Introducció a l'aprenentatge computacional

PID_00267991

Andrés Cencerrado Barraqué
Carles Ventura Royo

Temps mínim de dedicació recomanat: 3 hores

Universitat Oberta de Catalunya

Andrés Cencerrado Barraqué **Carles Ventura Royo**

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Carles Ventura Royo (2019)

Primera edició: setembre 2019

Av. Tibidabo, 39-43, 08035 Barcelona
Realització editorial: FUOC

Índex

Introducció

Els mètodes que hem estudiat fins ara han estat orientats a resoldre els problemes computacionalment aplicant les estratègies pròpies d'un **ens intel·ligent**, abstraient la informació adequada per a representar un problema i protocol·litzar maneres de trobar solucions, explotant el coneixement existent de què ja disposem d'un determinat sistema o, fins i tot, lidiant amb la incertesa intrínseca dels problemes del món real. No obstant això, aquests mètodes, tal com fins ara els hem estudiat, deixen de banda una de les característiques definitòries de la intel·ligència: **l'aprenentatge**. És a dir, la capacitat d'adaptar-se automàticament tenint en compte les situacions experimentades.

En aquest mòdul farem un pas més enllà per a presentar, de manera introductòria, un dels aspectes més decisius per a l'èxit de la Intel·ligència Artificial com a àrea de coneixement: **l'aprenentatge computacional** (o *Machine Learning*).

Són diverses les definicions per a saber en què consisteix l'aprenentatge computacional, però totes aquestes coincideixen en un aspecte clau: la importància de l'experiència passada i com aquesta modifica el model intern que té un sistema intel·ligent d'un aspecte o problema de la realitat.

> L'aprenentatge computacional té com a objectiu establir una sèrie de mètodes que milloren el rendiment del sistema a partir de la seva pròpia experiència.

Aquesta subdisciplina s'estudia a un nivell més profund en les assignatures *Aprenentatge computacional* i *Intel·ligència artificial avançada*. En aquest mòdul, farem un repàs més superficial i introductori dels seus principis, conceptes generals i mètodes existents més rellevants.

El primer apartat està destinat a exposar la classificació del tipus d'aprenentatge computacional més freqüentment utilitzada: aprenentatge supervisat, aprenentatge no supervisat i aprenentatge per reforç.

Seguidament, s'aprofundeix en l'aprenentatge supervisat per mitjà de l'estudi de mètodes tan estesos com el kNN, els arbres de decisió, les *Support Vector Machines* (SVM) i les xarxes neurals.

Es presenta una visió de l'aprenentatge no supervisat a partir dels fonaments d'un tipus d'algorismes molt representatius d'aquesta tipologia: els algorismes d'agrupament (*k-means* i *fuzzy c-means*).

L'estudi dels algorismes d'aprenentatge per reforç es duu a terme mitjançant l'anàlisi d'un dels algorismes més reconeguts: Q-learning. S'inclouen en aquesta categoria els algorismes genètics, els quals també se solen utilitzar en problemes d'optimització.

Finalment, s'exposen els fonaments teòrics de la partició de dades i els protocols de validació en l'aprenentatge computacional.

Objectius

Aquest mòdul didàctic vol assolir els objectius següents:

1. Obtenir una visió general del que són els sistemes d'aprenentatge computacional.

2. Conèixer els diferents tipus d'aprenentatge, en funció de la supervisió.

3. Estudiar els fonaments teòrics dels mètodes representatius de l'aprenentatge computacional.

4. Adquirir una base de coneixement de com es tracten els conjunts de dades a fi de validar el rendiment dels sistemes respecte de l'aprenentatge realitzat.

1. Taxonomia de les tècniques d'aprenentatge computacional

Un dels aspectes més importants a l'hora de considerar els mètodes d'aprenentatge computacional és la supervisió de la bondat dels seus resultats. És a dir, quin tipus d'informació es disposa del resultat que ha de donar un sistema.

Com veurem al llarg d'aquest mòdul, unes vegades disposarem de criteris ben definits i mostres reals que ens permetran avaluar el comportament dels nostres algorismes, altres vegades tindrem aproximacions de caire heurístic, i en d'altres no tindrem informació sobre què esperem de la sortida del sistema, sinó que més aviat esperem que el sistema n'extregui les característiques principals de les entrades per a ajudar-nos a entendre aspectes com ara la seva estructura o les interrelacions entre els seus elements.

Tenint en compte això, la classificació més àmpliament acceptada de les tècniques d'aprenentatge computacional és la següent: mètodes amb aprenentatge supervisat, aprenentatge per reforç i aprenentatge no supervisat. A continuació, repassem els fonaments de cadascun d'aquests tipus.

1.1. Aprenentatge supervisat

L'aprenentatge supervisat correspon a la situació en què hi ha un coneixement complet de quina és la resposta que s'ha de donar en una determinada situació.

Així, en l'aprenentatge supervisat l'objectiu és aconseguir un sistema basat en el coneixement que reprodueixi la sortida quan estem en una situació que correspongui a l'entrada.

En molts casos, el problema es pot descriure formalment com un problema d'optimització o com l'aproximació d'una funció a partir d'uns exemples (parells entrada/sortida). També es parla sovint d'aprenentatge a partir d'exemples i el conjunt d'exemples s'anomena *conjunt d'entrenament*.

1.2. Aprenentatge per reforç

En aquest cas, el coneixement de la qualitat del sistema només és parcial. No es disposa del valor que correspon a la sortida per a un determinat conjunt de valors d'entrada sinó, solament, una gratificació o una penalització segons el resultat que ha donat el sistema.

Per exemple, si un robot planifica una trajectòria i col·lideix amb un objecte rebrà una penalització, però en cap moment no hi ha un expert que li ofereixi una trajectòria alternativa correcta. De la mateixa manera, si la trajectòria planificada pel robot no provoca cap col·lisió, el robot rep una gratificació. Les planificacions que el robot faci a partir d'aquest moment tindran en compte les penalitzacions/gratificacions rebudes.

1.3. Aprenentatge no supervisat

En el cas de l'aprenentatge no supervisat no hi ha supervisió de casos. Només es disposa d'informació de les entrades i no de les sortides. L'aprenentatge ha d'extraure un coneixement útil a partir de la informació disponible.

En aquest tipus d'aprenentatge hi ha els algorismes de **categorització**, que permeten ordenar la informació disponible, i d'**agrupament**, que estructuren les dades d'entrada en diversos grups d'acord amb les característiques de cada component, generalment representades en un espai *n*-dimensional.

Seguidament, exposarem alguns dels algorismes més coneguts i utilitzats de cadascun d'aquests tipus.

2. Mètodes d'aprenentatge supervisat

Per a formalitzar aquest tipus d'aprenentatge, considerem un conjunt d'entrenament C format per N exemples, on cada exemple és un parell (x, y) en què x és un vector de dimensió M i y és el resultat d'aplicar una funció f (que no coneixem) al vector x. Per tant, tenim N exemples en un espai de dimensió M. És a dir, $X = \{x_1, ..., x_i, ..., x_N\}$. En cas que hi hagi un error en les mesures, tenim que y és $f(x)$ més un cert error ϵ. Això és, $y = f(x) + \varepsilon$. A partir d'aquesta informació es vol construir un model que denotem per M_C (utilitzem el subíndex perquè el model depèn del conjunt d'exemples C). Així, l'objectiu és aconseguir que el model M_C aplicat a un element x doni semblant a $f(x)$. És a dir, $M_C(x)$ és una aproximació de $f(x)$.

L'aprenentatge supervisat és una tècnica emprada en multitud de tipologies de problemes, d'entre les quals destaquen els problemes de regressió, de classificació o, fins i tot, els problemes de cerca com els que hem estudiat anteriorment.

En els propers subapartats ens recolzarem en una mostra de dades de dos conjunts (*datasets*) que podem trobar al repositori UCI (Frank i Asunción, 2010): el problema «iris» i el problema «mushroom», que contenen dades de característiques de les flors en funció de les mides del sèpal i pètals, i de característiques dels bolets en funció del seu barret i tronc, respectivament.

2.1. k Nearest Neighbors (kNN)

L'**algorisme kNN** (*k* veïns més propers) és un dels exemples clàssics de l'algorisme d'aprenentatge supervisat. S'utilitza principalment per a dur a terme classificacions i aquestes es fan a partir de mesures de similitud o distància. Així, es comparen nous exemples amb conjunts (un per classe) de prototips, assignant la classe del prototip més proper o buscant en una base d'exemples quin és el més proper.

Suposem que volem fer una aplicació per a un magatzem de flors, on arriben diàriament milers de productes. Disposem d'un sistema làser que ens subministra una sèrie de mesures de les flors i ens demanen que el sistema les classifiqui automàticament per a transportar-les mitjançant un robot a les diferents prestatgeries del magatzem. Les mesures que envia el sistema làser són la longitud i amplada del sèpal i el pètal de cada flor. La taula 1 mostra exemples d'aquest tipus de dades.

Taula 1. Conjunt d'entrenament

class	*sepal-length*	*sepal-width*	*petal-length*	*petal-width*
setosa	5,1	3,5	1,4	0,2
setosa	4,9	3,0	1,4	0,2

Font: problema «iris» del repositori UCI (Frank i Asunción, 2010).

class	*sepal-length*	*sepal-width*	*petal-length*	*petal-width*
versicolor	6,1	2,9	4,7	1,4
versicolor	5,6	2,9	3,6	1,3
virginica	7,6	3,0	6,6	2,1
virginica	4,9	2,5	4,5	1,7

Font: problema «iris» del repositori UCI (Frank i Asunción, 2010).

