'anirà actualitzant a mesura que s'executi el programa (Q.004) i funcionarà de la mateixa manera que la de recompenses exceptuant els valors que aquesta contindrà, començant per 0's.

```
[3]  1 matriu_q = np.zeros((12,12))
     2
     3 pd.DataFrame(matriu_q)
```

Q.004

	0	1	2	3	4	5	6	7	8	9	10	11
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Ara crearem totes les funcions que ens ajudaran a poder entrenar l'agent i que aquest aprengui: actualitzant la matriu "q":

En primer lloc, crearem una funció que ens generarà nombres pseudoaleatoris (Q.005) que determinaran l'habitació inicial en cada episodi d'entrenament, ja que el nostre agent aprendrà començant en habitacions aleatòries.

```
1 def habitacio_inicial(nre_hab = 11):
2     return random.randint(0,11)
3
```

Q.005

Seguidament, crearem una altra funció que ens dirà a quines habitacions podem anar depenent de l'habitació a la qual estiguem i escollirà una a l'atzar (imatge Q.006).

```
4 def obtenir_accio_valida(habitacio_actual):
5     accions = []
6     for habitacio, accio_valida in enumerate(entorn[habitacio_actual]):
7         if accio_valida != -1:
8             accions.append(habitacio)
9
10    return random.choice(accions)
```

Q.006

A continuació, crearem una funció que guardi i retorni l'acció que hem rebut de la funció anterior (habitació nova on l'agent es troba) i actualitzi la matriu "q" passant com a paràmetres l'alfa i la gamma (Q.007), tal com hem vist a la part teòrica de l'explicació ($\hat{Q}(s,a) = Q(s,a) + \alpha[R + \lambda \cdot maxQ(s',a') - Q(s,a)]$).

```
def fer_accio(habitacio_actual, alpha, gamma):
    accio = obtenir_accio_valida(habitacio_actual)

    rec_immediata = entorn[habitacio_actual, accio]
    rec_futura = max(matriu_q[accio])
    valor_actual = matriu_q[habitacio_actual,accio]

    nou_valor_matriu_q = valor_actual + alpha * (rec_immediata + gamma * rec_futura - valor_actual)

    matriu_q[habitacio_actual,accio] = nou_valor_matriu_q

    return accio
```

Q.007

Les últimes dues funcions que cal fer per entrenar finalment l'agent són les següents (Q.008):

```
def episodi_de_entrenament(habitacio_actual, alpha, gamma):
    while True:
        habitacio_actual = fer_accio(habitacio_actual, alpha, gamma)
        if habitacio_actual == 11:
            break

def entrenar_al_agent(iteracions, alpha, gamma):
    print("Entrenant l'agent...")
    for _ in range(iteracions):
        habitacio_actual = habitacio_inicial()
        episodi_de_entrenament(habitacio_actual, alpha, gamma)
    print("Entrenament completat!!")
```

Q.008

La primera funció (Q.008) controla un episodi d'entrenament, que bàsicament consisteix en un bucle infinit on l'agent anirà fent accions i no pararà fins que arribi a l'habitació final (11).

La segona és la funció (Q.008) que entrena l'agent, la qual consta d'un nombre d'iteracions que són les vegades que es faran els episodis d'entrenament, començant l'agent d'una habitació aleatòria (Q.005).

El que ara ens queda per fer és executar la funció d'entrenar l'agent, donant els valors d'alfa, gamma i el nombre d'iteracions que volem que l'agent s'entreni (Q.009) i observar quines modificacions rep la matriu "q" (Q.010):

```
alpha = 0.5
gamma = 0.8

entrenar_al_agent(1000, alpha, gamma)

pd.DataFrame(matriu_q)
```

Q.009

	0	1	2	3	4	5	6	7	8	9	10	11
0	10.483983	10.476528	0.000000	0.000000	0.000000	0.000000	0.000000	13.104978	0.000000	0.000000	0.000000	0.000000
1	10.483983	10.476528	25.595661	0.000000	0.000000	0.000000	16.381223	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	10.476528	25.595661	31.994576	0.000000	20.476528	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	25.595661	31.994576	39.993220	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	31.994576	39.993220	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	49.991525
5	0.000000	0.000000	25.595661	0.000000	0.000000	20.476528	16.381223	0.000000	0.000000	0.000000	16.381223	0.000000
6	0.000000	10.476528	0.000000	0.000000	0.000000	20.476528	16.381223	13.104978	0.000000	13.104978	0.000000	0.000000
7	10.483983	0.000000	0.000000	0.000000	0.000000	0.000000	16.381223	13.104978	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.483983	13.104978	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	16.381223	0.000000	0.483983	13.104978	16.381223	0.000000
10	0.000000	0.000000	0.000000	0.000000	0.000000	20.476528	0.000000	0.000000	0.000000	13.104978	16.381223	0.000000
11	0.000000	0.000000	0.000000	0.000000	39.991494	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	49.989406

Q.010

Com es pot veure a la imatge Q.010, la matriu “q” ha modificat els seus valors, per tant, ara podem saber amb certesa per quines habitacions hem de passar escollint el valor més gran de l’habitació on ens trobem (fila).

Per exemple, si ens situem a l’habitació 4, el moviment correcte seria anar a l’habitació 11 i com es pot veure a la matriu “q” (Q.011), anar a l’habitació 11 és l’acció que obté més benefici, en comptes d’anar a la 3 o quedar-se a la 4 (vegeu Q.001):

	0	1	2	3	4	5	6	7	8	9	10	11
0	10.483983	10.476528	0.000000	0.000000	0.000000	0.000000	0.000000	13.104978	0.000000	0.000000	0.000000	0.000000
1	10.483983	10.476528	25.595661	0.000000	0.000000	0.000000	16.381223	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.000000	10.476528	25.595661	31.994576	0.000000	20.476528	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
3	0.000000	0.000000	25.595661	31.994576	39.993220	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.000000	0.000000	31.994576	39.993220	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	49.991525
5	0.000000	0.000000	25.595661	0.000000	0.000000	20.476528	16.381223	0.000000	0.000000	0.000000	16.381223	0.000000
6	0.000000	10.476528	0.000000	0.000000	0.000000	20.476528	16.381223	13.104978	0.000000	13.104978	0.000000	0.000000
7	10.483983	0.000000	0.000000	0.000000	0.000000	0.000000	16.381223	13.104978	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.483983	13.104978	0.000000	0.000000
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	16.381223	0.000000	0.483983	13.104978	16.381223	0.000000
10	0.000000	0.000000	0.000000	0.000000	0.000000	20.476528	0.000000	0.000000	0.000000	13.104978	16.381223	0.000000
11	0.000000	0.000000	0.000000	0.000000	39.991494	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	49.989406

Q.011

Per finalitzar aquesta petita pràctica crearem una funció que executarà l’agent i utilitzarà la matriu “q” a l’hora d’escollir els seus moviments (els valors més grans d’aquesta, depenent de l’habitació on es trobi). Vegem els resultats (imatges Q.012 i Q.013).

```
1 habitacio_actual = 0
2 print("Recorregut: ")
3 executar_el_agent(habitacio_actual)
```

```
Recorregut: 0-7-6-5-2-3-4-11
```

Q.012

```python
def executar_el_agent(habitacio_actual):
    print(f"\b{habitacio_actual}-")
    while habitacio_actual != 11:
        accio = np.argmax(matriu_q[habitacio_actual,:])
        print(f"\b{accio}-")
        habitacio_actual = accio

    print("\b\b")
```

Q.013

El nostre agent ha sigut capaç d'arribar al final des de l'habitació 0, tal com volíem (0-7-6-5-2-3-4-11): sense cremar-se i pel camí més curt. Malgrat això, no només és capaç d'arribar al final des de l'habitació 0, sinó des de qualsevol habitació (imatges Q.014, Q.015).

```python
habitacio_actual = 10
print("Recorregut: ")
executar_el_agent(habitacio_actual)
```

```
Recorregut: 10-5-2-3-4-11
```

Q.014

```python
habitacio_actual = 1
print("Recorregut: ")
executar_el_agent(habitacio_actual)
```

```
Recorregut: 1-2-3-4-11
```

Q.015

3.3.3. Més enllà…

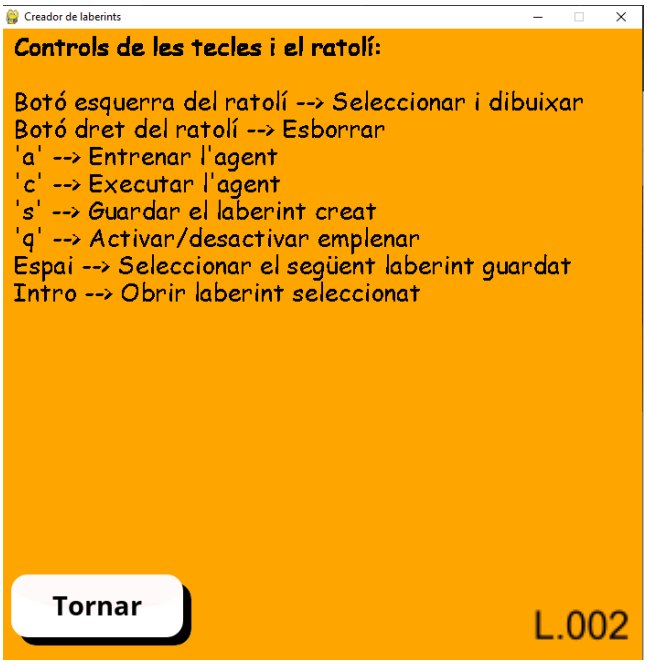
Com que aquest algorisme ens semblava tan interessant i innovador hem volgut arribar un nivell més enllà. És per això que hem creat un programa que resol qualsevol laberint que l'usuari dibuixi en pantalla, programant així un editor de laberints ([4] vegeu l'annex 4).

En aquest cas no explicarem tot el codi que hem programat, ja que es faria massa llarg i ja hem explicat l'algorisme a la implementació anterior, sinó més com s'utilitza l'aplicació i quines són les funcions que pots utilitzar.

El programa és molt intuïtiu. Consta d'una pantalla quadriculada de color blau i un menú de color taronja. En el menú se situen els elements que es poden dibuixar (en ordre: bloc, final, camí, inici) i un botó que es pot clicar per saber els controls (L.001). També hi ha text al costat del botó que explicarem seguidament què indica.

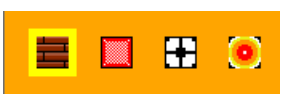
L.001

Quan es clica el botó de *controls* la pantalla esdevé en la següent (figura L.002):

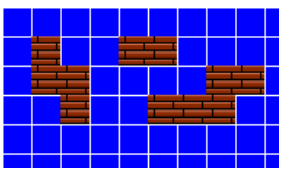


L.002

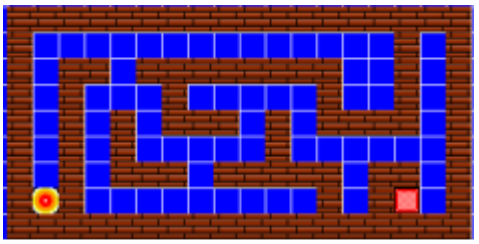
Aquest menú explica tot el que pots fer amb el ratolí i les tecles. Amb el botó esquerre del ratolí pots seleccionar l'element que vols dibuixar (L.003). En el moment que el cliques apareix envoltat de color groc, per saber que està seleccionat:



L.003



Després de seleccionar un element pots dibuixar-lo a la pantalla blava (clicant o mantenint polsat). Per esborrar el que has fet només cal prémer el botó dret del ratolí i passar el cursor per on has dibuixat (L.004).