En aquest algorisme, la classificació dels nous exemples es fa buscant el conjunt dels *k* exemples més propers entre un conjunt d'exemples etiquetats prèviament desats i seleccionant la classe més freqüent entre les etiquetes. La generalització es posposa fins al moment de la classificació dels nous exemples.

Aquest algorisme, en la seva forma més simple, desa en la memòria tots els exemples durant el procés d'entrenament i la classificació dels nous exemples es basa en les classes dels *k* exemples més propers. Per a obtenir el conjunt dels *k* veïns més propers, es calcula la distància entre l'exemple a classificar $x = (x_1, \ldots, x_m)$ i tots els exemples desats $x_i = (x_{i_1}, \ldots, x_{i_m})$. Una de les distàncies més utilitzades és l'euclidiana:

$$de(x, x_i) = \sqrt{\sum_{j=1}^{m} (x_j - x_{ij})^2}$$

La **distància euclidiana** és una mètrica útil en nombroses aplicacions, especialment si les magnituds són lineals i l'escala és uniforme. A més, és senzilla i ràpida de calcular. No obstant això, només és aplicable a atributs numèrics. Per a atributs nominals o binaris, s'acostuma a utilitzar la distància de Hamming, entre d'altres.

Distància de Hamming

La distància de Hamming és:

$$dh(x, x_i) = \sqrt{\sum_{j=1}^{m} \delta(x_j - x_{ij})^2}$$

en què $\delta(x_j - x_{ij})$ és la distància entre dos valors que corresponen a 0 si $x_j = x_{ij}$ i a 1 si són diferents.

Exemple d'aplicació

La taula anterior mostra un conjunt d'entrenament en què hem de classificar flors a partir de les seves propietats. En aquest exemple, aplicarem el kNN per a valors de *k* d'1 i 3, utilitzant com a mesura de distància l'euclidiana. El procés d'entrenament consisteix a desar les dades. No hem de fer res. A partir d'això, si ens arriba un exemple nou com el que mostra la taula 2, hem de calcular les distàncies entre el nou exemple i tots els del conjunt d'entrenament. La taula 3 mostra aquestes distàncies.

Taula 2. Exemple de test

class	*sepal-length*	*sepal-width*	*petal-length*	*petal-width*
setosa	4,9	3,1	1,5	0,1

Font: problema «iris» del repositori UCI (Frank i Asunción, 2010).

Taula 3. Distàncies

0,5	0,2	3,7	2,5	6,1	3,5

Per a l'1NN escollim la classe de l'exemple d'entrenament més proper, que coincideix amb el segon exemple (distància: 0,2) que té per classe *setosa*. Per al 3NN escollim els tres

exemples més propers: primer, segon i quart; amb les distàncies respectives: 0,5, 0,2 i 2,5. Les seves classes corresponen a *setosa*, *setosa* i *versicolor*. En aquest cas, també l'assignarem a *setosa* perquè és la classe més freqüent. En tots dos casos el resultat és correcte.

2.2. Arbres de decisió

Si bé l'algorisme kNN és un dels exemples més representatius del mètode basat en distàncies, els arbres de decisió ho són dels mètodes basats en regles. Aquests mètodes adquireixen regles de selecció associades a cadascuna de les classes. Atès un exemple de test, el sistema selecciona la classe que verifica algunes de les regles que determinen una de les classes.

Un arbre de decisió és una manera de representar regles de classificació inherents a les dades, amb una estructura en arbre *n*-ari que divideix les dades de manera recursiva. Cada branca d'un arbre de decisió representa una regla que decideix entre una conjunció de valors d'un atribut bàsic (nodes interns) o fa una predicció de la classe (nodes terminals).

L'algorisme dels arbres de decisió bàsic està pensat per a treballar amb atributs nominals. El conjunt d'entrenament queda definit per $S = \{(x_1, y_1), \ldots, x(x_n, y_n)\}$, en què cada component x_i correspon a $x_i = (x_{i_1}, \ldots, x_{i_m})$ en què m correspon al nombre d'atributs dels exemples d'entrenament; i el conjunt d'atributs per $A = \{a_1, \ldots, a_m\}$ en què dom(a_j) correspon al conjunt de tots els possibles valors de l'atribut a_j, i per a qualsevol valor d'un exemple d'entrenament $x_{ij} \in \mathrm{dom}(a_j)$.

El procés de construcció de l'arbre és un procés iteratiu, en què, en cada iteració, se selecciona l'atribut que fa la millor partició del conjunt d'entrenament. Per a fer aquest procés, hem de mirar la bondat de les particions que genera cadascun dels atributs i, en un segon pas, seleccionar el millor. La partició de l'atribut a_j genera $|\mathrm{dom}(a_j)|$ conjunts, que correspon al nombre d'elements del conjunt. Hi ha diverses mesures per a mirar la bondat de la partició. Una de bàsica consisteix a assignar la classe majoritària a cada conjunt de la partició, comptar quants queden ben classificats i dividir-ho pel nombre d'exemples. Una vegada calculades les bondats de tots els atributs, escollim el millor.

Cada conjunt de la millor partició passarà a ser un nou node de l'arbre. A aquest node s'arribarà per mitjà d'una regla de tipus *atribut* = *valor*. Si tots els exemples del conjunt han quedat ben classificats, el convertim en un node terminal amb la classe dels exemples. En cas contrari, el convertim en un node intern i apliquem una nova iteració al conjunt («reduït») eliminant l'atribut que ha generat la partició. En cas que no quedin atributs, el convertirem en un node terminal assignant-li la classe majoritària.

Per a fer el test, explorem l'arbre en funció dels valors dels atributs de l'exemple de test i les regles de l'arbre fins a arribar al node terminal, i donem com a predicció la classe del node terminal a què arribem.

Exemple d'aplicació

Suposem que volem fer una aplicació per a telèfons mòbils que ajudi els afeccionats a la recol·lecció de bolets i a destriar els bolets verinosos dels comestibles a partir de les seves propietats: forma i color del barret i color del tronc. La taula 4 mostra un exemple d'un conjunt d'aquestes dades, que farem servir com a conjunt d'entrenament.

Taula 4. Conjunt d'entrenament

class	***cap-shape***	***cap-color***	***gill-color***
poisonous	*convex*	*brown*	*black*
edible	*convex*	*yellow*	*black*
edible	*bell*	*white*	*brown*
poisonous	*convex*	*white*	*brown*
edible	*convex*	*yellow*	*brown*
edible	*bell*	*white*	*brown*
poisonous	*convex*	*white*	*pink*

Font: problema «mushroom» del repositori UCI (Frank i Asunción, 2010).

Per a construir un arbre de decisió a partir d'aquest conjunt hem de calcular la bondat de les particions dels tres atributs: *cap-shape*, *cap-color* i *gill-color*. L'atribut *cap-shape* ens genera una partició amb dos conjunts: un per al valor *convex* i un altre per a *bell*. La classe majoritària per al conjunt de *convex* és *poisonous* (tres elements són *poisonous* i dos elements són *edible*) i la de *bell* és *edible* (tots dos elements són *edible*). La seva bondat és bondat(*cap – shape*) = (3 + 2)/7 = 0,71. Si fem el mateix procés per a la resta d'atributs obtenim: bondat(*cap – color*) = (1 + 2 + 2)/7 = 0,71 i bondat(*gill – color*) = (1 + 3 + 1)/7 = 0,71.

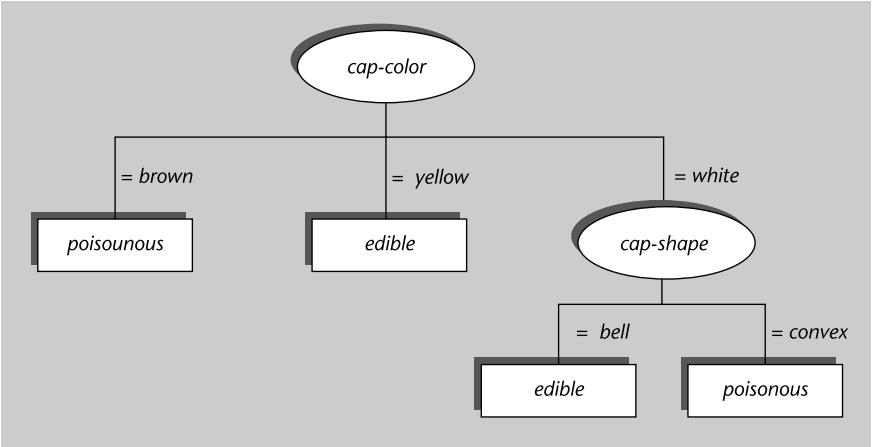
El pas següent consisteix a seleccionar el millor atribut. Hem obtingut un empat entre els tres atributs i en podem escollir qualsevol. Escollim *cap-color*. Els nodes generats pel conjunt de *brown* i *yellow* són terminals i els assignem les classes *poisonous* i *edible*, respectivament. Això és perquè els dos conjunts obtenen una bondat d'1. El node *white* ens queda amb les dades que mostra la taula 5. Aquest conjunt l'obtenim d'eliminar l'atribut *cap-color* dels exemples que tenen el valor *white* per a l'atribut *cap-color*. Tornem a iterar. La bondat de les noves particions és: bondat(*cap – shape*) = (2 + 2)/4 = 1 i bondat(*gill – color*) = (2 + 1)/4 = 0,75.

El millor atribut és *cap-shape*, que genera dos nodes terminals amb les classes *edible* per a *bell* i *poisonous* per a *convex*. La figura 1 mostra l'arbre construït en aquest procés.

Taula 5. Conjunt de la segona iteració

class	***cap-shape***	***gill-color***
edible	*bell*	*brown*
poisonous	*convex*	*brown*
edible	*bell*	*brown*
poisonous	*convex*	*pink*

Figura 1. Arbre de decisió



Per a etiquetar un exemple de test com el que mostra la taula 6 hem de recórrer a l'arbre partint de l'arrel i escollint les branques corresponents als valors dels atributs dels exemples de test. Per a aquest cas, mirem el valor de l'atribut *cap-color* i baixem per la branca que correspon al valor *brown*. Arribem a un node terminal amb classe *poisonous*, amb la qual cosa l'assignarem a l'exemple de test com a predicció. Aquesta predicció és correcta.

Taula 6. Exemple de test

class	***cap-shape***	***gill-color***	***gill-color***
poisonous	*convex*	*brown*	*black*

Font: problema «mushroom» del repositori UCI (Frank i Asunción, 2010).

Aquest algorisme té el gran avantatge de la facilitat d'interpretació del model d'aprenentatge. Un arbre de decisió ens dona una informació clara de la presa de decisions i de la importància dels diferents atributs involucrats. Per aquesta raó s'ha utilitzat molt en diversos camps, com ara el financer.

2.3. Support Vector Machines (SVM)

Un dels mètodes de classificació més utilitzats clàssicament són les màquines de vectors de suport[1], les quals estan basades en l'estadística. Aquest mètode permet clasificar objectes descrits mitjançant dades numèriques que consideren només l'existència de dues classes (com podria ser el cas del problema «mushroom» amb les classes *poisonous* i *edible*).

[(1)]En anglès, *Support Vector Machines*, SVM.

Vegeu també

En aquest apartat farem una breu introducció a les SVM lineals i no lineals, sense entrar en els detalls matemàtics de la formulació, la qual es podrà veure posteriorment a les assignatures *Aprenentatge computacional* i *Intel·ligència artificial avançada*.

2.3.1. SVM lineals

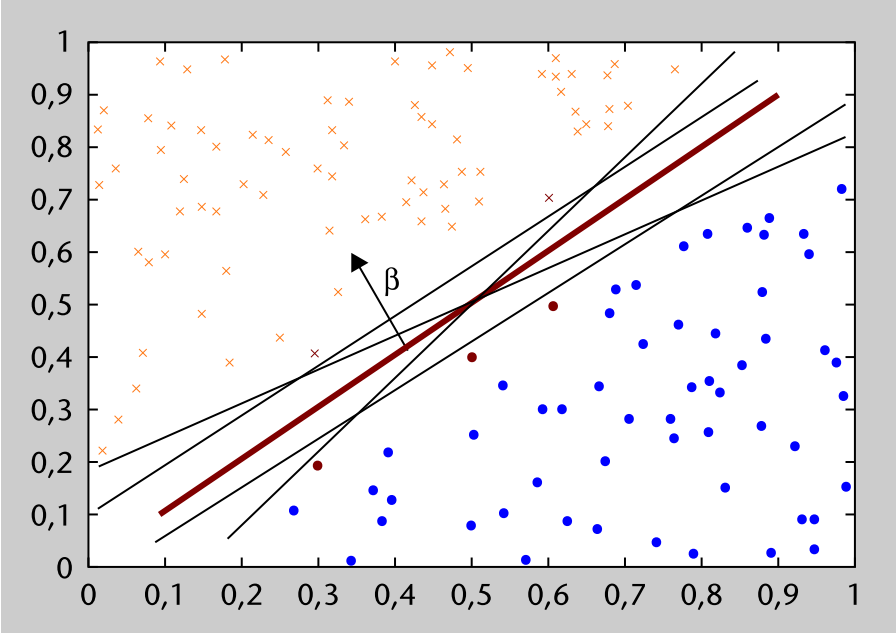
Les màquines lineals de vectors de suport es basen en la construcció d'un hiperplà en l'espai de les dades que separa els objectes que pertanyen a una classe dels que pertanyen a l'altra. Ens podem trobar amb dos casos:

1) Els objectes són linealment separables: és possible trobar un hiperplà que separi completament els objectes de les dues classes.

2) Els objectes no són linealment separables: no és possible trobar un hiperplà que separi completament els objectes de les dues classes i, per tant, cal cercar un hiperplà que separi la majoria dels objectes.

Quan les classes són linealment separables, la millor regla de classificació correspondrà a trobar l'hiperplà que defineixi un espai més ample amb els objectes. En la figura 2, es pot veure un exemple de classes linealment separables, en què es mostra un conjunt de punts que pertanyen a dues classes (creus i cercles). El problema es redueix a trobar l'hiperplà separador més adient. Noteu que hi ha infinits hiperplans que separen perfectament aquestes dues classes. En la figura, se n'ha dibuixat uns quants (hiperplans negres), però podem intuir que l'hiperplà que més interessa és aquell que deixa més distància entre cada classe i l'hiperplà separador. Per tant, l'hiperplà òptim que separa els objectes de les dues classes és el que està representat en grana en la figura 2.

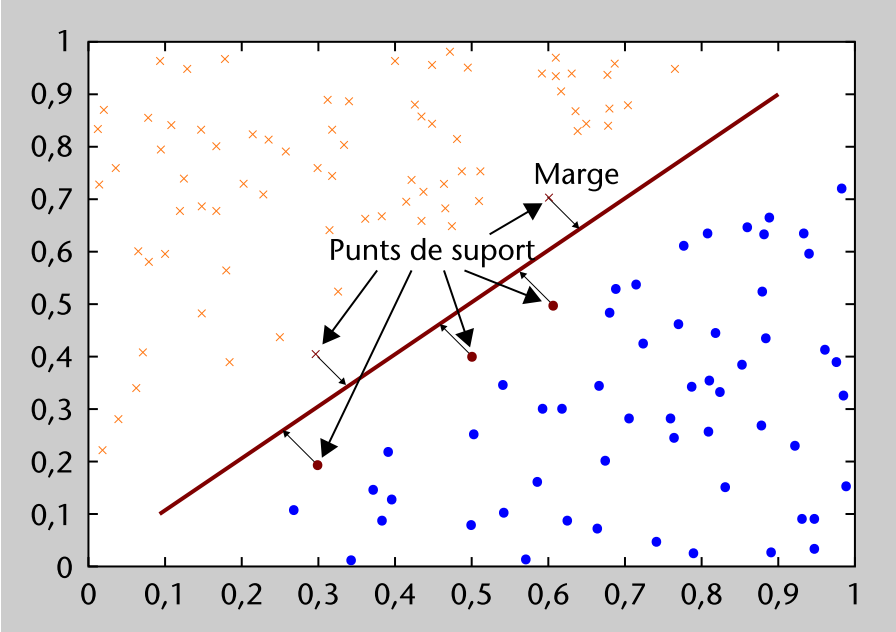
Figura 2. Exemple de classes linealment separables



Amb aquesta idea intuïtiva, definirem el concepte de **marge** com la distància entre el punt més proper de cada classe i l'hiperplà separador. Anomenarem **vectors de suport** (*support vectors*) als objectes que estan a aquesta distància de l'hiperplà.

La figura 3 mostra gràficament el concepte de vectors de suport i marge. Noteu que els paràmetres que defineixen l'hiperplà dependran exclusivament d'aquests vectors de suport, que són els que realment determinen la frontera de separació.

Figura 3. Vectors de suport i marge



Fixeu-vos que tot i que l'exemple donat és en un espai de dues dimensions, el mateix concepte és extensible a qualsevol espai de dimensions.

Quan les classes no són linealment separables, cal tenir en compte l'encavalcament entre classes. La idea és la mateixa, situar l'hiperplà de manera que la distància amb els objectes propers sigui el més gran possible. Ara, però, es deixa que alguns punts estiguin al costat incorrecte de l'hiperplà.

2.3.2. SVM no lineals

Quan les classes no són linealment separables, també hi ha una altra alternativa: la utilització de **màquines no lineals de vectors de suport**. Aquestes màquines consideren una transformació de l'espai de les dades originals en un nou espai diferent i més gran en què la separació lineal dels elements sigui possible o, almenys, es redueixi el nombre d'objectes que estaven al costat incorrecte de l'hiperplà de l'espai original.

Així doncs, es considera una funció Φ que transforma els atributs dels objectes de l'espai original a un nou espai en què s'aplica una SVM lineal (tal com s'ha descrit anteriorment).

La formulació matemàtica de les SVM permet que no calgui la definició de la funció Φ que transforma els atributs, sinó que n'hi ha prou amb definir el producte escalar de parelles d'elements transformats. La funció que defineix aquest producte és el que s'anomena *funció nucli* (kernel).

En la figura 4, podem veure un exemple de classes no linealment separables, ja que no és possible trobar a l'espai original un hiperplà que separi completament ambdues classes. No obstant això, es pot observar que les dades segueixen una distribució circular. En aquest cas, si utilitzem la funció nucli (kernel) gaussià, aleshores els objectes del nou espai transformat són linealment separables.