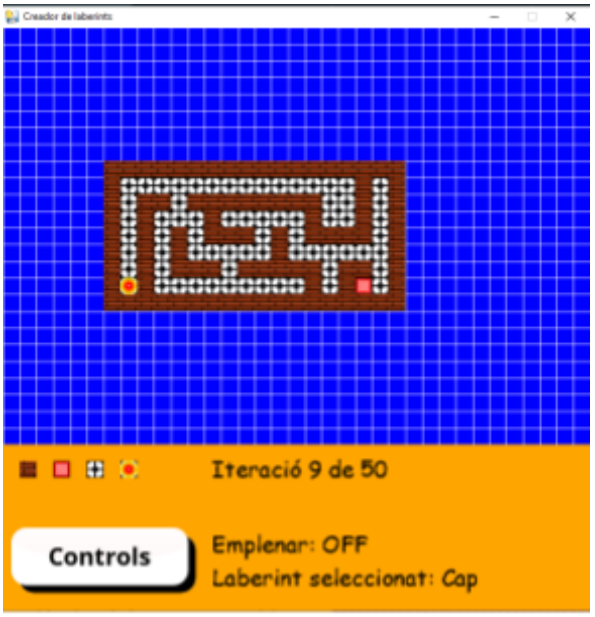


L.004

En el moment que tens dibuixat un laberint com aquest (imatge L.004) has d'omplir l'espai blau amb l'element blanc que és per on pot passar el nostre personatge i com que a vegades pot ser una mica llarg de fer hem implementat una funció que el que fa és omplir tot aquest espai amb l'element que seleccionis (semblant a l'emplenament del "paint").

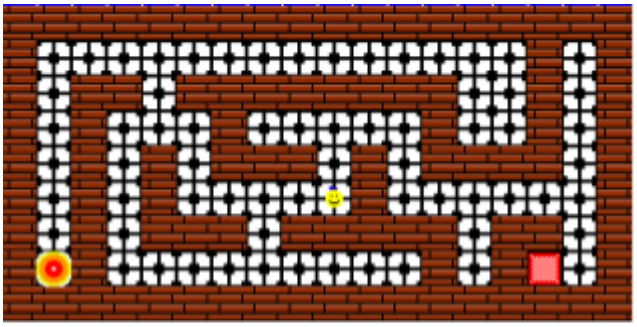
Per activar-ho i desactivar-ho has de clicar la lletra 'q' i per saber si està o no seleccionada aquesta opció has de mirar al menú on posa "Emplenar: …".

Després de dibuixar el laberint has d'entrenar a la intel·ligència i per fer-ho has de clicar la lletra 'a'. Quan ho facis, et sortiran les iteracions completades tal com es veu a la imatge L.005.



L.005

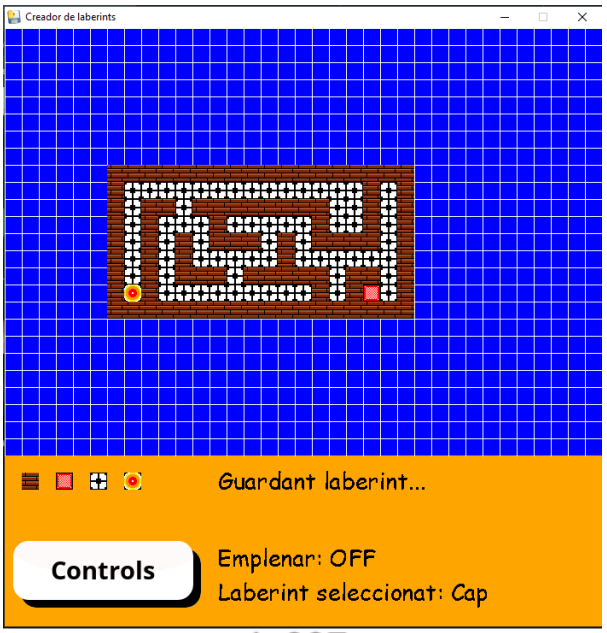
Quan ja hagin acabat les iteracions podràs executar l'agent intel·ligent polsant la lletra 'c' i veuràs com una cara somrient groga es va movent (figura L.006). Aquest objecte és el nostre personatge resolent el laberint:



L.006

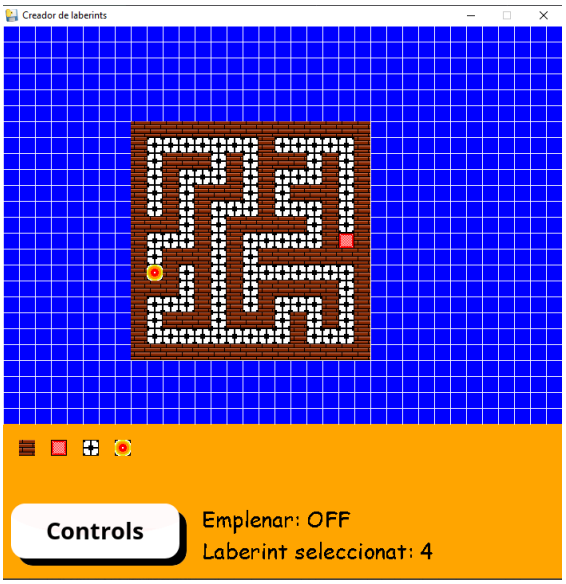
Una altra funció que hem creat és la de guardar els laberints i poder-los recuperar una vegada hagis tancat l'aplicació. Per guardar els laberints has de clicar la lletra 's'. Seguidament et sortirà un missatge de què s'ha guardat correctament (imatge L.007).

Si vols recuperar un laberint guardat anteriorment el primer que has de fer és seleccionar-ne un clicant a l'espai. En el moment que ho facis es veurà en el menú "Laberint seleccionat: 1". Sempre que cliquis a l'espai aquest número anirà augmentant (sempre que hi hagi sufients laberints guardats) fins que arribarà un punt en què tornarà a posar "Laberint seleccionat: Cap" (quan arribi a l'últim laberint guardat).



L.007

Quan ja tinguis el laberint escollit has de clicar a l'intro i veuràs com la pantalla canvia al nou laberint que has escollit (L.008).



L.008

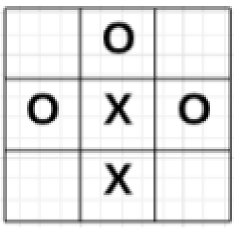
4. Hi ha més…

A banda del machine learning, també s'utilitzen altres algorismes els quals són considerats intel·ligència artificial, però no formen part directament del machine learning. Aquest és el cas de l'algorisme "minimax" que explicarem a continuació.