Kernels no lineals

Els kernels no lineals més habituals són el polinòmic i el gaussià.

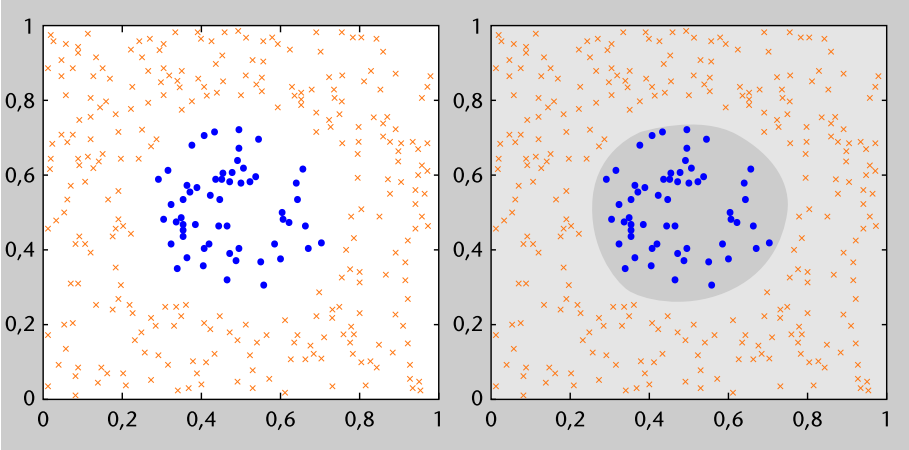
El **kernel polinòmic** es defineix:

$$\kappa(x_i, x_j) = p(\kappa_1(x_i, x_j))$$

El **kernel gaussià** es defineix:

$$\kappa(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

Figura 4. Exemple de classes no linealment separables



Hi ha dos kernels no lineals d'aplicació general que apareixen en totes les implementacions dels mètodes basats en kernels: el polinòmic i el gaussià.

2.4. Xarxes neurals

L'objectiu de les xarxes neuronals és crear un anàleg computacional a les neurones biològiques per a intentar crear intel·ligència en un ordinador. En aquest sentit, es defineixen uns elements anomenats *neurones* o simplement unitats amb connexions a altres unitats mitjançant les quals es propaguen senyals.

En aquest apartat, farem una introducció de les bases teòriques d'aquest mètode i com es relaciona amb la disciplina del Deep Learning, molt reconeguda i àmpliament aplicada durant la darrera dècada.

Vegeu també

En l'assignatura *Intel·ligència artificial avançada* s'estudia en profunditat aquesta temàtica, analitzant diverses arquitectures i exposant casos d'aplicació real com ara els classificadors d'imatges.

Una xarxa neuronal bàsica es compon dels elements següents:

- Una capa d'unitats d'entrada, que reben les variables disponibles de l'exterior per a tractar el problema: els píxels d'una imatge, la posició d'un objecte, els valors de certs productes, etc.

- Una capa d'unitats de sortida, composta per una o més unitats que produeixen una sortida a l'exterior, que és el «resultat» de la xarxa: la classe d'imatge, el valor de la tensió elèctrica per a accionar un motor, si s'ha de comprar o vendre un producte, etc.

- Una capa d'unitats ocultes, que reben connexions de la capa d'entrada i es connecten a les unitats de sortida.

- Finalment, el conjunt de connexions entre capes. En general, les connexions entre les unitats són unidireccionals.

Com podeu veure, les unitats s'organitzen en capes. En el tipus de xarxa neuronal més senzilla, la xarxa neuronal prealimentada o perceptró, les connexions sempre van de les unitats de la capa d'entrada a les de la capa oculta i d'aquí a la capa de sortida. Les connexions no tornen enrere.

Així, el funcionament del perceptró consisteix bàsicament en la **propagació** cap endavant (*forward*) dels senyals d'entrada, lògicament modificats a mesura que travessen les capes.

Quan totes les unitats d'una capa estan connectades a totes les unitats de la capa següent es diu que aquesta segona capa està **completament connectada** (*fully connected*). Al perceptró les capes són així, però en altres arquitectures no passa això.

De manera simplificada, el funcionament del perceptró és el següent:

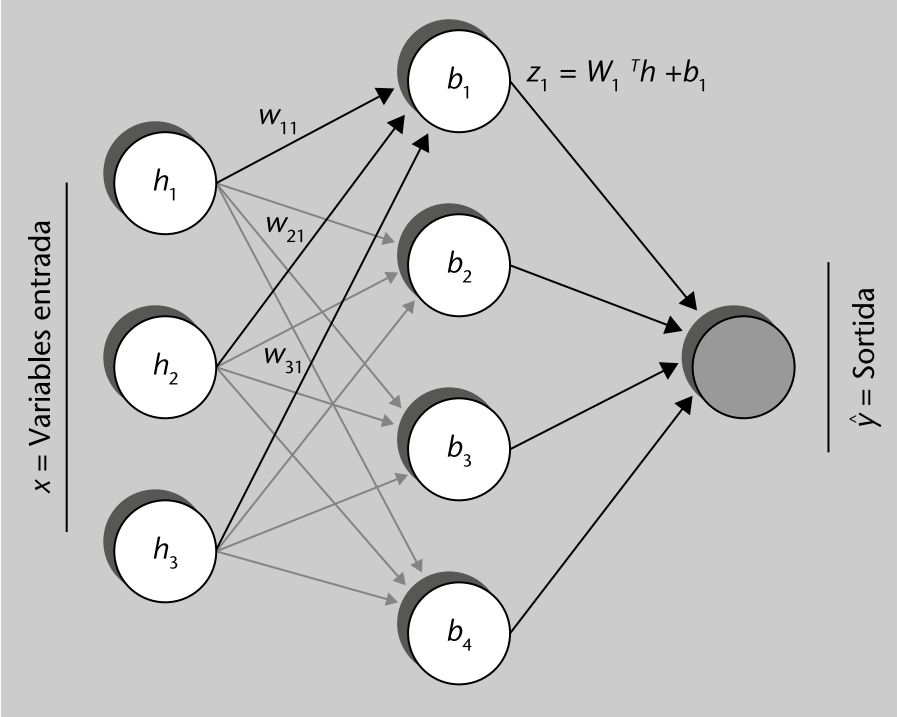
1) Les unitats d'entrada reben valors de l'exterior i s'activaran —és a dir, produiran un determinat valor a la sortida— o no en funció de l'entrada rebuda.

2) Les sortides de la capa d'entrada són, al seu torn, les entrades de la capa oculta, de manera que aquestes unitats reben un conjunt d'entrades enfront de les quals reaccionaran. Per a fer-ho, cada unitat de la capa oculta té un vector de pesos, un valor per a cada connexió entrant, que combina amb els senyals corresponents i, com a resultat, produeixen l'activació o no de la sortida de cada unitat oculta.

3) Finalment, les unitats de la capa de sortida també reben els senyals de la capa oculta, realitzen una operació amb aquests senyals i els seus propis vectors de pesos i, com a resultat, calculen la seva pròpia sortida, que serà el resultat de la xarxa.

La figura 5 mostra esquemàticament l'estructura d'una xarxa neural bàsica.

Figura 5. Xarxa neural bàsica



La manera que té una unitat de combinar les seves entrades X amb el seu vector de pesos W per a calcular la seva sortida z és donada, generalment, per l'expressió:

$$z = W^T X + b$$

on b és un valor escalar, denominat biaix (*bias*), necessari per a garantir un valor de base. El valor $z \in R$ s'utilitza, al seu torn, com a entrada per a la funció d'activació, que és la que decideix quina és la sortida final.

La funció d'activació més utilitzada en les unitats d'entrada i ocultes és la crida **ReLU**[2], que és tan senzilla com:

$$g(z) = \max(0,z)$$

És a dir, la sortida és 0 si l'entrada és negativa, i és igual a z si aquesta és positiva.

[2]De l'anglès, *Rectified Linear Unit*.

Funcions d'activació

L'elecció de la funció d'activació és clau en aquest tipus de sistemes. Depenent del problema, en podem trobar una gran diversitat: funcions d'activació lineals, logística sigmoide, etc.

2.4.1. Entrenament

Com es pot inferir, els valors W i b són els que controlen el comportament de la xarxa. Una de les característiques més potents de les xarxes neurals és que són elles mateixes les que s'aprenen els valors òptims per a aquestes variables amb la finalitat de dur a terme el seu propòsit de la millor manera possible.

Quan es treballa amb xarxes neuronals, es distingeixen dues fases:

1) Fase d'entrenament: en què la xarxa rep dades de què ha d'aprendre i en què va ajustant els pesos i biaixos, i

2) Fase d'execució: en què s'utilitza la xarxa per a dur a terme la tasca tal com l'ha après.

Com a mètode d'aprenentatge supervisat que són les xarxes neurals, l'entrenament del perceptró requereix un conjunt de dades etiquetades. Els pesos i biaixos s'inicialitzen a valors aleatoris petits (al voltant de 0,1). A continuació, es van injectant els exemples d'entrenament a la xarxa i s'analitza la diferència entre la sortida obtinguda i la sortida esperada (segons l'etiqueta associada a cada exemple).

Aquesta diferència entre la sortida obtinguda i la sortida esperada s'expressa mitjançant una funció de cost C. En xarxes amb una sola sortida, es pot utilitzar com a C la funció d'error quadràtic. L'objectiu de l'entrenament és reduir els valors de C, és a dir, la diferència entre la sortida obtinguda i l'esperada. Per això, cal ajustar gradualment els pesos i biaixos de cada capa calculant el **gradient de la funció de cost** respecte dels pesos i biaixos de la unitat de sortida:

$$\frac{\partial C}{\partial \widehat{W}}$$

on $\widehat{W} = \{W, b\}$, és a dir, s'integren pesos i biaix en un únic vector per a facilitar les operacions. Així doncs, quan es determina els $\widehat{W}$ òptims per a reduir la funció de cost C s'està ajustant la capa de sortida. No obstant això, les capes anteriors no han rebut cap ajust. Aquí entra en joc el procés denominat **propagació cap enrere** (*backpropagation*): una vegada ajustada la capa de sortida, es procedeix a ajustar la capa oculta mitjançant el mateix procediment, i així es van ajustat les capes des de la sortida cap a l'entrada, per aquest motiu es fa «cap enrere».

Gradient de funció

Els gradients d'una funció f d'n variables són les derivades d'f respecte de cadascuna de què depèn. Per exemple, si tenim $f(x, y)$, els seus gradients seran les derivades d'f respecte d'x i a y, i normalment es representen amb la forma:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}$$

D'aquesta manera s'acaben ajustant tots els paràmetres de la xarxa, fent que el conjunt de paràmetres produeixi el menor error possible amb les dades d'entrenament. Aquest **error d'entrenament** pot ser diferent de zero en funció de les dades i la complexitat del model. Un model més complex serà capaç

d'aprendre millor les dades d'entrenament, però, com es veurà més endavant, no és convenient portar aquesta idea a l'extrem per a aconseguir un error d'entrenament igual a zero.

2.4.2. Aprenentatge profund

En aquest apartat hem estudiat una xarxa neuronal senzilla, el perceptró, en què només hi ha tres capes. Una xarxa amb una sola capa oculta com aquesta és capaç d'aprendre qualsevol conjunt de dades finit, sempre que tingui un nombre suficientment gran d'unitats a la capa oculta.

El problema d'aquest plantejament, augmentar l'amplària de la capa oculta, és doble:

- És una solució computacionalment costosa, ja que pot requerir una capa amb molts milers o, fins i tot, milions d'unitats.

- Amb un sistema així només s'aprenen les combinacions de variables presents en les dades d'entrenament. Així doncs, el seu poder de generalització és molt rudimentari.

Una estratègia alternativa per a augmentar la complexitat d'una xarxa neuronal és incrementar el nombre de capes ocultes. Com a resultat, s'obté una xarxa que és capaç de generalitzar millor les noves dades, ja que a cada capa es modela un nivell d'abstracció superior que en l'anterior. Així doncs, al final la xarxa és capaç de detectar característiques més generals.

Aquesta és la idea de l'aprenentatge profund: crear xarxes neuronals amb diverses capes ocultes. Per això s'anomena «profund». Generalment, es considera profunda una xarxa amb deu o més capes.

Fins fa uns anys (al voltant de 2010) no s'havien utilitzat aquests sistemes de manera generalitzada per la dificultat i el cost computacional d'entrenar-los. No obstant això, diferents avenços teòrics i tècnics han obert les tècniques d'aprenentatge profund i han revolucionat l'IA.

3. Mètodes d'aprenentatge no supervisat

L'aprenentatge no supervisat correspon a una situació en què no es disposa d'informació del resultat que hauria de donar el sistema per a un exemple concret. Els algorismes d'agrupament són uns clars representants d'aquests mètodes i per aquest motiu en donarem dos exemples concrets.

3.1. Algorismes d'agrupament

La tasca d'agrupament de les dades és una tasca no supervisada, ja que les dades que es proporcionen al sistema no porten associada cap etiqueta o informació afegida per un revisor humà. Per contra, és el mateix mètode d'agrupament el que ha de descobrir les noves classes o grups a partir de les dades rebudes.

Sovint l'agrupament de les dades precedeix a la classificació de noves dades en algun dels grups obtinguts en l'agrupament. Per aquesta raó, l'agrupament i la classificació de dades estan estretament relacionats.

Una de les característiques més importants dels algorismes d'agrupament és que permeten organitzar dades que, en principi, no sap o pot classificar, evitant criteris subjectius de classificació, la qual cosa produeix una informació molt valuosa a partir de dades desorganitzades.

Una característica comuna a gairebé tots els algorismes d'agrupament és que no són capaços de determinar per si mateixos el nombre de grups idoni, sinó que cal fixar-lo per endavant o utilitzar algun criteri de cohesió per a saber quan cal detenir-se (en el cas dels jeràrquics). En general, això requereix provar amb diferents nombres de grups fins a obtenir uns resultats adequats.

A continuació, es presenten breument un parell dels algorismes d'agrupament més estudiats.

3.1.1. *k-means*

L'algorisme d'**agrupament k-mitjanes**[3] busca una partició de les dades tal que cada punt estigui assignat al grup amb el centre (anomenat *centroide*, ja que no necessàriament ha de ser un punt de les dades) més proper. Se li ha d'indicar el nombre *k* d'agrupaments (o clústers) que volem, ja que per si mateix no és capaç de determinar-lo.

[3] *k-means*, en anglès.

En essència, l'algorisme és el següent:

1) Triar *k* punts a l'atzar com a centroides inicials. No necessàriament han de pertànyer al conjunt de dades, encara que les seves coordenades han d'estar en el mateix interval.

2) Assignar cada punt del conjunt de dades al centroide més proper, i formar així *k* grups.

3) Recalcular els nous centroides dels *k* grups, que estaran en el centre geomètric del conjunt de punts del grup.

4) Tornar al pas 2 fins que les assignacions a grups no variïn o s'hagin superat les iteracions previstes.

Si bé es tracta d'un algorisme senzill i ràpid, com que només té en compte la distància als centroides pot fallar en alguns casos (núvols de punts de diferents formes o densitats), ja que en general tendeix a crear esferes de mida similar que fan la partició de l'espai.

La figura 6 mostra un exemple de partició de dades produït per l'algorisme d'un determinat conjunt etiquetat segons cinc categories diferents (corresponents a les diferents formes dels punts). Es pot observar que, per exemple, alguns punts del grup superior dret (triangles) s'han assignat al superior esquerre (cercles) per a estar més propers al centroide, sense tenir en compte l'espai que els separa. El mateix problema es dona en altres punts. A més, el resultat pot variar d'una execució a una altra, ja que depèn dels centroides generats aleatòriament en el primer pas de l'algorisme.

Figura 6. Partició d'un conjunt de dades mitjançant *k-means*.

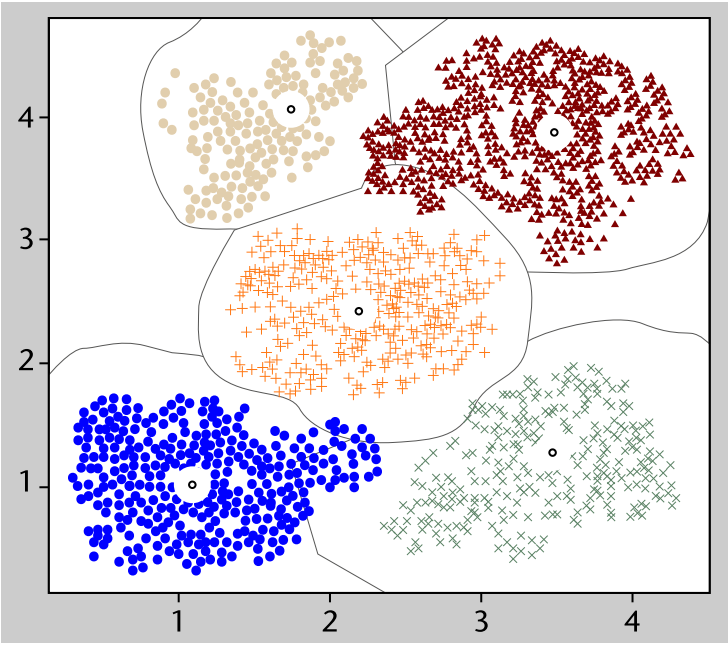


Figura 6

Els punts negres envoltats d'una zona blanca representen els centroides dels grups obtinguts.

3.1.2. *Fuzzy c-means*

Resulta lògic pensar que, en *k*-mitjanes, un punt que estigui al costat del centroide estarà més fortament associat al seu grup que un punt que estigui en el límit amb el grup veí. L'**agrupament difús** (*fuzzy clustering*) representa aquest grau diferent de vinculació fent que cada dada tingui un grau de pertinença a cada grup, de manera que un punt al costat del centroide pot tenir 0,99 de pertinença al seu grup i 0,01 al veí, mentre que un punt al costat del límit pot tenir 0,55 de pertinença al seu grup i 0,45 al veí. Això és extensible a més de dos grups, entre els quals es repartirà la pertinença de les dades. Es pot dir que l'**agrupament discret** (no difús) és un cas particular del difús en què els graus de pertinença són 1 per al grup principal i 0 per als restants.

L'algorisme *fuzzy c-means* (*c*-mitjanes difús) és pràcticament idèntic a l'algorisme *k*-mitjanes vist anteriorment. Les diferències principals són:

- Cada dada x té associat un vector de k valors reals que indiquen el grau de pertinença $m_x(k)$ d'aquesta dada a cadascun dels k grups. El grau de pertinença d'un punt a un grup depèn de la seva distància al centroide corresponent. Habitualment, la suma dels graus de pertinença d'una dada és igual a 1.