4.1 Minimax

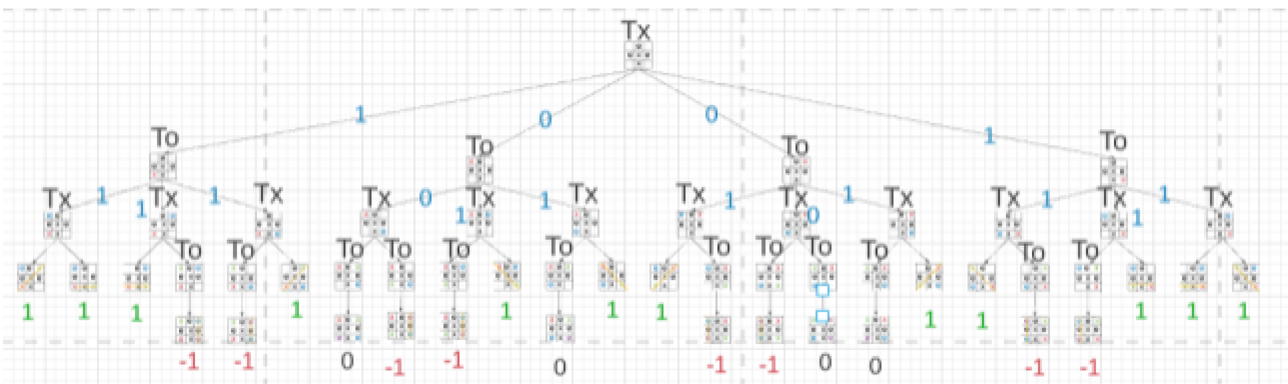
El minimax és un algorisme d'intel·ligència artificial recursiu emprat en la presa de decisions d'un joc basat en reduir la teva pèrdua de victòria en una certa posició de joc. No forma part dels algorismes de machine learning, ja que en aquest cas l'algorisme no aprèn (a diferència dels altres algorismes explicats anteriorment). L'objectiu és escollir la jugada més bona entre totes les disponibles tenint en compte el que pot fer el nostre rival les següents jugades. Aquest algorisme, no obstant, no es pot aplicar a tots els jocs, sinó que ha de constar de quatre bases (depenent del joc, es poden modificar segons convingui): que sigui un joc per torns, de dos jugadors, que ambdós jugadors puguin saber tot el que passa i el que pot passar al joc i de suma 0, és a dir, com millor sigui la teva puntuació, pitjor serà la de l'oponent i viceversa.

Minimax considera que cada posició de la partida té un valor el qual s'acosta més cap al més o menys infinit depenent del jugador al qual afavoreix la posició. El jugador anomenat *min* vol fer que la seva puntuació s'acosti cap al menys infinit i el jugador *max* vol que s'acosti cap al més infinit.



M.001

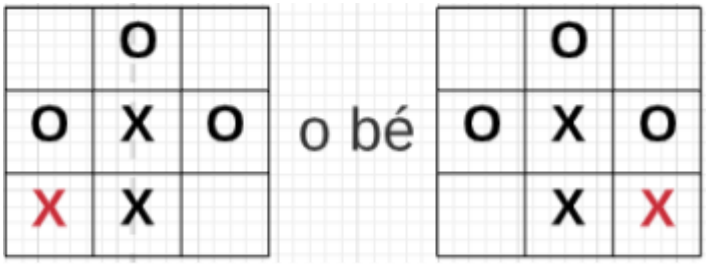
Per exemple donada la posició segons la figura M.001 del joc del tres en ratlla, l'objectiu de la IA és trobar la millor jugada del jugador 'X'. Per fer-ho avaluarà totes les posicions possibles donant l'arbre mostrat a la figura M.002, que al final de totes les avaluacions possibles retorna un 1 si guanya el jugador Tx ('X'), un -1 si guanya el To ('O') o un 0 si queden taules (marcats en vermell, verd i negre):



M.002

El jugador Tx és el jugador *max*, per tant a l'hora d'escollir una jugada, escollirà la que tingui el nombre més **gran** possible (recordem que guanya la partida amb el nombre 1). A la inversa passa amb el jugador To, que és el jugador *min* i triarà el valor més **petit** (el valor agafat per cadascun dels jugadors és de color blau).

Veiem que finalment el jugador Tx pot escollir entre dues possibles jugades (el primer camí i l'últim) per poder guanyar. Aquestes serien les dues possibles posicions (vegeu a la imatge M.003):



M.003

Aquest mètode podria no semblar una IA, però sí que ho és, ja que troba la millor resposta (*output*) amb diferents posicions (*inputs*) amb el mateix algorisme.

Cal destacar que la IA no sap jugar al tres en ratlla, sinó que avalua posicions futures depenent de l'avantatge que hi hagi (que la decideixen els programadors segons es guanyi o es perdi en el joc, per exemple). Aquestes avaluacions fan que la IA decideixi el que fer el següent torn.

4.2. Petita implementació

Aquest cop la implementació la farem a partir de l'exemple anterior (tres en ratlla) ([5] vegeu l'annex 5).

Després d'importar el mòdul "pygame" (que és el que ens ajudarà a dibuixar el taulell) i aplicar la lògica del joc ens trobem que li toca jugar a l'ordinador i volem que faci la millor jugada possible.

La lògica que aplicarem a l'hora de trobar la millor jugada és que la funció 'Minimax' ens retornarà un índex el qual utilitzarem per accedir al nostre taulell (una llista de 9 posicions) i donar-li el valor 'O' que en aquest cas és el signe de la IA. Però a causa que aquest algorisme és recursiu i ha de retornar l'anàlisi de les posicions futures, també retornarà l'avaluació de la partida (però no la utilitzarem quan accedim al taulell).

Aquesta funció rep tres paràmetres: la posició actual de la partida, la profunditat de l'avaluació (ja que hi ha jocs en què les possibles avaluacions creixen molt ràpidament) i un valor booleà que determina si el jugador que juga ara és el *max* o el *min* ('X' o 'O').