- En lloc de crear k centroides aleatòriament, s'assigna el grau de pertinença de cada punt a cada grup aleatòriament i, després, es calculen els centroides a partir d'aquesta informació.

- El càlcul dels centroides està ponderat pel grau de pertinença de cada punt al grup corresponent $m_x(k)$.

A grans trets, els avantatges i inconvenients de *c*-mitjanes difús són els mateixos que els de *k*-mitjanes: simplicitat, rapidesa, però no determinisme i excessiva dependència de la distància dels punts als centroides, sense tenir en compte la densitat de punts de cada zona (per exemple, espais buits). Es tracta d'un algorisme especialment utilitzat en el processament d'imatges.

4. Mètodes d'aprenentatge per reforç

L'aprenentatge per reforç es pot considerar com el tipus d'aprenentatge de què les diferents espècies vives han tret profit principalment al llarg de milions d'anys a la natura. Com s'ha comentat abans, aquest tipus d'aprenentatge es caracteritza per no tenir un criteri explícit i específicament definit per a avaluar si la resposta que dona un sistema és correcta o no, sinó una retroalimentació que rep el sistema des de l'entorn en què interactua. Aquesta retroalimentació dona a conèixer al sistema el resultat de les seves interaccions i el sistema l'avalua seguint uns criteris que permeten identificar el grau d'èxit que ha tingut en el seu propòsit inicial.

Aquesta analogia fa que molts dels algorismes per reforç existents estiguin considerats dins la categoria d'**algorismes bioinspirats**.

Com a algorisme representatiu de la tècnica d'aprenentatge per reforç, en aquesta secció estudiarem en primer lloc els mètodes *Q-learning* i, a continuació, un dels algorismes bioinspirats que ha tingut més popularitat i aplicacions a les darreres dècades, els algorismes genètics.

4.1. Mètodes *Q-learning*

En l'aprenentatge per reforç, l'agent o sistema ha d'aprendre i actuar, guiar el seu aprenentatge en funció de les recompenses que rep com a conseqüència de les seves accions. Les recompenses poden ser de naturalesa diversa, tot i que sempre orientades a premiar l'avenç cap a l'objectiu de l'agent: punts aconseguits en un joc, quilòmetres conduïts cap a la destinació sense accidents, peces acoblades per un robot, etc.

De manera més formal, un sistema d'aprenentatge per reforç es compon dels elements següents:

- **Un conjunt d'estats del problema, *S***. Al joc del tres en ratlla, per exemple, *S* serà tots els estats possibles de creu, cercle o casella buida, un total de 3^9 estats possibles, tot i ser un joc tan senzill. No obstant això, el nombre d'estats possibles es dispara ràpidament a unes 10^{40} posicions vàlides per als escacs o unes 10^{100} per al Go. Fora de l'àmbit dels jocs, el nombre d'estats possibles és impossible de calcular i, a la pràctica, infinit: quants estats possibles hi ha per al problema de la conducció automàtica? Quantes combinacions de vehicles, combinacions ambientals, carretera, estat del vehicle, vianants, etc.?

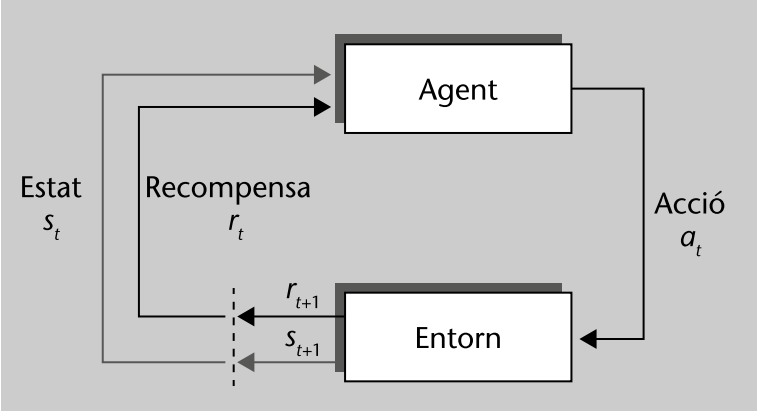
- **Un conjunt de possibles accions**, ***A***. En un joc serien els possibles moviments. Amb un robot A són els moviments que pot executar; en un sistema de conducció automàtica, les diferents accions que pot ordenar al cotxe: accelerar, frenar, girar, etc.

- **Una funció de recompensa**, ***R***, que retorna un nombre real en recompensa per l'acció presa per l'agent en un estat determinat, és a dir, R: $S \times A \rightarrow R$. És molt important destacar que la recompensa depèn de l'acció presa i de l'estat actual, ja que de vegades serà bo que el cotxe acceleri, però altres vegades no.

- En teoria, un agent d'un sistema d'aprenentatge per reforç (*reinforcement learning*, RL) s'hauria d'aprendre una **taula de recompenses** $Q = S \times A$ per a saber perfectament quina és l'acció que s'ha d'aplicar en cada estat del sistema. No obstant això, per a gairebé qualsevol problema S té una grandària infinita, per la qual cosa és impossible emmagatzemar aquest coneixement en una taula. Per aquest motiu, s'han proposat diferents mètodes per a aprendre aproximacions raonablement bones a Q, mètodes que reben el nom de *Q-learning*.

La figura 7 resumeix els elements d'un sistema d'aprenentatge per reforç que s'acaben d'explicar.

Figura 7. Diagrama de control per a un sistema d'aprenentatge per reforç



Tot i que hi ha diverses estratègies per a *Q-learning*, els sistemes que utilitzen xarxes neuronals profundes estan obtenint grans èxits en diferents àmbits, com el sistema DQN, que aprèn a jugar autònomament a jocs de consola; el sistema AlphaGo, que va vèncer al campió mundial de Go, o nombrosos sistemes de conducció automàtica i robòtica, entre d'altres.

4.2. Algorismes genètics

Els algorismes genètics són algorismes de cerca estocàstics inspirats en els fenòmens naturals d'herència genètica i que el millor és el que sobreviu (supervivència de les espècies). En una població d'individus, les noves generacions

estan més adaptades al medi que les precedents i, en mitjana, les noves poblacions seran més ràpides, amb un grau de mimetisme més gran, etc., que les anteriors perquè això és el que els permet sobreviure.

L'analogia amb el procés biològic de l'evolució dirigeix tots els passos dels algorismes genètics. Així, considerarem poblacions d'individus representats pels seus cromosomes (cromosomes formats per gens) i, atès un cromosoma, considerarem el seu creuament amb un altre cromosoma o la seva mutació. Els creuaments i les mutacions portaran a noves generacions de poblacions.

Com tots els mètodes de cerca, els algorismes genètics permeten trobar una solució a un problema donat. Tanmateix, a causa que són algorismes estocàstics, normalment no troben la millor solució del problema sinó que en troben una que aproxima la solució òptima.

4.3. Aplicació i implementació

Per a poder aplicar aquest mètode a un problema concret hem de començar trobant una manera de codificar les diverses alternatives que s'han de considerar en la solució (quines són les possibles solucions d'un problema). Per exemple, si considerem el problema del camí mínim entre un parell de poblacions, les alternatives que cal considerar són tots els camins que connecten les dues poblacions.

La codificació de les alternatives es fa sobre la base d'una seqüència de símbols de longitud finita. Als símbols que es fan servir per a la codificació se'ls anomena *gens* i a la seqüència se l'anomena *cromosoma*. Al conjunt de gens se'ls denomina *pool*. Així, tenim que cada possible alternativa serà un cromosoma format per gens.

A més de seleccionar una representació de les alternatives, necessitem una funció que avaluï les diferents alternatives de manera que la millor alternativa (la solució) tingui el millor valor. Aquesta funció és l'anomenada *funció d'avaluació* (*fitness function*). En el cas del camí mínim, podem fer servir la longitud del camí com a funció (o la inversa de la longitud del camí si volem una funció per a maximitzar).

A partir d'aquests elements, els algorismes genètics cerquen la solució en un procés iteratiu que consisteix a crear una població de cromosomes (un conjunt de possibles solucions), avaluar els cromosomes de la població mitjançant la funció d'avaluació i seleccionar els millors cromosomes d'aquesta població. Aquests cromosomes seran la llavor de la nova població que es crearà en el pas d'iteració següent. L'esquema general es presenta de la forma següent.

```
funcio Algorismes genètics retorna cromosoma es
    inicialitza la població de cromosomes
```

```
    valua tots els cromosomes de la població
    selecciona el millor cromosoma
    mentre no se satisfà el criteri d'acabament fer
        crea una nova població de cromosomes
        avalua tots els cromosomes de la població
        selecciona el millor cromosoma
    fmentre
    retorna el millor cromosoma
ffuncio
```

A continuació, descriurem amb més detall aquesta funció. Considerarem que la població conté m individus i que cada individu està format per una seqüència de longitud n.