El primer que fem és determinar si la profunditat és 0 o si a la posició actual algú ha guanyat amb una funció definida anteriorment com a 'Guanya (posició)' que rep com a paràmetre la posició que volem avaluar i retorna un 1 si guanyen les 'X', un -1, si ho fan les 'O' o 0 si queden taules o no guanya ningú. Si passa el condicional retornem l'avaluació de la partida i *None* que és el valor que rebrà l'índex (perquè encara no estem avaluant cap posició).

Seguidament ens preguntem si li toca el torn al jugador *max* ('X') i en cas afirmatiu crearem una llista de nom 'Millor' que contingui de primer valor l'avaluació de la posició (que la inicialitzem a -2, perquè aquesta manera forcem que canviï de valor, ja que el jugador *max* el pitjor que pot tenir és el valor -1 i això ho fem per poder tenir un índex i que no ens doni error) i de segon l'índex, que l'inicialitzarem amb un valor buit. A continuació, amb un bucle *for* avaluem les possibles posicions (tornant a utilitzar la funció Minimax, però canviant els paràmetres: passant la posició nova, restant 1 a la profunditat i canviant la variable 'JugadorX', que determina si és el jugador *max*, a *false*), recollint l'avaluació i l'índex corresponents a la variable 'Possible'. Comparem si l'avaluació actual és més gran que l'anterior i si és així canviem la variable 'Millor' a l'avaluació nova i l'índex nou. Després d'avaluar totes les posicions retornem la variable 'Millor' amb l'avaluació i l'índex que millor s'adapten a la posició proporcionada inicialment.

Amb el jugador *min* passa el mateix, però en comptes d'inicialitzar el primer valor de la variable 'Millor' a -2 ho farem a 2 (per les mateixes raons explicades anteriorment: forcem a fer que aquest valor canviï; en aquest cas, el pitjor valor de *min* és l'1). Un altre canvi és que passem el paràmetre 'JugadorX' a *True* a la nova crida de la funció 'Minimax' en comptes de *False* i que en comptes de canviar la variable 'Millor' quan la nova avaluació sigui més gran ho farem quan aquesta sigui més petita.

Mostrem petits fragments del codi (M.004, M.005):

```
index = Minimax(TAULELL, TAULELL.count(""), False)[1]
if index != None:
    TAULELL [index] = 'O'
```

M.004

```python
def Minimax(posicio, profunditat, JugadorX):

    if Guanya(posicio) != 0 or profunditat == 0:
        return Guanya(posicio), None

    if JugadorX:
        Millor = [-2, None]

        for i in range(len(posicio)):
            if posicio[i] == '':
                posicio[i] = 'X'
                Possible = Minimax(posicio, profunditat - 1, False)

                if Possible[0] > Millor[0]:
                    Millor[0] = Possible[0]
                    Millor[1] = i

                posicio[i] = ''

        return Millor

    if not JugadorX:
        Millor = [2, None]

        for i in range(len(posicio)):
            if posicio[i] == '':
                posicio[i] = 'O'
                Possible = Minimax(posicio, profunditat -1, True)

                if Possible[0] < Millor[0]:
                    Millor[0] = Possible[0]
                    Millor[1] = i

                posicio[i] = ''

        return Millor
```

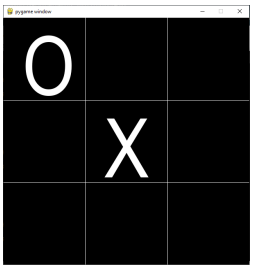
M.005

Com a curiositat, ara veurem quantes vegades s'executa la funció 'Minimax' cada posició. Hem creat una variable de nom 'aux', que s'incrementa cada vegada que entra a la funció i al final imprimeix per pantalla el seu valor tornant-se a igualar a 0 (imatges M.006).

```python
def Minimax(posicio, profunditat, JugadorX):

    global aux
    aux += 1
```

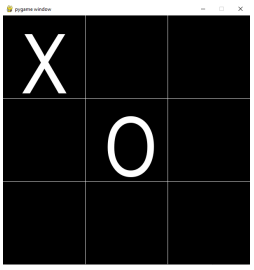
```python
index = Minimax(TAULELL, TAULELL.count(""), False)[1]
if index != None:
    TAULELL [index] = 'O'

print(aux)
aux = 0
```

M.006



55505

M.007



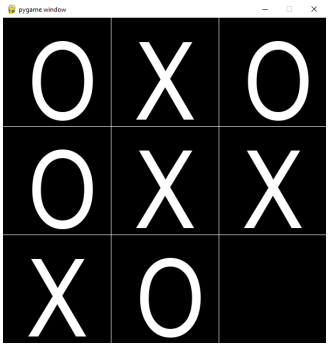
59705

M.008

63905

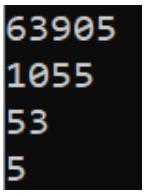
M.009

Aquestes són les avaluacions de les possibles posicions inicials (M.007, M.008, M.009). Les altres posicions són les mateixes però girades 90º, 180º, etc. Es pot observar que n'hi ha més posicions possibles a la imatge M.009 (a causa que n'hi ha menys jugades guanyadores: recordem que quan algun jugador guanya, s'atura l'avaluació de les jugades posteriors, encara que hi hagi més espais buits).