Denotarem el conjunt de gens per G. Així, un cromosoma c satisfà $c \in G^n$. La població en la iteració k-èssima la denotem per $p^{(k)}$ i serà de la forma $p^{(k)} = \{c_1^{(k)}, \ldots, c_m^{(k)}\}$ on $c_1^{(k)} \in G^n$. Quan tenim una població concreta p_{ij} serà el gen j-èssim de l'individu i-èssim.

```
funcio Algorismes genètics (f: funció d'avaluació) retorna cromosoma es
    k:=1;
    p(k):=inicialitza();
    av:=avalua(f,p(k));
    m(k):=selecciona_millor(av, p(k))
    mentre no s'ha d'acabar (k, m(k), m(k-1)) fer
        k:=k+1;
        p(k):=crea_nova_poblacio (av, p(k-1));
        av:=avalua(f,p(k));
        m(k):=selecciona_millor(av, p(k));
    fmentre;
    retorna m(k);
ffuncio;
```

En aquest esquema, `av` correspon a les avaluacions dels cromosomes, i `m(k)` és el millor cromosoma de la població.

Ara descrivim cadascun dels passos que ens apareixen en aquest procés.

1) Inicialització: S'ha de crear el conjunt de cromosomes inicial que correspondrà a la primera població $p^{(1)}$. Normalment es fa de manera aleatòria. Això és, per a cada $x^{(k)}$) es defineix una seqüència de dimensió n amb gens de G triats a l'atzar.

2) Avaluació: Per a cadascun dels cromosomes de la població, es dona una valoració del punt fins on és bona la solució corresponent. L'avaluació reconstruirà l'alternativa a partir del cromosoma i després li aplicarà la funció d'avaluació. En el cas del camí mínim, a partir d'una llista de bits, reconstruirà el camí i després s'aplicarà la funció seleccionada per a indicar el punt fins on és bo aquest camí.

3) Selecció: L'avaluació dels cromosomes permet triar quin és el millor de tots els que hi ha a la població.

4) Creació: Utilitzant la informació codificada en els cromosomes i usant l'avaluació de cadascun es crea una nova població. El procés de construir els nous cromosomes és una analogia de l'evolució en el medi natural. S'utilitzen operacions de creuament i mutació.

Seguidament es donen els algorismes per a cadascuna d'aquestes funcions.

```
funcio inicialitza retorna població es per i:=1 fins m fer
    per j:=1 fins n fer
        pij:=tria gen aleatòriament de (G)
    fper;
fper;
retorna p;
ffuncio;

funcio avalua (f: funció, p: població) retorna avaluacions es
    per i:=1 fins m fer
        avi:=f(pi);
    fper;
    retorna av;
ffuncio;

funcio selecciona_millor (av:avaluacions,p:població)
    retorna cromosoma es
        millor:=p1; av_millor:=av1;
        per i:=2 fins m fer
            si (av_millor ≤ avi) llavors millor:=pi; av_millor:=avi; fsi;
    fper;
    retorna millor;
ffuncio;
```

Queden pendents les operacions de construir la nova població, que es considera més avall, i la funció que decideix si s'ha d'acabar el procés d'iteració. Per a saber si el procés ha d'acabar, podem tenir en compte el nombre d'iteracions

que volem fer (si k sobrepassa un determinat valor) o si la solució convergeix: si la diferència entre les avaluacions de dues solucions consecutives més bones és prou petita ($\| m^{(k)} - m^{(k-1)} \| < \varepsilon$ per a un ϵ donat).

Per a construir la nova població a partir de la població actual hi ha moltes alternatives. A continuació, se'n dona una en què la construcció està dividida en dues etapes. Primer es determinen els cromosomes que entren a formar part de la nova població i després es construirà la nova població operant amb els cromosomes (farem creuaments i mutacions dels cromosomes que hi prenen part).

```
funcio crea_nova_població (av:avaluacions,p:població)
                        retorna avaluacions es
    total_av:=0;

    per i:=1 fins m fer
    total_av:=total_av+avi;
    fper;

    pr_acc0:=0;

    per i:=1 fins m fer
    pr_acci:=pr_acci-1 + avi/total_av;
    fper;

    per i:=1 fins m fer
        r:=valor aleatori entre 0 i 1;
        p'i:=selecciona pi tal que pr_acci-1<r≤pr_acci;
    fper;

    per i:=1 fins m fer
    p''i:=genera(p'i,p');
    fper;

    retorna p'';
ffuncio;
```

La funció calcula per a cada cromosoma la probabilitat de passar a formar part de la nova població. Aquesta probabilitat es defineix com $\texttt{av}_i/\texttt{total\_av}$ on `total_av` és la suma de totes les avaluacions. Noteu que aquest valor es pot entendre com una probabilitat perquè $\sum_{j \leq i} \texttt{av}_i/\texttt{total\_av} = 1$ i, a més, té sentit ja que la probabilitat és més gran com més ben avaluat ha estat el cromosoma. Tanmateix, en lloc de guardar la probabilitat guardem els acumulats de les probabilitats (això està a $\texttt{pr\_acc}_i$) ja que així és més fàcil seleccionar

després un cromosoma donat un nombre aleatori entre zero i u. Podem veure que en la funció la selecció dels nous cromosomes es fa primer seleccionant aquest valor aleatori i després utilitzant-lo per a seleccionar el cromosoma p_i tal que el valor aleatori està entre $\texttt{pr\_acc}_{i-1}$ i $\texttt{pr\_acc}_i$. Això es fa m vegades amb la finalitat d'obtenir una població amb m cromosomes.

Ara, hem de modificar els cromosomes seleccionats per tal que la nova població no sigui com l'anterior, i que hi hagi un **intercanvi genètic** entre els diferents individus. Així, a cada cromosoma de la població seleccionada (p') li apliquem la funció $\texttt{genera(p'}_i\texttt{,p')}$ que ens construeix el nou cromosoma a partir del seleccionat i de la població p'. Per a fer aquesta construcció hi ha diverses alternatives. Aquí considerem quatre tipus d'operacions per a aconseguir l'intercanvi genètic:

- **Creuament simple**: intercanvia els gens d'un cromosoma amb els d'un altre cromosoma a partir d'una posició donada.

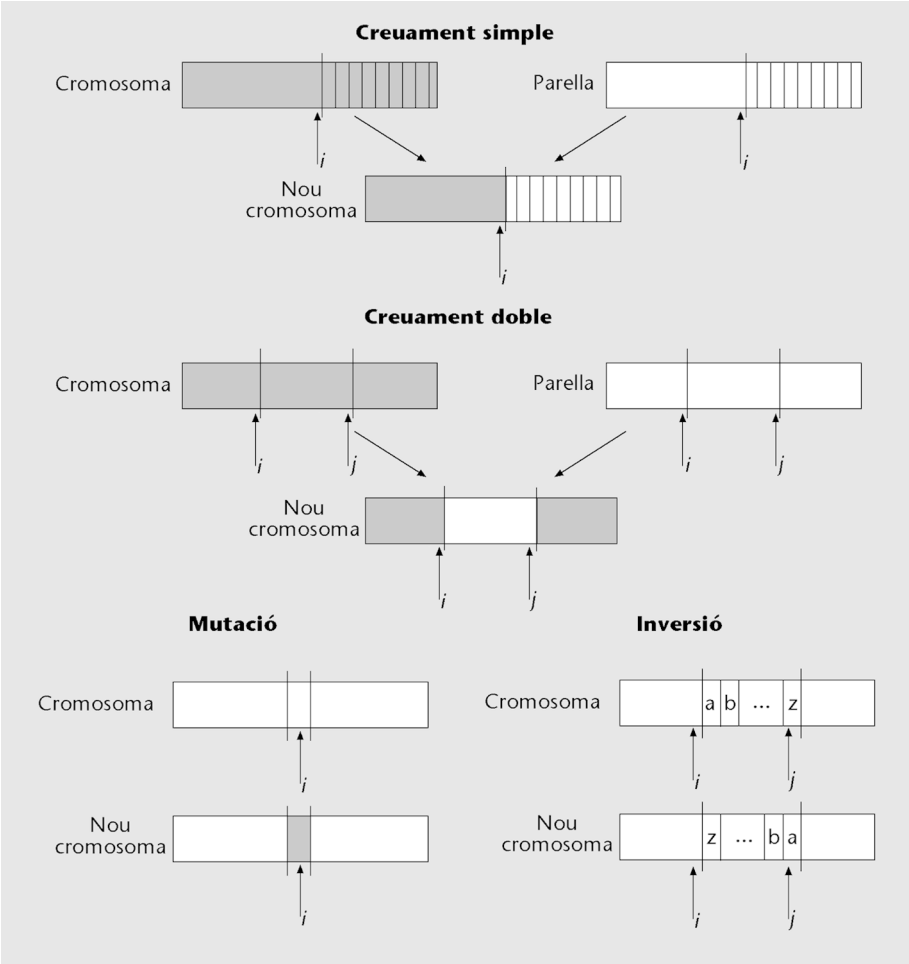
- **Creuament doble**: intercanvia una part dels gens d'un cromosoma amb els d'un altre cromosoma.

- **Mutació**: un determinat gen d'un cromosoma és substituït per un altre gen del *pool*.

- **Inversió**: en un cromosoma, una subseqüència de gens és invertida.