M.010

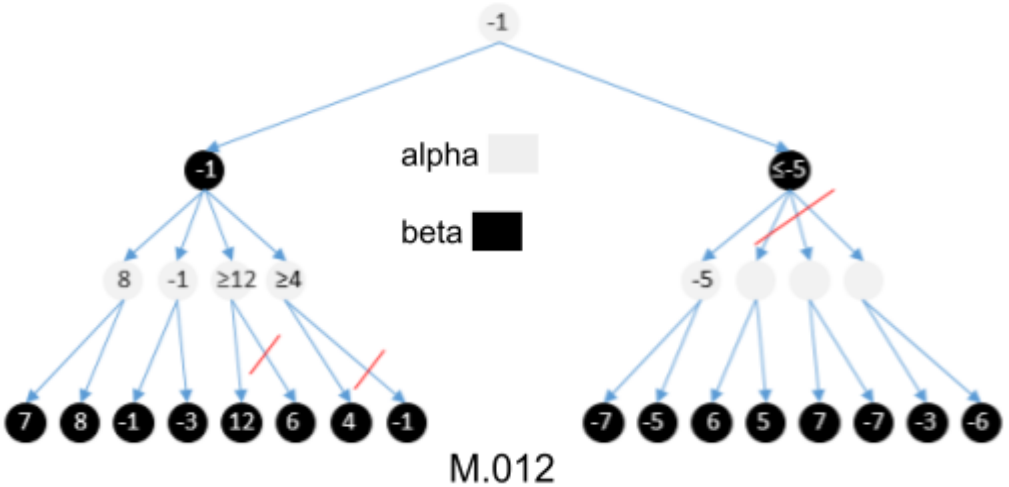
En aquesta partida (vegeu M.010) (començada com la imatge M.009) podem observar que el nombre d'avaluacions va disminuint, ja que cada vegada queden menys posicions. Si sumem totes les iteracions, ens dona la gran quantitat de 65018 iteracions (M.011). Això en jocs senzills com aquest no preocupa, però en jocs més complexos, sí que és un gran problema, ja que l'algorisme Minimax és d'ordre r^p, sent r les ramificacions i p la profunditat assolida i és per això que es redueix la profunditat fins a un cert punt (en aquest cas no hem limitat la profunditat, sinó que l'hem donada el valor de tots els possibles espais en blanc que podien ser avaluats) o s'utilitzen altres mètodes que redueixen aquestes iteracions.



M.011

Un d'aquests mètodes és la **poda *alpha beta*** que en el millor dels casos pot reduir aquest ordre fins $r^{\frac{p}{2}}$. Aquesta poda consisteix en no avaluar certes ramificacions (nodes), perquè sabem que no les tindrem en compte a l'hora de decidir la nostra millor jugada; no ens resulten útils. Les podes realitzades al node *max* s'anomenen alpha i les realitzades al *min*, beta.

Donat el següent arbre de minimax (M.012) podem observar que hi ha posicions que no hem avaluat; les "podem" (del verb podar).



M.012

Això ho fem per optimitzar molt més el nostre programa i fer-lo més eficient. Per poder saber quan s'han de fer aquestes podes a l'hora de programar es creen dues variables: α i β. Alpha està inicialitzada amb -∞ i beta amb +∞. El que fan és anar actualitzant el seu valor, alpha en nodes *max* (sempre que el nou valor sigui més gran que l'actual) i beta en nodes *min* (quan el nou valor sigui més petit), i anar passant el seu valor a través de nodes més profunds. En el moment que $\alpha \geq \beta$sabem que no podrem millorar el valor obtingut, per tant, no cal seguir avaluant més en aquell node.

Aquest mètode de reducció d'iteracions aplicat a codi quedaria així (M.013, M.014):

```python
def Minimax(posicio, profunditat, alpha, beta, JugadorX):

    global aux
    aux += 1

    if Guanya(posicio) != 0 or profunditat == 0:
        return Guanya(posicio), None

    if JugadorX:
        Millor = [-2, None]

        for i in range(len(posicio)):
            if posicio[i] == '':
                posicio[i] = 'X'
                Possible = Minimax(posicio, profunditat - 1, alpha, beta, False)

                if Possible[0] > Millor[0]:
                    Millor[0] = Possible[0]
                    Millor[1] = i

                posicio[i] = ''

                alpha = max(alpha, Possible[0])
                if alpha >= beta:
                    break

        return Millor
```

M.013

```python
    if not JugadorX:
        Millor = [2, None]

        for i in range(len(posicio)):
            if posicio[i] == '':
                posicio[i] = 'O'
                Possible = Minimax(posicio, profunditat -1, alpha, beta, True)

                if Possible[0] < Millor[0]:
                    Millor[0] = Possible[0]
                    Millor[1] = i

                posicio[i] = ''

                beta = min(beta, Possible[0])
                if alpha >= beta:
                    break

        return Millor
```

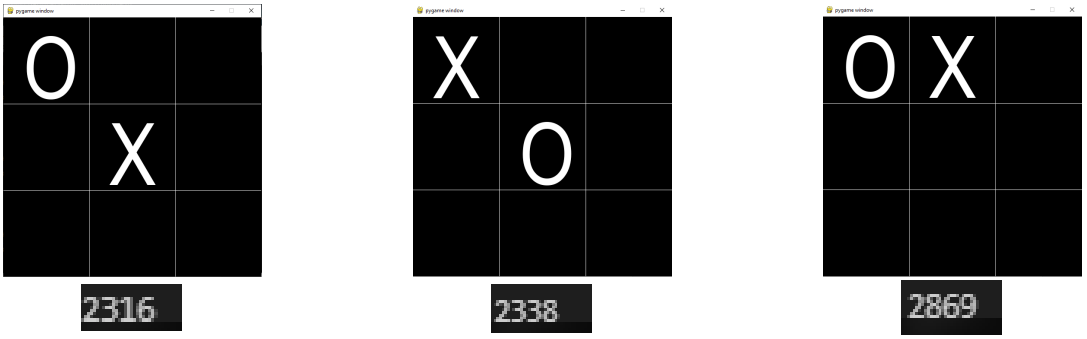
M.014

En comptes de passar $+\infty$ i $-\infty$, passem -2 i 2, ja que sabem que les avaluacions no arribaran mai a -2 o 2.

```
index = Minimax(TAULELL, TAULELL.count(""), -2, 2, False)[1]
if index != None:
    TAULELL [index] = 'O'

print(aux)
aux = 0
```

M.015

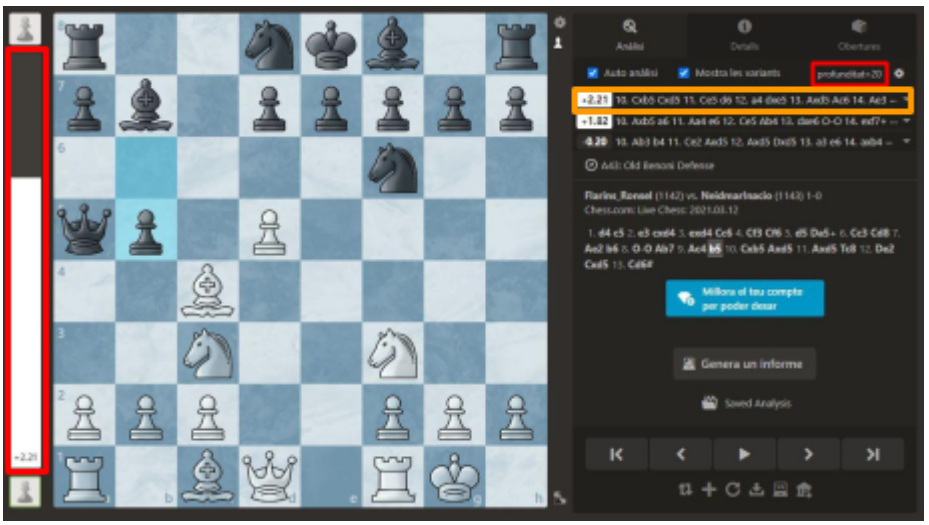
La part de codi que hem afegit actualitza *alpha* i *beta* depenent si l'avaluació és més gran o més petita (explicat anteriorment) i en el cas que $\alpha \geq \beta$ passarà a la següent iteració (cosa per la qual hem utilitzat el *break*). Realment és eficient? Ho comprovem (M.016):



M.016

El que anteriorment eren 55.505, 59.705 i 63.905 iteracions inicials han esdevingut a 2.316, 2.338 i 2.869 iteracions respectivament. Com es pot notar hi ha hagut un gran canvi gràcies a la poda *alpha-beta*, per tant, realment sí que és eficient pel nostre programa.

Un bon exemple de tot el que hem vist anteriorment seria la figura M.017:



M.017

Es pot observar a l'esquerra l'avaluació de la partida i a la dreta els moviments més òptims i la profunditat de la recerca d'aquests, fruit del gran nombre de jugades possibles.

5. Conclusions

L'objectiu principal d'aquest treball era comprovar el que deia la teoria d'alguns dels principals algorismes del machine learning i de la intel·ligència artificial amb la pràctica. Com bé hem pogut observar no ens han calgut altres llibreries d'intel·ligència artificial per tal realitzar o completar els exercicis pràctics, sinó que tot ho hem realitzat des de zero i tal com s'havia explicat a la part teòrica. A causa d'això hem après python, un llenguatge de programació molt ampli i amb multitud d'usos que ens ha servit per completar tot aquest procés d'aprenentatge.

Aquest treball ha permès acostar-nos en el món de la intel·ligència artificial, però en ser un tema tan ampli es fa impossible poder dominar-ho tot. És per això que cal destacar que hi ha molts més tipus d'algorismes d'intel·ligència artificial i de machine learning que no hem pogut incloure en el treball a causa de la seva dificultat o la seva extensió.

Un dels principals problemes que ens hem trobat a l'hora de fer el treball era la recerca d'informació. Cada pàgina web i vídeo explicava el mateix algorisme d'una manera diferent, cosa que creava confusió i dubtes. No obstant això, el que vam fer va ser visitar moltes pàgines web i vídeos fiables i de confiança de persones qualificades en aquest tema. D'aquesta manera ens garantia una visió global de l'algorisme per poder redactar-lo d'una manera clara i concisa. A més, havíem de descartar aquella informació en què s'utilitzaven llibreries externes d'intel·ligència artificial per explicar els algorismes, ja que el que nosaltres volíem era recrear l'algorisme des de zero; sense cap peça de codi preprogramada.

L'organització del treball s'ha ajustat al nostre horari escolar i de vacances (més treball a les vacances i menys en època d'exàmens i d'estudi); assolint els objectius que teníem marcats en cada període. El que ens passava era que no teníem un camí marcat, a causa que no coneixíem l'existència de molts dels algorismes i sovint havíem de fer recerca de diferents algorismes que ens semblessin interessants i reprogramar el temps dedicat al treball.

A banda de les dificultats que ens hem anat trobant en l'elaboració del nostre treball, pensem que el tema que hem escollit ens ha causat força interès i hem gaudit a l'hora de realitzar-lo.

Esperem que aquest treball us hagi servit per a la vostra recerca d'informació o per treure-us certs neguits que tinguéssiu sobre la intel·ligència artificial i el machine learning.

6. Bibliografia

Algorismes d'agrupació:

https://datascience.eu/es/aprendizaje-automatico/algoritmos-de-agrupacion-y-su-importancia-en-el-aprendizaje-de-las-maquinas/ (6/8/21)

Algorismes no supervisats:

https://es.wikipedia.org/wiki/Aprendizaje_no_supervisado (14/9/21)

https://aprendeia.com/aprendizaje-no-supervisado-machine-learning/ (14/9/21)

Algorismes supervisats:

https://ca.wikipedia.org/wiki/Aprenentatge_supervisat (14/9/21)

Algorismes supervisats VS algorismes no supervisats:

https://www.youtube.com/watch?v=WyDGryVC8Iw&list=PLJjOveEiVE4CpEtQjRBVe4FgmqhYq-EKa (6/8/21)

https://blog.bismart.com/ca/difer%C3%A8ncies-machine-learning-supervisat-i-no-supervisat (14/9/21)

Aplicacions:

https://es.wikipedia.org/wiki/Aplicaciones_de_la_inteligencia_artificial (26/09/21)