A continuació, es donen les representacions gràfiques i les implementacions d'aquestes operacions:

Figura 8




```
funcio creuament_simple (c:cromosoma,p:població)
                    retorna cromosoma es
    parella:=selecciona cromosoma de la població(p);
    i:=selecciona posició entre 1 i m;
    per k:=i+1 fins m fer
        ck:=parellak;
    fper;
    retorna c;
ffuncio;

funcio creuament_doble (c:cromosoma,p:població)
                      retorna cromosoma es
    parella:=selecciona cromosoma de la població(p);
    i:=selecciona posició entre 1 i m;
    j:=selecciona posició entre i i m;
    per k:=i+1 fins j fer
        ck:=parellak;
    fper;
    retorna c;
ffuncio;
```

```
funcio mutació (c:cromosoma) retorna cromosoma es
    i:=selecciona posició entre 1 i m;
    g:=selecciona gen (G);
    ci:=g;
    retorna c;
ffuncio;

funcio inversió (c:cromosoma) retorna cromosoma es
    i:=selecciona posició entre 1 i m;
    j:=selecciona posició entre i i m;
    per k:=i+1 fins j fer
        ck:=c(i+1)+(j-k);
    fper;
    retorna c;
ffuncio;
```

A partir d'aquestes funcions es pot definir la funció `genera` que havia quedat pendent. La funció triarà per a cada cromosoma una de les funcions de modificació genètica de manera aleatòria. La tria es basa en probabilitats de triar cadascuna de les operacions (les anomenem `pr_creuament_simple`, `pr_creuament_múltiple`, `pr_mutacio`, `pr_inversió`). La suma d'aquestes probabilitats ha de ser menor o igual que 1 (si la suma d'aquestes probabilitats és α i és menor que 1 voldrà dir que un cromosoma no serà modificat amb probabilitat 1 – α). La definició de la funció és la que presentem a continuació:

```
funcio genera (c:cromosoma, p:població) retorna cromosoma es
    constants pr_creuament_simple, pr_creuament_multiple;
    constants pr_mutacio, pr_inversio;
    r:=valor aleatori entre 0 i 1;
    si r ≤ pr_creuament_simple llavors
        retorna creuament_simple(c,p);
    sino si r ≤ pr_creuament_simple+pr_creuament_multiple llavors
        retorna creuament_doble(c,p);
    sino si r ≤ pr_creuament_simple+pr_creuament_multiple+pr_mutació llavors
        retorna mutació(c);
    sino si r ≤ pr_creuament_simple+pr_creuament_multiple+pr_mutacio
            +pr_inversio
    llavors
        retorna inversió(c);
    sino retorna c;
    fsi;
ffuncio;
```

5. Partició de dades i protocols de validació

Quan es duu a terme un procés d'aprenentatge supervisat, especialment en problemes de classificació, és important establir un protocol per a validar com de bé el nostre sistema ha après un conjunt de dades.

La validació més simple, coneguda com a **validació simple**, consisteix a dividir el conjunt de dades en dues: un anomenat d'entrenament i l'altre de test. Es construeix el model del conjunt d'entrenament i, tot seguit, es classifiquen els exemples de test amb el model generat a partir del d'entrenament. Una vegada obtingudes les prediccions, s'aplica alguna de les mesures d'avaluació com les que es descriuen en el subapartat següent. Una qüestió important que cal tenir en compte quan dividim un conjunt de dades en els conjunts d'entrenament i test és mantenir la proporció d'exemples de cada classe en tots dos conjunts.

En el cas que tinguem un algorisme d'aprenentatge que necessita fer ajustos de paràmetres, es divideix el conjunt d'entrenament en dos conjunts: un d'entrenament pròpiament dit i un altre de validació. S'ajusten els paràmetres entre el conjunt d'entrenament i el conjunt de validació i, una vegada trobats els òptims, ajuntem el conjunt d'entrenament amb el de validació i generem el model amb l'algorisme d'aprenentatge. La validació la fem del conjunt de test. Un error de mètode força usual és utilitzar el conjunt de test per a fer l'ajust dels paràmetres. Això no és estadísticament correcte. Una de les tècniques més utilitzades per a ajustar els paràmetres són els algorismes d'optimització com ara els algorismes genètics, que acabem d'estudiar en el subapartat anterior.

Un problema que s'ha detectat en l'ús de la validació simple és que segons el conjunt de dades que tinguem pot variar molt el comportament del sistema en funció de la partició d'exemples que hàgim considerat. És a dir, diferents particions d'exemples condueixen a diferents resultats. En molts casos, els investigadors deixen disponibles els conjunts ja dividits en l'entrenament i el test perquè les comparacions entre els sistemes siguin més fiables.

Per a minimitzar aquest efecte, se sol utilitzar la **validació creuada**[4] en lloc de la simple. Aquest tipus de validació consisteix a dividir un conjunt en k subconjunts.

(4)En anglès, *cross-validation* o *k-fold cross-validation*.

(5)En anglès, *leave-one-out*.

Tot seguit, es fan *k* proves utilitzant en cadascuna un subconjunt com a test i la resta com a entrenament. A partir d'això es calcula la mitjana i la desviació estàndard dels resultats. Amb això podem veure millor el comportament del sistema. Un valor molt utilitzat de la *k* és 10. Una variant coneguda de la validació creuada és el **deixar-ne un a fora**[5]. Aquest cas és com l'anterior definint

la *k* com el nombre d'exemples del conjunt total. Ens quedaria el conjunt de test amb un únic exemple. Aquest mètode no se sol utilitzar a causa del seu alt cost computacional.

Un dels problemes que ens trobem quan descrivim les mesures d'avaluació és la gran diversitat de mesures que s'utilitzen a les diferents àrees d'investigació. Aquestes mesures solen tenir en compte les diferents peculiaritats dels problemes dels diferents camps. Un altre problema és que de vegades aquestes mesures d'avaluació reben noms diferents en funció de l'àrea. En aquest apartat, veurem les mesures que s'utilitzen en els contextos de la classificació, la recuperació de la informació i la teoria de la detecció de senyals.

La majoria de les mesures d'avaluació es poden expressar en funció de la matriu de confusió o taula de contingència. La matriu de confusió conté una partició dels exemples en funció de la seva classe i predicció. La taula 7 mostra el contingut de les matrius de confusions per al cas binari. A tall d'exemple, la cel·la *positiu vertader* correspon al nombre d'exemples del conjunt de test que tenen tant la classe com la predicció positives. Els *falsos positius* també es coneixen com a *falses alarmes* o *error de tipus I*; els *falsos negatius* com a *error de tipus II*; els *positius vertaders* com a *èxits*; i els *negatius vertaders* com a *rebutjos correctes*.

Taula 7. Matriu de confusió

		classe real	
		positiva	negativa
predicció	positiva	positiu vertader (*tp*)	fals positiu (*fp*)
	negativa	fals negatiu (*fn*)	negatiu vertader (*tn*)

L'**error** o **ràtio d'error** mesura el nombre d'exemples que s'han classificat incorrectament del total d'exemples.

Es pretén que els algorismes d'aprenentatge tendeixin a minimitzar aquesta mesura. La fórmula és:

$$\text{error} = \frac{fp + fn}{tp + fp + tn + fn}$$

L'**exactitud**[6] correspon als exemples que s'han classificat correctament del total d'exemples. Aquesta mesura és la complementària de l'anterior. La fórmula correspon a:

(6)*Accuracy*, en anglès.

$$\text{exactitud} = \frac{tp + tn}{tp + fp + tn + fn}$$

La **precisió** o **valor predictiu positiu**[7] correspon als exemples positius ben classificats del total d'exemples amb predicció positiva. La fórmula és:

(7)*Precision*, en anglès.

$$\text{precisió} = \frac{tp}{tp + fp}$$

La **sensibilitat**[8] correspon als exemples positius ben classificats del total d'exemples positius. La fórmula és:

[8] *Recall*, en anglès.

$$\text{sensibilitat} = \frac{tp}{tp + fn}$$

El conjunt de precisió i sensibilitat es pot veure com una versió estesa de l'exactitud. Aquestes mesures venen del camp de la recuperació de la informació, en què el valor de *tn* és molt superior a la resta i esbiaixa les mesures d'avaluació.

D'allà també ens arriba la mesura *F1*, proposta el 1979 per van Rijsbergen, que combina les dues anteriors:

$$F1 = 2 \times \frac{\text{precisió} \times \text{sensibilitat}}{\text{precisió} + \text{sensibilitat}} = \frac{2 \times tp}{2 \times tp + fn + fp}$$

Aquesta mesura descarta els elements negatius a causa del biaix produït per la diferència entre el nombre d'exemples negatius i el de positius. En l'àmbit de la recuperació d'informació, arriba a ser molt crític.

Hi ha altres mesures que s'utilitzen a partir de la matriu de confusió, com pot ser l'**especificitat**, que se solen utilitzar en l'àrea de la teoria de detecció de senyals:

$$\text{especificitat} = \frac{tn}{fp + tn}$$

Totes les mesures d'aquest subapartat s'han descrit a partir del problema binari, però també es poden utilitzar en problemes multiclasse.

Bibliografia

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. EUA: Springer.

Duda, R. O.; Hart, P. E.; Stork, D. G. (2001). *Pattern Classification* (2a. ed.). EUA: John Wiley and Sons, Inc.

Frank, A.; Asuncion, A. (2010). *UCI Machine Learning Repository* (Disponible en línia). EUA: University of California, School of Information and Computer Science. [Data de consulta: 28 de març de 2019.] <http://archive.ics.uci.edu/ml>

Goodfellow, I.; Bengio, Y.; Courville, A. (2016). *Deep Learning*. EUA: MIT Press.

Mitchell, T. M. (1997). *Machine Learning*. EUA: McGraw-Hill.