Aprenentatge per reforç:

https://es.wikipedia.org/wiki/Aprendizaje_por_refuerzo (20/7/21)

https://github.com/ShangtongZhang/reinforcement-learning-an-introduction (20/7/21)

https://www.xataka.com/inteligencia-artificial/conceptos-inteligencia-artificial-que-aprendizaje-refuerzo (20/7/21)

https://ichi.pro/es/aprendizaje-por-refuerzo-para-principiantes-q-learning-y-sarsa-173871259878059 (20/7/21)

https://www.youtube.com/watch?v=PJl4iabBEz0 (20/7/21)

https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/ (20/7/21)

https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/ (20/7/21)

https://ccc.inaoep.mx/~emorales/Cursos/Busqueda/refuerzo.pdf (20/7/21)

https://bootcampai.medium.com/reinforcement-learning-aprendizaje-por-refuerzo-c34bb085bb5 (20/7/21)

http://www.cs.us.es/~fsancho/?e=109 (20/7/21)

https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/ (20/7/21)
https://cayetanoguerra.github.io/ia/rl/Aprendizaje_por_refuerzo_apuntes.pdf (20/7/21)
https://programmerclick.com/article/2837602293/ (20/7/21)
https://ccc.inaoep.mx/~emorales/Cursos/Aprendizaje2/Acetatos/refuerzo.pdf (20/7/21)
https://addi.ehu.es/bitstream/handle/10810/29123/Asier%20Aguayo%20Velasco%20-%20TFG.pdf?sequence=4 (20/7/21)
https://www.youtube.com/watch?v=wVXXLLT6srY (1/8/21)
https://www.youtube.com/watch?v=EnCoYY087Fc (4/8/21)

K-mitjanes o k-means:

https://datascience.eu/es/aprendizaje-automatico/k-means-clustering-en-machine-learning/ (6/8/21)
https://www.youtube.com/watch?v=_xbtHUyTHXA (29/8/21)
https://es.wikipedia.org/wiki/K-medias (29/8/21)
https://datascience.eu/es/aprendizaje-automatico/k-means-clustering-en-machine-learning/ (29/8/21)
https://ai538393399.wordpress.com/2020/09/29/k-means-clustering-algorithm-without-libraries/ (29/8/21)
https://medium.com/machine-learning-algorithms-from-scratch/k-means-clustering-from-scratch-in-python-1675d38eee42 (29/8/21)
https://www.kaggle.com/andyxie/k-means-clustering-implementation-in-python (30/8/21)
https://www.machinelearningplus.com/predictive-modeling/k-means-clustering/ (30/8/21)
https://medium.com/@rishit.dagli/build-k-means-from-scratch-in-python-e46bf68aa875 (30/8/21)
https://www.askpython.com/python/examples/k-means-clustering-from-scratch (30/8/21)
https://www.youtube.com/watch?v=HRoeYblYhkg&list=PLQVvvaa0QuDfKTOs3Keq_kaG2P55YRn5v&index=38 (30/8/21)
https://www.unioviedo.es/compnum/laboratorios_py/kmeans/kmeans.html (3/9/21)

Machine learning:

https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico (9/6/21)
https://www.bancofinandina.com/finanblog/noticias/2020/01/03/que-es-machine-learning-y-como-funciona (9/6/21)
https://www.apd.es/que-es-machine-learning/ (9/6/21)
https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico (10/6/21)

https://www.asimovinstitute.org/neural-network-zoo/ (25/06/21)
https://www.youtube.com/watch?v=8h94lDosMMc (25/06/21)
https://www.youtube.com/watch?v=4c7oFu36d6k (25/06/21)
http://www.joanybelortiz.com/aplicaciones-machine-learning-ejemplos/ (20/7/21)
https://www.grapheverywhere.com/algoritmos-de-machine-learning/ (20/7/21)
https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/ (20/7/21)
https://e-archivo.uc3m.es/bitstream/handle/10016/30007/TFG_Jesus_Lopez_Baeza-Rojano_2019.pdf?sequence=1&isAllowed=y (20/7/21)
https://www.asimovinstitute.org/neural-network-zoo/ (20/7/21)
https://www.aprendemachinelearning.com/principales-algoritmos-usados-en-machine-learning/ (1/8/21)

Minimax:

https://ca.wikipedia.org/wiki/Minimax (20/04/21)
https://www.youtube.com/watch?v=trKjYdBASyQ (21/04/21)
https://www.youtube.com/watch?v=l-hh51ncgDI (21/04/21)
https://www.youtube.com/watch?v=QJjM7EKDRuc (22/04/21)
https://www.youtube.com/watch?v=I0y-TGehf-4 (25/04/21)

Regressió lineal:

https://es.wikipedia.org/wiki/Regresi%C3%B3n_lineal (24/7/21)
https://www.aprendemachinelearning.com/regresion-lineal-en-espanol-con-python/#more-5722 (24/7/21)
http://eio.usc.es/eipc1/BASE/BASEMASTER/FORMULARIOS-PHP-DPTO/MATERIALES/Mat_50140116_Regr_%20simple_2011_12.pdf (24/7/21)
https://www.youtube.com/watch?v=qm_dfifxRE4 (24/7/21)
https://www.iartificial.net/regresion-lineal-con-ejemplos-en-python/ (26/7/21)
https://www.youtube.com/watch?v=k964_uNn3l0 (26/7/21)
https://www.iartificial.net/error-cuadratico-medio-para-regresion/ (26/7/21)

7. Annexos

En aquests annexos el lector disposa del codi de programació explicat anteriorment perquè aquest pugui manipular-lo i estudiar-lo més en profunditat:

[1] Annex 1: Regressió lineal (petita implementació):
https://colab.research.google.com/drive/1HN3rCnkg3QtGlZu-EUb167Tbo6FrRgl8?usp=sharing

[2] Annex 2: K-means o K-mitjanes (petita implementació):
https://colab.research.google.com/drive/1Eb0OnMvku2q9OrPW9XJKOw_MnYIEnGhK?usp=sharing

[3] Annex 3: Q-learning (arribar al final sense cremar-se):
https://colab.research.google.com/drive/1gy85f5jkuAEi6FIfO-VjO9BxYeX6yUh0?usp=sharing

[4] Annex 4: Q-learning (creador de laberints):
https://drive.google.com/file/d/1lEuQVw_Zn9IIlpT9YUrYMoLAv9ZNHzXZ/view?usp=sharing

[5] Annex 5: Minimax (tres en ratlla):
https://drive.google.com/file/d/1k8wCCAdmvbuz8KkCDF0aCDo_c-PLzDl6/view?usp=sharing

En aquest annex mostrem l'esquema inicial que hem fet servir per començar el nostre treball:

[6] Annex 6: Esquema Lucidchart (esquema inicial del treball):
https://drive.google.com/file/d/134IgvgUcXvysqSJbjCQ0AP68bd4EgvV5/view?usp=sharing